

GROWTH OF FUNCTIONS AND RECURRENCE RELATIONS

Document 2 contains more examples

Chapters 1.1.4-1.1.6, and 11.1 of GT

DISCLAIMER: Report an error / typo to the author so that he resolves it.
These notes are NOT to be distributed outside of NJIT without the expressly written permission of A. Gerbessiotis

Growth of Functions

Asymptotic Notation

The analysis of the performance of algorithms is done for large problem sizes and this is known as asymptotic analysis.

If problem size is n and if the performance measure is for example running time $T(n)$, asymptotic analysis is considering the running time asymptotically for large n , in other words, as $n \rightarrow \infty$. Thus what happens to $T(n)$ for $n = 1$, $n = 10$, $n = 100$ is (asymptotically) irrelevant. The asymptotic growth of the running time is of interest.

Note that problem size n is a non-negative integer. The discussion below however can extend to the domain of positive real numbers. Deriving the asymptotic growth of $T(n)$ is equivalent to identifying the dominant function term in $T(n)$. Thus for example, if $T(n)$ is such that $T(n) = 2n^3 + 1000n^2 + 10000$, we care only that $T(n)$ has a cubic growth contributed by the n^3 term. The multiplicative constant two is irrelevant as both $2n^3$ and n^3 are cubic functions. The remaining terms are asymptotically negligible $\lim_{n \rightarrow \infty} 1000n^2/2n^3 = 0$ or $\lim_{n \rightarrow \infty} 10000/2n^3 = 0$ for $n \rightarrow \infty$.

A compact way to express this is by using appropriate asymptotic notation that “hides” lower order terms but also multiplicative constants in the dominant term. In Mathematics we could have used the notation $T(n) \approx n^3$ which is equivalent to saying that $\lim_{n \rightarrow \infty} T(n)/n^3 = 1$. We are going to propose a more elegant approach.

Moreover establishing the asymptotic performance of one or more algorithms, performance comparison becomes more intricate. When we compare the performance of two algorithms, we have available through asymptotic analysis only their asymptotic performance. Thus we perform an asymptotic comparison rather than a regular comparison of their performance. Thus two algorithms, one with $T_1(n) = 2n^3 + 1000n^2 + 10000$ and the other $T_2(n) = n^3$ are to be declared as having the same asymptotic performance. Their running time are asymptotically equal even if $T_1(n) > T_2(n)$ for all $n > 0$. This is because $T_1(n)$, $T_2(n)$ are both cubic functions. The “best algorithm” becomes “the asymptotically best algorithm” or the “algorithm with the best asymptotic performance”. And we prefer to use the term efficient for best. And efficient might be further refined to time efficient if $T(n)$ is to be used for asymptotic comparison, or space efficient if $S(n)$ is to be used, or even comparison efficient if we talk about a comparison-based algorithm and $C(n)$ is studied.

Moreover with asymptotic efficiency one needs to be aware of its limitations (not limits). For example if we say a person A is of height at most 6ft and a person B is of height at most 7ft, it might be wrong to conclude B is taller than A. It might be the case that B is 5ft10in and A 5ft10in.

Asymptotic Notation

Introduction

We introduce **asymptotic notation** that we will use to describe and compare the asymptotic performance of algorithms. We do this on the understanding that exact comparison of two algorithms is usually futile. In practice, compiler settings, cpu architectures, including memory speed and memory hierarchies (L1, L2, L3 caches) might misclassify the relative performance of two algorithms. For the previous example we will use the notation $T_1(n) = \Theta(n^3)$ or $T_2(n) = \Theta(n^3)$ to denote that the two functions are cubic. Sometimes we may even write $T_1(n) \in \Theta(n^3)$ or $T_2(n) \in \Theta(n^3)$. Either way is more elegant than $T_1(n) \approx n^3$. This can be read as “the order of growth of $T_1(n)$ is n^3 ”, or “the order of growth of $T_1(n)$ is cubic”. It should never be read literally however that “ $T_1(n)$ is equal to n^3 ” : this is plainly wrong. Moreover the alternative writing that uses $\in$ can also be read “ $T_1(n)$ belongs to the set of cubic functions”, which is the same as “the order of growth of $T_1(n)$ is cubic”.

Fact 1 (Comparison vs Asymptotic comparison). When we compare two functions $f(n)$ and $g(n)$ we directly compare them using $<, =, >$ etc. When we asymptotically compare two functions $f(n)$ and $g(n)$ we take the limit of $\lim_{n \rightarrow \infty} f(n)/g(n)$ and then use the asymptotic symbols o, ω, Θ depending on whether the limit is $0, \infty$ or some non-zero (and positive) constant.

Formally, an asymptotic comparison of functions involves limits.

Definition 1 (Formal Approach for asymptotic comparison of functions). In order to compare asymptotically two functions $f(n)$ and $g(n)$ we consider the limit $\lim_{n \rightarrow \infty} f(n)/g(n)$ for $n \rightarrow \infty$.

Definition 2 (Informal Approach for asymptotic comparison of functions). In order to informally compare asymptotically two functions $f(n)$ and $g(n)$ we eliminate from both functions all low-order terms, and turn multiplicative constants into one and then compare the leftovers $F(n)$ and $G(n)$ of $f(n)$ and $g(n)$. Note that then a careful conclusion needs to be drawn!

Asymptotic notation

Writing the symbols

Definition 3 (Asymptotic notation Symbols). For positive or non-negative functions $f(n)$ and $g(n)$ that depend on an integer variable n , we shall introduce symbols to denote asymptotically smaller, asymptotically at most (no greater), asymptotically larger, asymptotically at least (no smaller), and asymptotically equal respectively.

- o (little oh),
- O (big oh),
- ω (little omega),
- Ω (big omega), and
- Θ (big theta, or plain theta)

Definition 4 (Writing). We would write

$$f(n) = ?(g(n)),$$

where $?$ is one of $o, O, \omega, \Omega, \Theta$ to indicate the asymptotic relationship between $f(n)$ and $g(n)$.

Definition 5 (Alternative Writing). We would write

$$f(n) \in ?(g(n)),$$

where $?$ is one of $o, O, \omega, \Omega, \Theta$ to indicate the asymptotic relationship between $f(n)$ and $g(n)$.

Writing the symbols more info

The $\Theta(g(n))$ denotes a set of functions.

It is accurately reflected by the use of $f(n) \in \Theta(g(n))$. We "abuse it" by writing $f(n) = \Theta(g(n))$ instead. Then the relationship $f(n) = \Theta(g(n))$, becomes also a set-membership relationship. It should be borne in mind that the left hand side of $=$ is a function and the right hand side is a set of functions.

Never use a $\leq$ or $\geq$ nor $\neq$. Thus $T(n) \leq \Theta(n)$ is NONSENSE.

Never write a $\Theta(g(n)) = f(n)$ since it is meaningless and wrong!

The notation $f(n) \geq \Theta(g(n))$ is also meaningless and wrong (see previous remark)!

A $T(n) \notin \Theta(n)$ would mean that $T(n)$ does not have linear growth, whereas a $T(n) \in \Theta(n)$ means that $T(n)$ has linear growth, belongs to the set of linear functions and thus is a linear function itself.

If we write $\Theta(f(n)) = \Theta(g(n))$, this is to mean that **no matter how we choose a function on the left hand side, there is a way to choose a function on the right hand side to make equality to hold.**

In other words, the two sets are equal.

DRAFT

Definition 6 (Meaning of Θ). What does $f(n) = \Theta(g(n))$ mean? It means that the two functions have the same growth or $f(n)$ is asymptotically equal to $g(n)$ or $g(n)$ is asymptotically equal to $f(n)$, or $f(n)$ belongs to the set of functions that have the same growth as $g(n)$.

Definition 7 (Meaning of O). What does $f(n) = O(g(n))$ mean? It means that the growth of $f(n)$ is no more than the growth of $g(n)$ (or abusing notation that $f(n)$ is asymptotically at most no greater than $g(n)$).

IT DOES NOT NECESSARILY MEAN that $f(n) \leq g(n)$. For example for $f(n) = 10n$ and $g(n) = n$ the two functions have the same growth i.e. $10n = O(n)$, even if $10n \geq n$. It is also $n = O(10n)$. And in fact $f(n) = \Theta(g(n))$!

Definition 8 (Meaning of Ω). What does $f(n) = \Omega(g(n))$ mean? It means that the growth of $f(n)$ is greater than (faster) or equal to the growth of $g(n)$, or that $f(n)$ is asymptotically at least (no smaller) than $g(n)$.

IT DOES NOT NECESSARILY MEAN that $f(n) \geq g(n)$. For example $f(n) = n$ and $g(n) = 10n$ have the same growth i.e. $n = \Omega(10n)$, even if $n \leq 10n$.

Asymptotic notation

Meaning of Symbols o, ω

Definition 9 (Meaning of o). What does $f(n) = o(g(n))$ mean? It means that the growth of $f(n)$ is slower than the growth of $g(n)$ or $f(n)$ is asymptotically smaller than $g(n)$. (It also means that the growth of $g(n)$ is faster than the growth of $f(n)$ or $g(n)$ is asymptotically larger than $f(n)$.)

Definition 10 (Meaning of ω). What does $f(n) = \omega(g(n))$ mean? It means the growth of $f(n)$ is faster than the growth of $g(n)$ or $f(n)$ is asymptotically larger than $g(n)$. (Likewise as above for the other direction.)

A formal definition for O and Ω and Θ is given on the next page. This reminds you of the formal definition of a limit. In fact the formal definitions of o and ω on the following page use limits for large enough n i.e. as $n \rightarrow \infty$.

Asymptotic comparison

In (at most) 10 lines

Two functions $f(n)$ and $g(n)$ are formally asymptotically compared as follows.

Definition 11 (Quick o, ω, Θ). In order to asymptotically compare two non-negative functions $f(n)$ and $g(n)$ with the three symbols o, ω, Θ , we first find the limit $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ and then determine which case is applicable as follows. (c is a positive constant).

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \Rightarrow f(n) = o(g(n)) \\ \infty & \Rightarrow f(n) = \omega(g(n)) \\ c \neq 0 & \Rightarrow f(n) = \Theta(g(n)) \end{cases}$$

Corollary 1 ($o \Rightarrow O$ implication). If $f(n) = o(g(n))$, then $f(n) = O(g(n))$.

Corollary 2 ($\omega \Rightarrow \Omega$ implication). If $f(n) = \omega(g(n))$, then $f(n) = \Omega(g(n))$.

Corollary 3 ($\Theta \Leftrightarrow \Omega \wedge O$). $f(n) = \Theta(g(n))$, if and only if $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$.

Definition 12 (Running time of instance I for algorithm A). The running time of algorithm A on an input instance I of problem size n is denoted by $T_A(n, I)$.

Definition 13 ($W_A(n)$). The worst-case running time of A is denoted by $W_A(n)$ and defined as follows

$$W_A(n) = \max_{I, |I|=n} \{T_A(n, I)\}.$$

Definition 14 ($B_A(n)$). The best-case running time of A is denoted by $B_A(n)$ and defined as follows

$$B_A(n) = \min_{I, |I|=n} \{T_A(n, I)\}.$$

Definition 15 ($T_A(n)$ or $T(n)$). The running time of algorithm A is denoted $T_A(n)$ or $T(n)$ and defined as follows.

$$T_A(n) = T(n) = O(W_A(n)).$$

If $W_A(n) = \Theta(B_A(n))$, then we can also write,

$$T_A(n) = T(n) = \Theta(W_A(n)).$$

Asymptotic notation

Formal definition for O , Ω

Definition 16 (O definition). Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. For a function $g(n)$ we denote by $O(g(n))$ the set of all functions $f(n)$ that have the following property. There exist positive constants c_2, n_0 such that

$$\exists \text{pos.consts } c_2, n_0 : \quad 0 \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$

for all $n \geq n_0$. For a function $f(n)$ that satisfies this property we write

$$f(n) = O(g(n)) \quad \text{or equivalently} \quad f(n) \in O(g(n)).$$

We read this by saying that function $f(n)$ has at most the asymptotic growth of $g(n)$.

Definition 17 (Ω definition). Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. For a function $g(n)$ we denote by $\Omega(g(n))$ the set of all functions $f(n)$ that have the following property. There exist positive constants c_1, n_0 such that

$$\exists \text{pos.consts } c_1, n_0 : \quad 0 \leq c_1 g(n) \leq f(n) \quad \forall n \geq n_0$$

for all $n \geq n_0$. For a function $f(n)$ that satisfies this property we write

$$f(n) = \Omega(g(n)) \quad \text{or equivalently} \quad f(n) \in \Omega(g(n)).$$

We read this by saying that function $f(n)$ has at least the asymptotic growth of $g(n)$.

Asymptotic notation

Formal definition for Θ

Definition 18 (Θ definition). Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. For a function $g(n)$ we denote by $\Theta(g(n))$ the set of all functions $f(n)$ that have the following property. There exist positive constants c_1, c_2, n_0 such that

$$\exists \text{pos.consts } c_1, c_2, n_0 : \quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$

for all $n \geq n_0$. For a function $f(n)$ that satisfies this property we write

$$f(n) = \Theta(g(n)) \quad \text{or equivalently} \quad f(n) \in \Theta(g(n)).$$

We read this by saying that function $f(n)$ has the asymptotic growth of $g(n)$ or that $f(n)$ is asymptotically equal to $g(n)$.

We can then provide an alternative definition for Θ .

Definition 19 (Θ alternative definition). Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. We write

$$f(n) = \Theta(g(n)) \quad \text{or equivalently} \quad f(n) \in \Theta(g(n)).$$

if and only if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c,$$

where c is a non-zero positive constant. We then say that $f(n)$ is asymptotically equal to $g(n)$ or the asymptotic growth of $f(n)$ is equal to the asymptotic growth of $g(n)$.

Asymptotic notation

Formal definition for o , ω

Definition 20 (o definition). Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. We write

$$f(n) = o(g(n)) \quad \text{or equivalently} \quad f(n) \in o(g(n)).$$

if and only if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$$

We then say that $f(n)$ is asymptotically smaller than $g(n)$ or the asymptotic growth of $f(n)$ is smaller (slower) than the asymptotic growth of $g(n)$.

Definition 21 (ω definition). Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. We write

$$f(n) = \omega(g(n)) \quad \text{or equivalently} \quad f(n) \in \omega(g(n)).$$

if and only if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$$

We then say that $f(n)$ is asymptotically larger than $g(n)$ or the asymptotic growth of $f(n)$ is greater (faster) than the asymptotic growth of $g(n)$.

Growth of Functions

Examples

Example 1. Replace ? with the best of $o, \omega, O, \Omega, \Theta$.

1. $10 = ?(1)$.
2. $n = ?(\lg n)$.
3. $2^{\lg n} = ?(n)$.
4. $2^{\lg n} \lg n = ?(\lg(n!))$.
5. $2^n = ?4^{n/2}$.
6. $n^2 = ?(2^n)$.
7. $n! = ?(2^n)$.

Answers. $\Theta, \omega, \Theta, \Theta, \Theta, o, \omega$. ■

Growth of Functions

Examples

Example 2. Use the definition of O, Ω, Θ to shown that

$$n^2 - 10n + 5 = \Theta(n^2)$$

Answer. We have $f(n) = n^2 - 10n + 5$, $g(n) = n^2$.

1. We show first $f(n) = O(g(n))$, that is $n^2 - 10n + 5 = O(n^2)$.

$O : n^2 - 10n + 5 = O(n^2) : \exists$ **pos. constants** $c_2, n_0 > 0 : n^2 - 10n + 5 \leq c_2 n^2 \quad \forall n \geq n_0$

For all values of $n \geq 1 = n_0$, we have that $-10n \leq 0$ and $5n \leq 5n^2$. Therefore, we upper bound

$$n^2 - 10n + 5 \leq n^2 - 0 + 5n \leq n^2 + 5n \leq n^2 + 5n^2 \leq 6n^2$$

Thus for $\boxed{c_2 = 6 \text{ and } n_0 = 1}$ we have shown $f(n) = n^2 - 10n + 5 \leq 6(= c_2)n^2$ for all $n \geq 1 = n_0$.

2. We show next $f(n) = \Omega(g(n))$, that is $n^2 - 10n + 5 = \Omega(n^2)$.

$O : n^2 - 10n + 5 = \Omega(n^2) : \exists$ **pos. constants** $c_1, n_0 > 0 : n^2 - 10n + 5 \geq c_1 n^2 \quad \forall n \geq n_0$

For all values of $n > 1$, we have that $5 \geq 0$ and for all values of $n \geq 20$ we have that $n^2 - 10n \geq (1/2)n^2$ since

$$n^2 - 10n \geq 1/2 n^2 \iff n^2/2 \geq 10n \iff n/2 \geq 10 \iff n \geq 20.$$

Therefore, we lower bound for $n \geq 20 = n_0$ and $c_1 = 1/2$,

$$n^2 - 10n + 5 \geq n^2 - 10n + 0 \geq n^2 - 10n \geq n^2/2 \geq (1/2)n^2$$

Thus for $\boxed{c_1 = 1/2 \text{ and } n_0 = 20}$ we have shown $f(n) = n^2 - 10n + 5 \geq 1/2(= c_1)n^2$ for all $n \geq 20 = n_0$.

3. We combine the O and Ω results. For $n_0 = \max(1, 20) = 20$ and $c_1 = 1/2$ and $c_2 = 6$ we have

$$f(n) = O(n^2) \quad \text{AND} \quad f(n) = \Omega(n^2)$$

implying $n^2 - 10n + 5 = \Theta(n^2)$. ■

Note 1 (Large enough means asymptotically large). The expression “**for large enough n** ” means “**there is a positive constant n_0 such that for all $n > n_0$** ”, that is for “asymptotically large values of n ”.

Remark 1. Normally, the phrase “**the running time is $O(n^2)$** ” is meaningless as the running time is expressed by a number and a unit of time (e.g. 6 seconds, 10 milliseconds). *This expression when used this way in this class means* that the worst-case running time (which is a function of n) is $\Theta(n^2)$, and by extension, no matter what a particular input of size n is chosen for each value of n , the running time on that set of input instances is $\Theta(n^2)$ or smaller, i.e. no worse than the worst-case running time.

Remark 2. A running time of $\Omega(g(n))$ for an algorithm means that no matter what the input of size n is, for each value of n , the running time of the algorithm will be at least $cg(n)$, for some constant c for large enough n . For example the **running time of insertion-sort** is $\Omega(n)$. Yet the **running time of bubble-sort** is $\Omega(n^2)$.

Note 2 (Polynomial functions). A function n^m for any positive constant integer $m > 0$ is a polynomial in n i.e. a polynomial function. The linear combination of polynomial functions is also a polynomial function.

Note 3 (Polylogarithmic functions). A function $\lg^m n$ for any positive constant integer $m > 0$ means $(\lg n)^m$ and is a polylogarithmic function.

Note 4 (Polynomial vs Polylogarithmic). For any positive constant integer $k, m > 0$ and integer $n > 0$, we have that $n^m > \lg^k n$ for large enough n . *This means that every polynomial function is asymptotically larger than any logarithmic function of the same variable n .*

Note 5 (Exponential vs Polynomial). For any positive constant integer m and integer $n > 0$, we have that $2^n > n^m$ for large enough n . *This means that every exponential function (base two) is asymptotically larger than any polynomial function of the same variable n .*

Note 6 (Non-commutative use of symbols). $\Theta(1) = 5$ means nothing. The symbols as defined earlier appear always to the right of the equal = sign. The set theoretic view make it easy to argue that $\Theta(1) \in 5$ does not make sense.

Note 7 (Tomatoes vs Potatoes or $<$ and O etc). $1 \leq O(n)$ is meaningless. One can say $1 = O(n)$ and this was properly defined above, or $1 = o(n)$. Nothing else was defined involving the $<, >, \leq, \geq$ and the five letter symbols!

Example 3. The best-case running time of InsertionSort is attained by a sorted sequence: Thus

$$T_{\text{InsSort}}(n, \text{Sorted}) = \Theta(n).$$

Example 4. The worst-case running time of InsertionSort is attained by a reverse sorted sequence:

$$T_{\text{InsSort}}(n, \text{ReverseSorted}) = \Theta(n^2).$$

Example 5. The running time bound of InsertionSort is

$$O(\max_I T_{\text{InsSort}}(n, I)) = O(n^2).$$

Example 6. $10 = \Theta(1)$. Since $\lim_{n \rightarrow \infty} 10/1 = 10$, we have $10 = \Theta(1)$. By Lemma 3 we have $10 = O(1)$ and $10 = \Omega(1)$ as well.

Growth of Functions

Examples and Exercises

Example 7. $n = ?(\lg n)$. Since $\lim n/\lg n = \lim 1/(1/n) = \infty$ as $n \rightarrow \infty$, we have $n = \omega(\lg n)$. By Lemma 2 we have $n = \Omega(\lg n)$ as well.

Example 8. $2^{\lg n} = ?(n)$. Since $\lim 2^{\lg n}/n = \lim n/n = 1$ as $n \rightarrow \infty$, we have $2^{\lg n} = \Theta(n)$. By Lemma 3 we can also use O and Ω .

Example 9. $2^n = ?(n)$. Since $\lim 2^n/n = \infty$ as $n \rightarrow \infty$, we have $2^n = \omega(n)$. By Lemma 2 we have $2^n = \Omega(n)$ as well.

Example 10. $n \lg n = ?(\lg(n!))$. Note that by Stirling's approximation formula $\lg(n!) \approx n \lg n$. Since $\lim n \lg n / \lg(n!) = c > 0$ as $n \rightarrow \infty$, and c some constant, we have $n \lg n = \Theta(\lg(n!))$. By Lemma 3 we have O and Ω as well.

Example 11. $n = ?(n^3)$. Since $\lim n/n^3 = 0$ as $n \rightarrow \infty$, we have $n = o(n^3)$. By Lemma 1 we have $n = O(n^3)$ as well.

Example 12. $2^n = \omega(n^2)$. Since $\lim 2^n/n^2 = \infty$ as $n \rightarrow \infty$, we have $2^n = \omega(n^2)$. By Lemma 2 we have $2^n = \Omega(n^2)$ as well.

Example 13. $n! = \omega(2^n)$. Since $\lim n!/2^n = \infty$ as $n \rightarrow \infty$, we have $n! = \omega(2^n)$. By Lemma 2 we have $n! = \Omega(2^n)$ as well.

Example 14. Show that $n^2 - 2n = \Theta(n^2)$. We determine POSITIVE CONSTANTS c_1, c_2, n_0 so that for all $n \geq n_0$.

$$c_1 n^2 \leq n^2 - 2n \leq c_2 n^2$$

We separate the O from the Ω part and then combine the two parts.

Part O first. As $n^2 - 2n \leq n^2$ for all $n \geq 1$, we have shown that $n^2 - 2n \leq c_2 n^2$, for $c_2 = 1$ and for all $n \geq n_1 = 1$. This is equivalent to showing that $n^2 - 2n = O(n^2)$ as well.

Part Ω next. As $n^2/2 \leq n^2 - 2n$ for all $n \geq 4$, we have shown that $c_1 n^2 \leq n^2 - 2n$, for $c_1 = 1/2$ and $n \geq n_2 = 4$. This is equivalent to showing that $n^2 - 2n = \Omega(n^2)$ as well.

Combine the two. The n_0 in the definition is the largest of n_1 and n_2 i.e. $n_0 = 4$. Then, for $c_1 = 1/2$, $c_2 = 1$ and for all $n \geq n_0 = 4$, we have

$$c_1 n^2 \leq n^2 - 2n \leq c_2 n^2$$

Growth of Functions

Examples and Exercises

Note 8. $c_1 = 1/2$, $c_2 = 1$ and $n_0 = 1000$ is also a correct answer to this problem. OUR OBJECTIVE IS TO FIND A SET OF SATISFYING CONSTANTS not THE BEST SET OF SATISFYING CONSTANTS.

Example 15. Since a constant is a degree 0 polynomial any constant c is such that $c = \Theta(1)$, $c = O(1)$, and $c = \Omega(1)$.

Example 16. Show that $n^3 = \omega(n)$. Since $\lim n^3/n = \infty$ as $n \rightarrow \infty$, we have $n^3 = \omega(n)$. The result can also be proved as a consequence of $n = o(n^3)$.

Example 17. Show that $n^{2/\lg n} = o(n)$.

$$\frac{n^{2/\lg n}}{n} = \frac{(2^{\lg n})^{2/\lg n}}{n} = \frac{2^2}{n} = \frac{4}{n} \rightarrow 0$$

Thus $n^{2/\lg n} = o(n)$.

Example 18. Show that $n! = \omega(n)$. From Stirling approximation formula, $n! \approx (n/e)^n$. Therefore, $\frac{n!}{n} \rightarrow \infty$. Thus $n! = \omega(n)$.

Example 19. Is $2^n = 2^{\Theta(n)}$? Be careful. The Θ is not on the right of the equal size. You are looking for trouble (or not)! (We sometimes write something similar to this because it is convenient.) Let me rephrase it: Is $2^n = \Theta(2^{\Theta(n)})$? A function $g(n)$ that is $\Theta(n)$ for the second Θ is $g(n) = 2n$. So let me rephrase the question: Is $2^n = \Theta(2^{2n})$? Well $2^n/2^{2n} \rightarrow 0$.

Example 20. Show that $n^2 - 10n + 2 = O(n^2)$. We are going to determine for $f(n) = n^2 - 10n + 2$ and $g(n) = n^2$, that there

$$\exists \text{ positive constants } c_2, n_0 : f(n) \leq c_2 g(n) \quad \forall n \geq n_0.$$

Towards this

$$f(n) = n^2 - 10n + 2 \leq n^2 + 0 + 2 \leq n^2 + 0 + 2n^2 \leq 3n^2$$

Note that $-10n \leq 0$ is true for all $n \geq 0$ and thus for positive n_0 and $n \geq n_0$. Moreover $2 \leq 2n^2$ for all $n \geq 1$ and this requires $n_0 \geq 1$.

Conclusion. There exist $c_2 = 3$ and $n_0 = 1$ such that $f(n) = n^2 - 10n + 2 = O(n^2)$ for all $n \geq n_0 = 1$.

Example 21. Show that $n^2 - 10n + 2 = \Omega(n^2)$. We are going to determine for $f(n) = n^2 - 10n + 2$ and $g(n) = n^2$, that there

$$\exists \text{ positive constants } c_1, n_0 : 0 \leq c_1 g(n) \leq f(n) \quad \forall n \geq n_0.$$

Towards this

$$\frac{1}{2}n^2 \leq n^2 - \frac{1}{2}n^2 \leq n^2 - 10n \leq n^2 - 10n + 0 \leq n^2 - 10n + 2.$$

For $n^2 - 10n \geq n^2 - (1/2)n^2$ to hold we need $n^2 \geq 20n$ i.e. $n \geq 20$. Thus $n_0 \geq 20$; we pick $n_0 = 20$.

Conclusion. There exist $c_1 = 1/2$ and $n_0 = 20$ such that $f(n) = n^2 - 10n + 2 = \Omega(n^2)$ for all $n \geq n_0 = 20$.

Exercise 1. Show that

$$\sum_{i=1}^n i^2 = \Theta(n^3).$$

What are the values of c_1, c_2 and n_0 ? Justify your answer.

Exercise 2. Show that $1000 = O(1)$.

Example 22. Use the definition of Θ to show that $n^5 - 25n = \Theta(n^5)$.

Proof. We show first that $n^5 - 25 = O(n^5)$ and then that $n^5 - 25 = \Omega(n^5)$. **Case a.** Show that $n^5 - 25 = O(n^5)$.

For all $n \geq 1$ we have that

$$n^5 - 25 \leq n^5$$

Therefore there exist constant $n_2 = 1$ and $c_2 = 1$ such that $n^5 - 25 \leq c_2 n^5$ for all $n \geq n_2$. This proves the claim.

Technique 1. What we use in this proof is the fact that $n^2 \pm An \pm B$ is bounded above, for positive A, B , by $n^2 + An^2 + Bn^2 \leq (1 + A + B)n^2$.

Case b. Show that $n^5 - 25 = \Omega(n^5)$. Technique 1 can not be used in this case. The next step is non-trivial. We bound $n^5 - 25$ from below by $n^5/2$. This is so as long as

$$\begin{aligned} n^5 - 25 &\geq n^5/2 \Leftrightarrow \\ n^5/2 &\geq 25 \Leftrightarrow \\ n^5 &\geq 50 \Leftrightarrow \\ n &\geq 3 \end{aligned}$$

We can do this as long as n is not zero; this is true since for all cases we assume that at least $n \geq 1$. Therefore there exist constant $n_1 = 3$ and $c_1 = 1/2$ such that $n^5 - 25 \geq n^5/2$ for all $n \geq n_1 = 3$. This proves the claim. Note that 3 is not the best possible constant. We don't need to find the best possible constant but THE EASIEST POSSIBLE! In order to show that $n^5 - 25 = \Theta(n^5)$ we need to establish c_1, c_2 and n_0 . c_1 and c_2 are $1/2$ and 1 respectively. $n_0 = \max(n_1, n_2) = 3$. For these values the problem is thus shown. ■

Recurrence relations or Recurrences

Introduction

For a recursive algorithm like mergesort, its running time is described by a recurrence relation like the one shown below. We call it sometimes a recurrence dropping the relation from recurrence relation.

$$T(n) = T(n/2) + T(n/2) + \Theta(n) = 2T(n/2) + \Theta(n).$$

Likewise the number of comparisons in mergesort (upper bound of number of comparisons) is also described by the following recurrence

$$C(n) = 2C(n/2) + (n - 1) \quad C(1) = 0.$$

In order to solve the former recurrence we need a boundary condition for the base case of the recurrence. The base case for the running time of merge-sort is the running time for an input of size 1 or in general, of some small constant size. In this case, we may assume that $T(1) = 1$, or $T(1) = \Theta(1)$ i.e. the sorting of a one-key sequence takes one step or constant time. The solution for this recurrence was claimed to be $T(n) = \Theta(n \lg n)$ (proof by induction or by using a recursion tree).

Floors and Ceilings. In the recurrence above, we ignored floors and ceilings. When we solve recurrences, we first remove floors/ceilings, show the claimed bounds and then check whether the existence of floors/ceilings would or could have made a difference. For the remainder we assume that values like $n/2$ are always integer, and so the effect of the removal of floors/ceilings will not be discussed.

Bounday values and Boundary value conditions. For boundary conditions we assume in general that $T(k) = \Theta(1)$, for some small constant k , as constant problem sizes can be solved in constant time. We use $T(k) = \Theta(1)$ instead of a more specific/explicit $T(k) = l$ (l some constant), as such a choice affects only constants in the expression for $T(n)$; i.e. growth rate is preserved. In general, **we shall ignore statements about boundary conditions**, because this simplifies the recurrence solution. If a boundary condition is explicitly given, the solution must be consistent with it. By the way the value k is a boundary value and $\Theta(1)$ or $T(k) = \Theta(1)$ is the corresponding boundary condition.

Recurrence relations

Three methods

We introduce **three methods** to solve recurrences.

- **Master** method: Provides a solution formula for recurrences of certain form such as $T(n) = aT(n/b) + f(n)$, for constant $a \geq 1$ and $b > 1$ and asymptotically positive $f(n)$.
- **RecursionTree/Iteration** method: Unfold recurrence by turning it into a sum.
- **Substitution or “guess and check”** method: Guess solution and then verify/check it (eg. proof by induction).

The master method only provides a tight Θ asymptotic bound.

The recursion tree (iteration) method is more versatile and can provide exact or asymptotic answers but is math manipulation intensive.

The substitution method is more versatile and can provide exact or asymptotic answers it is math manipulation intensive, uses (strong) induction and one needs to have a good guess. Usually the good guess is after spending some time with the recursion tree or iteration method.

We start the discussion of the master method on the next page.

The master method

Introduction

The master method is one of three methods to solve recurrence relations. The solution however it provides is an asymptotic one and it works only for recurrences of a certain form.

Method 1 (Master method). Let $T(n) = aT(n/b) + f(n)$ be such that $a \geq 1$, $b > 1$ are (real) constant and $f(n)$ is an asymptotically positive function. Then $T(n)$ is bounded as follows.

m1 If $f(n) = O(n^{\lg_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\lg_b a})$.

m2 If $f(n) = \Theta(n^{\lg_b a})$, then $T(n) = \Theta(n^{\lg_b a} \lg n)$.

m3 If $f(n) = \Omega(n^{\lg_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $0 < c < 1$ and for large n , then $T(n) = \Theta(f(n))$.

There is an alternative formulation for Case 2 (aka m2) of the master method. Sometimes the condition $a \geq 1$ is stated as $a > 0$.

Method 2 (Master method alternative formulation of case m2).

- **m2'** If $f(n) = \Theta(n^{\lg_b a} (\lg n)^k)$, for some non-negative constant k , then $T(n) = \Theta(n^{\lg_b a} (\lg n)^{k+1})$.

Method 3 (Alternative form of Master method). Let $T(n) = aT(n/b) + f(n)$ be such that $a > 0$, $b > 1$ are (real) constant and $f(n)$ is an asymptotically positive function. Then $T(n)$ is bounded as follows.

M1 If $f(n) = O(n^{\lg_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\lg_b a})$.

M2' If $f(n) = \Theta(n^{\lg_b a} (\lg n)^k)$, for some non-negative constant k , then $T(n) = \Theta(n^{\lg_b a} (\lg n)^{k+1})$.

M3 If $f(n) = \Omega(n^{\lg_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $0 < c < 1$ and for large n , then $T(n) = \Theta(f(n))$.

This is going to be the form utilized in the remainder.

DRAFT

Instructional Example 1. Solve the recurrence relation using the master method.

$$T(n) = 2T(n/2) + n.$$

Proof. $a = 2 > 0$,
 $b = 2 > 1$ and
 $\lg_b a = 1$ i.e. $n^{\lg_b a} = n$,
 $f(n) = n$

Obviously $f(n) = \Theta(n^{\lg_b a})$ i.e. $n = \Theta(n)$ thus **CASE 2** is applicable. Then $T(n) = \Theta(n \lg n)$. ■

Instructional Example 2. Solve the recurrence relation using the master method.

$$T(n) = 8T(n/2) + n^2.$$

Proof. $a = 8 > 0$,
 $b = 2 > 1$ and
 $\lg_b a = \lg_2 8 = 3$ i.e. $n^{\lg_b a} = n^3$,
 $f(n) = n^2$

Obviously $f(n) = O(n^{\lg_b a - 1})$ i.e. $n^2 = \Theta(n^{3-1})$ thus **CASE 1** is applicable with $\epsilon = 1 > 0$. Then $T(n) = \Theta(n^3)$. ■

Instructional Example 3. Solve the recurrence relation using the master method.

$$T(n) = 2T(n/2) + n^2.$$

Proof. $a = 2 > 0$,
 $b = 2 > 1$ and
 $\lg_b a = 1$ i.e. $n^{\lg_b a} = n$,
 $f(n) = n^2$

Obviously $f(n) = \Omega(n^{\lg_b a + 1})$ i.e. $n^2 = \Omega(n^{1+1})$ with $\epsilon = 1 > 0$, i.e. **CASE 3** might be applicable. For that we need to verify the second condition $af(n/b) \leq cf(n)$.

$$\begin{aligned} af(n/b) &\leq cf(n) \\ 2f(n/2) &\leq cf(n) \\ 2(n/2)^2 &\leq cn^2 \\ n^2/2 &\leq cn^2 \end{aligned}$$

If we choose $c = 1/2 < 1$ the secondary condition is also true. Then $T(n) = \Theta(n^2)$. ■

Example 23. Note that $a^{\lg n / \lg b} = 2^{\lg a \lg n / \lg b} = n^{\lg a / \lg b}$. Consider $T(n) = 9T(n/3) + f(n)$. Then $a = 9$, $b = 3$ and $\lg_b a = 2$ i.e. $n^{\lg_b a} = \Theta(n^2)$. If $f(n) = n$, then $T(n) = \Theta(n^2)$. If however $f(n) = n^2$, then $T(n) = \Theta(n^2 \lg n)$.

Example 24. $T(n) = T(n/2) + f(n)$, where $a = 1$ and $b = 2$, and $n^{\lg_b a} = n^0 = 1$. If $f(n) = 1$, then $T(n) = \Theta(\lg n)$.
If $f(n) = \sqrt{n}$, then because $af(n/b) = a\sqrt{n/b} = 1\sqrt{n}/\sqrt{2} \leq \sqrt{1/2}\sqrt{n} = cf(n)$ for $c = \sqrt{1/2}$, we get that $T(n) = \Theta(\sqrt{n})$.

DRAFT

The substitution method

Introduction

The second method for solving a recurrence is known as the substitution method. In order to use it either we have a good guess of a solution, or we exhaustively search for multiple solution candidates. Here we know what the solution would look like either through trial of error or some prior knowledge. We would like to verify indeed that this is a solution. We verify a candidate solution by using strong induction. (Reminder a $T(n/2)$ implies or reminds a $P(n/2)$ as part of the induction hypothesis.) The solution to be verified can be an exact one, an asymptotic one (Big-Oh, or little-oh, Big-Omega, or little-omega) or a tight one (Theta).

DRAFT

The substitution method
Substitution method Example

Instructional Example 4. Solve the following recurrence $T(n) = T(n/5) + T(7n/10) + n$ where $T(5) = 10$.

Solution Preliminaries. This is a the recurrence for worst-case linear time selection (k order statistic problem) of a later Subject. Thus we know that solution is supposed to be linear i.e. $T(n) = \Theta(n)$. We prove the result in two steps first $T(n) = O(n)$ and then $T(n) = \Omega(n)$, utilizing the definition of O and Ω

DRAFT

The substitution method

Example continued: Ω part

Proof.

$T(n) = \Omega(n)$. We will find two positive constant c_1, n_0 such that $c_1 n \leq T(n)$ for all $n \geq n_0$. We define $P(n) : T(n) \geq c_1 n$.

Base case ($n = 5$).

Show $P(5)$ is true. This is equivalent to $T(5) \geq c_1 \cdot 5$. For the latter to be true utilizing the base case we need to have the following.

$$\begin{aligned} P(5) & \text{ is true?} \\ T(5) & \geq? c_1 \cdot 5 \\ 10 & \geq? c_1 \cdot 5 \\ 2 & \geq? c_1 \end{aligned}$$

Thus if $c_1 \leq 2$, then $P(5)$ is true and base case is proved (also implying $n_0 = 5$).

Inductive Step.

$$P(5) \wedge \dots \wedge P(n/5) \wedge \dots \wedge P(7n/10) \wedge \dots \wedge P(n-1) \Rightarrow P(n)$$

There is not much to be done. Starting with the recurrence $T(n) = T(n/5) + T(7n/10) + n \geq 0 + 0 + n$ we conclude that $T(n) \geq 1 \cdot n$.

Conclusion. From the base case we have $T(n) \geq c_1 n$ for $c_1 \leq 2$. From the inductive step we have $T(n) \geq c_1 n$ with $c_1 \leq 1$. Reconciling the two c_1 's we pick $c_1 = 1$ with n_0 the base case value i.e. $n_0 = 5$. $c_1 = 1, n_0 = 5$ for the Ω part. ■

We continue with the second part and proving that

The substitution method

Example continued: O part

Proof.

$T(n) = O(n)$. We will find two positive constant c_2, n_0 such that $T(n) \leq c_2 n$ for all $n \geq n_0$. We define $P(n) : T(n) \leq c_2 n$.

Base case ($n = 5$).

Show $P(5)$ is true. This is equivalent to $T(5) \leq c_2 \cdot 5$. For the latter to be true utilizing the base case we need to have the following.

$$\begin{aligned} P(5) & \text{ is true?} \\ T(5) & \leq? c_2 \cdot 5 \\ 10 & \leq? c_2 \cdot 5 \\ 2 & \leq? c_2 \end{aligned}$$

Thus if $c_2 \geq 2$ for $n_0 = 5$ establishing the base case, then $P(5)$ is true and base case is proved.

Inductive Step.

$$P(5) \wedge \dots \wedge P(n/5) \wedge \dots \wedge P(7n/10) \wedge \dots \wedge P(n-1) \Rightarrow P(n)$$

The induction hypothesis (assumption) considers among others $P(n/5)$ and $P(7n/10)$ i.e.

$$\begin{aligned} & \boxed{X: P(n/5) \text{ implies } T(n/5) \leq c_2(n/5)} \text{ and} \\ & \boxed{Y: P(7n/10) \text{ implies } T(7n/10) \leq c_2(7n/10)}. \end{aligned}$$

We then use the recurrence and expressions X and Y as follows.

The substitution method
Example continued: O part

$$\begin{aligned}T(n) &= T(n/5) + T(7n/10) + n \\&\leq_X c_2 n/5 + T(7n/10) + n \\&\leq_Y c_2 n/5 + c_2(7n/10) + n \\&\leq c_2(9n/10) + n\end{aligned}$$

We have shown so far that $T(n) \leq c_2(9n/10) + n$. If we manage to prove that $c_2(9n/10) + n \leq c_2 n$ then by transitivity we have shown $P(n) : T(n) \leq c_2 n$ and we are done.

$$\begin{aligned}c_2(9n/10) + n &\leq? c_2 n \\n &\leq? c_2 n - c_2(9n/10) \\n &\leq? c_2 n/10 \\1 &\leq? c_2/10 \\10 &\leq? c_2\end{aligned}$$

In the previous derivation n was cancelled out provided $n > 0$. Since by the definition of big-Oh n is integer we use $n \geq 1$. Now if $T(n) \leq c_2(9n/10) + n$ and $c_2(9n/10) \leq c_2 n$ provided $n \geq 1$ and $c_2 \geq 10$ we have also shown by transitivity $T(n) \leq c_2 n$ and thus we conclude the inductive step.

Conclusion. By the base case $c_2 \geq 2$ and $n_0 = 5$. By the inductive step $n_0 = 1$ and $c_2 \geq 10$. To reconcile we use $n_0 = 5$ and $c_2 = 10$. ■

O and Ω reconciliation. For the Ω part we used $n_0 = 5$ and $c_1 = 1$. For the O part we have $n_0 = 5$ and $c_2 = 10$. Thus $c_1 = 1, c_2 = 10, n_0 = 5$ and

$$1 \cdot n \leq T(n) \leq 10 \cdot n \quad \forall n \geq 5.$$

DRAFT

Example 25. Solve the recurrence $T(n) = 2T(n/2) + n$ using the substitution (a.k.a. guess-and-check) method. (Implicit assumption is that $T(n)$ is nonnegative and defined for all positive n , or for arbitrarily large n).

Since no boundary condition is given we can thus choose k and l constants greater than zero so that $T(k) = l$. We choose k, l in such a way to make the inductive proof as simple as possible. Let us choose $T(1) = 0$.
Proof.

A. Guess Step: $T(n) = O(n \lg n)$. We guess $T(n) = O(n \lg n)$, i.e. $\exists$ pos. constant $c_2, n_0 : T(n) \leq c_2 n \lg n, \forall n \geq n_0$.

A.0 Check Step (Strong Induction). We shall prove our claim by using induction. Our $P(n)$, the proposition to be proven true would be $T(n) \leq c_2 n \lg n$ for arbitrarily large values of $n \geq n_0$, for some positive constant n_0 .

A.1. Base Case of Induction. We show $P(1)$ is true is $T(1) \leq c_2 \cdot 1 \cdot \lg 1$. Since $T(1) = 0$ it is trivially true that $0 = T(1) \leq c_2 \cdot 1 \cdot \lg 1 = 0$ for all choices of a positive and constant c_2 . Base case completed.

A.2. Inductive Step. $P(1) \wedge \dots \wedge P(n/2) \wedge \dots \wedge P(n-1) \Rightarrow P(n)$. We show that if $P(i)$ is true for all $i < n$ then we will show that $P(n)$ is also true. $P(n)$ is $T(n) \leq c_2 \cdot n \cdot \lg n$. If $P(i)$ is true for all $i < n$, since $n/2 < n$ then $P(n/2)$ is also true. Then by the inductive assumption for $i = n/2$ we have $T(n/2) \leq c_2(n/2) \lg(n/2)$. We then use the recurrence and utilize the inductive hypothesis for $T(n/2)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\stackrel{P(n/2): T(n/2) \leq c_2(n/2) \lg(n/2)}{=} 2 \underbrace{T(n/2)}_{\leq c_2(n/2) \lg(n/2)} + n \\ &\leq 2c_2(n/2) \lg(n/2) + n = c_2 n \lg n - c_2 n + n \end{aligned}$$

To complete the inductive step we must show that the rightmost expression $c_2 n \lg n - c_2 n + n$ is at most $c_2 n \lg n$. For that to be true i.e. $c_2 n \lg n - c_2 n + n \leq c_2 n \lg n$ we need $-c_2 n + n < 0$ i.e. $c_2 \geq 1$. Thus if we choose a $c_2 = 1$ which is ≥ 1 and for the n_0 of the base case i.e. $n_0 = 1$ we have that

$$T(n) \leq n \lg n \quad \forall n \geq 1.$$

■

The substitution method

Another Example reoriented

Can we do better in the guessing game? Let us try a lower bound.

B. Guess Step: $T(n) = \Omega(n \lg n)$. We guess $T(n) = \Omega(n \lg n)$, i.e. $\exists$ pos. constant $c_1, n_0 : T(n) \geq c_1 n \lg n, \forall n \geq n_0$.

Proof. **B.0 Check Step (Strong Induction).** We shall prove our claim by using induction. The $P(n)$ is $T(n) \geq c_1 n \lg n$ for arbitrarily large values of $n \geq n_0$, for some positive constant n_0 .

B.1. Base Case of Induction. We show $P(1)$ is true: $T(1) \geq c_1 \cdot 1 \cdot \lg 1$. Since $T(1) = 0$, it is true that $0 = T(1) \geq c_1 \cdot 1 \cdot \lg 1 = 0$ for all choices of a positive and constant c_1 . Base case completed.

B.2. Inductive Step. $P(1) \wedge \dots \wedge P(n/2) \wedge \dots \wedge P(n-1) \Rightarrow P(n)$. We show that if $P(i)$ is true for all $i < n$ then we will show that $P(n)$ is also true. $P(n)$ is $T(n) \geq c_1 \cdot n \cdot \lg n$. If $P(i)$ is true for all $i < n$, since $n/2 < n$ then $P(n/2)$ is also true. Then by the inductive assumption for $i = n/2$ we have $T(n/2) \geq c_1(n/2) \lg(n/2)$. We then use the recurrence and utilize the inductive hypothesis for $T(n/2)$.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\stackrel{P(n/2): T(n/2) \geq c_1(n/2) \lg(n/2)}{=} 2 \underbrace{T(n/2)}_{\geq c_1(n/2) \lg(n/2)} + n \\ &\geq 2c_1(n/2) \lg(n/2) + n = c_1 n \lg n - c_1 n + n \end{aligned}$$

To complete the inductive step we must show that the rightmost expression $c_1 n \lg n - c_1 n + n$ is at least $c_1 n \lg n$. For that to be true i.e. $c_1 n \lg n - c_1 n + n \geq c_1 n \lg n$ we need $-c_1 n + n > 0$ i.e. $c_1 \leq 1$. Thus if we choose a $c_1 = 1$ which is ≤ 1 and for the n_0 of the base case i.e. $n_0 = 1$ we have that

$$T(n) \geq n \lg n \quad \forall n \geq 1.$$

Well we have just proven that $T(n) \geq n \lg n$. We also proved (previous page) that $T(n) \leq n \lg n$. Both of them prove that $T(n) = n \lg n$! (The last mark is an exclamation mark, not a factorial!)

Thus while our objective was to prove a tight asymptotic bound the derivation of the three constants n_0, c_1, c_2 allowed us to determine an exact bound. ■

The substitution method

Another example optimized

C. Guess Step : $T(n) = An \lg n + Bn + C$. Let us guess an answer that contains a log-linear, a linear and a constant term. Let's try to compute the constant values A , B and C . Only condition is that A should be positive.

Proof. C.1. Base Case utilization. We have $T(1) = 0$ substituting 1 for n in $T(n) = An \lg n + Bn + C$ we get $0 = T(1) = A \cdot 1 \cdot \lg 1 + B \cdot 1 + C$ i.e. $B + C = 0$.

C.2. Recurrence utilization. Since $T(n) = 2T(n/2) + n$ we have that

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ An \lg n + Bn + C &= 2(A(n/2) \lg(n/2) + B(n/2) + C) + n \\ An \lg n + Bn + C &= An \lg n + (B - A + 1)n + 2C \end{aligned}$$

In the last equality the first term of both sides cancels out. We then equate the coefficients of n and the constant terms and solve for A, B, C also utilizing the base case $B + C = 0$.

The constant terms C and $2C$ should be equal to each other for $C = 0$. Since $B + C = 0$ we also have $B = 0$.

The linear term coefficients B and $(B - A + 1)$ should be equal to each other $B = B - A + 1$. This results to an $A = 1$. Thus the guessed function $T(n) = An \lg n + Bn + C$ gets resolved to having $A = 1$ and $B = C = 0$ leading to the same result with the iteration or previous substitution method. No need to explicitly do induction. ■

The substitution method

Yet another example

Let us try to guess the solution of a variant where the linear term is replaced now with a constant term.

Example 26. What if we try to solve $T(n) = T(n/3) + T(2n/3) + 7$?

Proof. If we try to prove that $P(n) : T(n) \leq cn$ utilizing $P(n/3)$ and $P(2n/3)$ we end up with proving

$$T(n) \leq cn/3 + c2n/3 + 7 \leq cn + 7$$

not $T(n) \leq cn$. The $+7$ is causing us problems. How do we fix this “tiny” problem? By subtracting something smaller than the high order term from the solution i.e. try for a solution a $T(n) \leq cn - a$. ($cn - a$ MUST be positive as well i.e. $n > a/c$). We then get

$$T(n) \leq \overbrace{cn/3 - a}^{P(n/3)} + \overbrace{2cn/3 - a}^{P(2n/3)} + 7 \leq cn - a + (7 - a).$$

In order to prove that $T(n) \leq cn - a$ we must show that $T(n) \leq cn - a + (7 - a) \leq cn - a$ i.e. $7 - a \leq 0$. For the assumption to hold it suffices that $a \geq 7$.

Remark. What if we try to solve $T(n) = T(n/2) + T(n/3) + 10n$? Guess $T(n) = \Theta(n)$ (note that $1/2$ of $n/2$ plus $1/3$ of $n/3$ is $5/6 < 1$).

Remark. What if we try to solve $T(n) = T(2n/3) + T(n/3) + 10n$? Guess $T(n) = O(n \lg n)$ as $2/3 + 1/3 = 1$. $T(n) = \Theta(n \lg n)$ is the tightest possible bound.

Remark. What if we try to solve $T(n) = T(4n/3) + T(n/3) + 10n$? Guess $T(n) = O(n^\alpha)$ or $T(n) = \Theta(n^\alpha)$ for some constant $\alpha > 1$.

Remark. Useful Tricks: Change of variables might help. Remember to derive the correct boundary conditions when changing variables. $T(n) = T(\sqrt{n}) + \lg n$. Change variables $H(m) = T(2^m)$. ■

The Iteration or Recursion Tree Method

Introduction

The iteration method is also known as the repeated substitution method. If the method is done graphically it is sometimes called the **recursion tree method**.

We have seen solutions follow the recursion tree method already: one was in the context of the recursive Fibonacci sequence problem, and the second one in the analysis of the running time but primarily the upper bound on the comparison count of MergeSort.

We will be using the traditional name **iteration method**.

Iteration method.

Expand the recurrence all the way down to the base case and sum up or combine (sometimes multiplication might also help!) the residuals.

In order to avoid problems with floors and ceilings we are going to make an assumption that n is a power of 2 and thus $n/2$, $n/4$, $n/8$ etc are all integer, or a similar, along to the same line assumption, as needed. Otherwise one may have to make use of identities such as that for all integers n, a, b , we have $\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/(ab) \rfloor$.

DRAFT

Instructional Example 5. Solve exactly $T(n) = aT(n/b)$ where $T(1) = 1$. a, b as in the master method.

Proof. First we generate (A1) through (Ai) as needed.

$$T(n) = aT(n/b)$$

$$T(n/b^0) = aT(n/b^1) \tag{A1}$$

$$T(n/b^1) = aT(n/b^2) \tag{A2}$$

$$T(n/b^2) = aT(n/b^3) \tag{A3}$$

$$T(n/b^3) = aT(n/b^4) \tag{A4}$$

...

$$T(n/b^{i-1}) = aT(n/b^i) \tag{Ai}$$

As there are no additive terms above, we can multiply all the equations (A1) through (Ai). The result is $T(n) = a^i T(n/b^i)$. We then equate $T(n/b^i)$ with the base case $T(1) = 1$ so that $n/b^i = 1$ which also implies $T(n/b^i) = T(1) = 1$. Solving for i the $n/b^i = 1$, we determine the number of iterations needed to reach the base case $n = 1$. This is $n/b^i = 1 \implies i = \lg n / \lg b$. Then we substitute $\lg n / \lg b$ for i in $T(n) = a^i T(n/b^i)$.

$$\begin{aligned} T(n) &= a^i T(n/b^i) = a^{\lg n / \lg b} T(n/b^{\lg n / \lg b}) \\ &= 2^{\lg a \lg n / \lg b} T(n/2^{\lg b \lg n / \lg b}) \\ &= n^{\lg a / \lg b} T(n/2^{\lg n}) \\ &= n^{\lg_b a} T(n/n) = n^{\lg_b a} T(1) \\ &= n^{\lg_b a}. \end{aligned}$$

Thus $T(n) = n^{\lg_b a}$. ■

The iteration method

A first example

Example 27. Solve exactly $T(n) = 2T(n/2) + n$, $T(1) = 0$. You may assume that n is a power of two.

Proof. In order to compute $T(n)$ we need to establish $T(n/2)$. Likewise in order to compute $T(n/2)$ we need to establish $T(n/4)$. We iterate i iterations until $T(n/2^i)$ with the intent of making $n/2^i$ equal to the base case value 1. If $n/2^i = 1$ then we know that $T(n/2^i) = T(1) = 0$. For $n/2^i = 1$ we have $n = 2^i$ and solving for i we get $i = \lg n$. Thus unfolding the recurrence involves $i = \lg n$ iterations!

$$n \rightarrow n/2^1 \rightarrow n/2^2 \rightarrow n/2^3 \rightarrow \dots n/2^i \dots \quad \text{until } n/2^i \text{ is base case } 1.$$

$$\begin{aligned} T(n) &= 2T(n/2) + n = 2 \overbrace{(2T(n/2^2) + n/2)}^{T(n/2)} + n \\ &= 2^2 T(n/2^2) + 2n = 2^2 \overbrace{(2T(n/2^3) + n/2^2)}^{T(n/2^2)} + 2n \\ &= 2^3 T(n/2^3) + 3n \\ &\dots \\ &= 2^i T(n/2^i) + i \cdot n \quad \text{Now, Substitute } i = \lg n \text{ as } n/2^i \text{ becomes } n/2^{\lg n} = 1 \\ &= 2^{\lg n} \cdot T(n/2^{\lg n}) + \lg n \cdot n = n \cdot T(1) + \lg n \cdot n = n \lg n \end{aligned}$$

- Keep track of number of iterations/depth of recursion tree.
- Keep track of sum of terms per iteration/level of recursion tree.
- Sometimes, the two previous steps and experience allow us to guess the solution correctly. We can then stop the solution with this method and switch to the substitution method instead.

We have generated the closed form (solution) for $T(n)$. Done. ■

Example 28. Solve exactly $T(n) = 2T(n/2) + n$, $T(1) = 0$. Assume n is a power of 2.

Proof. Same recurrence but different formulation. The recurrence becomes formula (A1). We then substitute $n/2$, $n/2^2$, $n/2^3$, etc for n in (A1). This way we generate (A2), (A3), through (Ai). (Note that $T(n) = 2T(n/2) + n$ gets rewritten as $T(n/2^0) = 2T(n/2^1) + n/2^0$).

$$T(n/2^0) = 2T(n/2^1) + n/2^0 \quad (\text{A1})$$

$$T(n/2^1) = 2T(n/2^2) + n/2^1 \quad (\text{A2})$$

$$T(n/2^2) = 2T(n/2^3) + n/2^2 \quad (\text{A3})$$

...

$$T(n/2^{i-1}) = 2T(n/2^i) + n/2^{i-1} \quad (\text{Ai})$$

Moving forward, we use (A2) to rewrite (A1) then (A3), then all the way to (Ai).

$$\begin{aligned} T(n) &= 2T(n/2) + n \stackrel{(\text{A2})}{=} 2(2T(n/2^2) + n/2^1) + n/2^0 = 2^2T(n/2^2) + 2n \\ &= 2^2T(n/2^2) + 2n \stackrel{(\text{A3})}{=} 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n \\ &= 2^3T(n/2^3) + 3n \stackrel{(\text{A4})}{=} 2^3(2T(n/2^4) + n/2^3) + 3n = 2^4T(n/2^4) + 4n \\ &\dots \\ &= 2^iT(n/2^i) + i \cdot n \end{aligned}$$

Thus $T(n) = 2^iT(n/2^i) + i \cdot n$. We want to reach the base case i.e. $T(n/2^i) = T(1)$. This implies $n/2^i = 1$ i.e. $i = \lg n$. Continuing from where we stopped using $i = \lg n$ we have the following.

$$\begin{aligned} T(n) &= 2^i \cdot T(n/2^i) + i \cdot n \\ &= 2^{\lg n} \cdot T(n/2^{\lg n}) + \lg n \cdot n \\ &= n \cdot T(1) + n \lg n = n \cdot 0 + n \lg n \\ &= n \lg n \end{aligned}$$

We conclude that $T(n) = n \lg n$ is the closed form solution for $T(n)$. ■

Example 29. Solve exactly $T(n) = 2T(n/2) + n$, $T(2) = 1$. Assume n is a power of 2.

Proof. Same recurrence but different base case.

$$T(n) = 2T(n/2) + n \tag{A1}$$

$$T(n/2^1) = 2T(n/2^2) + n/2^1 \tag{A2}$$

$$T(n/2^2) = 2T(n/2^3) + n/2^2 \tag{A3}$$

$$T(n/2^3) = 2T(n/2^4) + n/2^3 \tag{A4}$$

...

$$T(n/2^{i-2}) = 2T(n/2^{i-1}) + n/2^{i-2} \tag{Ai-1}$$

$$T(n/2^{i-1}) = 2T(n/2^i) + n/2^{i-1} \tag{Ai}$$

We notice that the left-hand side appears in the right-hand side of the previous equation.

Thus we can multiply each equation with some quantity and then add up the result in equations.

For the base case we utilized now that $T(n/2^i) = T(2) = 1$. Solving $n/2^i = 2$ we obtain $i = \lg n - 1$ as before.

Continues on the following page.

The iteration method
Example mostly redone continued

$$\begin{aligned}T(n) &= 2T(n/2) + n \\2^1 \times T(n/2^1) &= 2^1 \times (2T(n/2^2) + n/2^1) \\2^2 \times T(n/2^2) &= 2^2 \times (2T(n/2^3) + n/2^2) \\2^3 \times T(n/2^3) &= 2^3 \times (2T(n/2^4) + n/2^3) \\&\dots \\2^{i-2} \times T(n/2^{i-2}) &= 2^{i-2} \times (2T(n/2^{i-1}) + n/2^{i-2}) \\2^{i-1} \times T(n/2^{i-1}) &= 2^{i-1} \times (2T(n/2^i) + n/2^{i-1})\end{aligned}$$

We then have

$$\begin{aligned}T(n) &= 2T(n/2) + n \\2T(n/2^1) &= 2^2T(n/2^2) + n \\2^2T(n/2^2) &= 2^3T(n/2^3) + n \\2^3T(n/2^3) &= 2^4T(n/2^4) + n \\&\dots \\2^{i-2}T(n/2^{i-2}) &= 2^{i-1}T(n/2^{i-1}) + n \\2^{i-1}T(n/2^{i-1}) &= 2^iT(n/2^i) + n\end{aligned}$$

We add up all the equations. All terms of the left hand side but the top first term cancel out. The $T(\cdot)$ terms of the right hand side cancel out as well except for the last term. The additive n terms of the right hand side are retained. The inner terms of the right hand side cancel out except for the last term. The end result is that

$$T(n) = 2^iT(n/2^i) + n.$$

We conclude, based on $i = \lg n - 1$ obtained on the previous page that the closed form solution is $T(n) = n \lg n - n/2$. ■

The iteration method

An instructional example

Instructional Example 6. Solve exactly $T(n) = 2T(n/2) + n$ where $T(4) = 12$. You may assume n is a power of 2.

Proof.

$$T(n/2^0) = 2T(n/2^1) + n/2^0 \quad (\text{A1})$$

$$T(n/2^1) = 2T(n/2^2) + n/2^1 \quad (\text{A2})$$

$$T(n/2^2) = 2T(n/2^3) + n/2^2 \quad (\text{A3})$$

$$T(n/2^3) = 2T(n/2^4) + n/2^3 \quad (\text{A4})$$

...

$$T(n/2^{i-1}) = 2T(n/2^i) + n/2^{i-1} \quad (\text{Ai})$$

Moving forward, we use (A2) to rewrite (A1), then we use (A3), then we use (A4) all the way to (Ai).

$$\begin{aligned} T(n) &= 2T(n/2) + n \stackrel{(\text{A2})}{=} 2(2T(n/2^2) + n/2^1) + n/2^0 = 2^2T(n/2^2) + 2n \\ &= 2^2T(n/2^2) + 2n \stackrel{(\text{A3})}{=} 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n \\ &= 2^3T(n/2^3) + 3n \stackrel{(\text{A4})}{=} 2^3(2T(n/2^4) + n/2^3) + 3n = 2^4T(n/2^4) + 4n \\ &\dots \\ &= 2^iT(n/2^i) + i \cdot n \end{aligned}$$

Thus $T(n) = 2^iT(n/2^i) + i \cdot n$. We want to reach the base case i.e. $T(n/2^i) = T(4) = 12$. This implies $n/2^i = 4$ i.e. $i = \lg n - 2$. Continuing from where we stopped using $i = \lg n - 2$ we have the following.

$$\begin{aligned} T(n) &= 2^i \cdot T(n/2^i) + i \cdot n \\ &= 2^{\lg n - 2} \cdot T(n/2^{\lg n - 2}) + (\lg n - 2) \cdot n \\ &= (n/4) \cdot T(4) + n(\lg n - 2) = n \lg n + n \end{aligned}$$

We conclude that $T(n) = n \lg n + n$ as needed. This is the closed form solution (answer). ■

The iteration method

A third example

The previous examples were straightforward and uncomplicated. Let's go back to the merge-sort recurrence to estimate an upper bound on the number of comparisons. We use $T(n)$ below to count for the number of comparisons (upper bound) of MergeSort.

Example 30. Solve exactly $T(n) = 2T(n/2) + n - 1$, $T(1) = 0$. You may assume n is a power of two.

Proof. Again we shall terminate recursion for $T(n/2^i)$ when $n/2^i = 1$ and solving for i we shall get $i = \lg n$. We shall also need the geometric sequence result $2^0 + 2^1 + \dots + 2^{i-1} = 2^i - 1$.

$$\begin{aligned} T(n) &= 2T(n/2) + n - 1 \\ &= 2(2T(n/2^2) + n/2 - 1) + n - 1 = 2^2T(n/2^2) + 2n - 1 - 2 \\ &= 2^2(2T(n/2^3) + n/2^2 - 1) + 2n - 1 - 2 = 2^3T(n/2^3) + 3n - 2^0 - 2^1 - 2^2 \\ &\dots \\ &= 2^iT(n/2^i) + i \cdot n - 2^0 - 2^1 - \dots - 2^{i-1} = 2^iT(n/2^i) + i \cdot n - (2^i - 1) \\ &= 2^iT(n/2^i) + i \cdot n - (2^i - 1) \quad \text{Now, Substitute } i = \lg n \\ &= 2^{\lg n} \cdot T(n/2^{\lg n}) + \lg n \cdot n - (2^{\lg n} - 1) \\ &= n \cdot T(1) + \lg n \cdot n - n + 1 = n \lg n - (n - 1) \end{aligned}$$

The previous recurrences were generating an additive term which was the same in each iteration: n . This one generates a term that is iteration dependent: $n - 2^{i-1}$ for iteration i . Thus for $i = 1$ we get $n - 1$, for $i = 2$ we get $n - 2$, and so on.

The answer for $T(n)$ with the $-(n - 1)$ low-order term) is the one we got through the recursion tree method as well. ■

The iteration method

A fourth example

Example 31. Solve exactly the recurrence $T(n) = 2T(n/2) + n$, $T(2) = 5$. Assume n is a power of 2.

Proof. Substituting $n/2$ for n in the recurrence we get. $T(n/2) = 2T((n/2)/2) + n/2 = 2T(n/2^2) + n/2$.

Substituting $n/4 = n/2^2$ for n we get. $T(n/2^2) = 2T((n/4)/2) + n/4 = 2T(n/2^3) + n/2^2$

Substituting $n/8 = n/2^3$ for n we get. $T(n/2^3) = 2T(n/2^4) + n/2^3$

Similarly we can substitute $n/2^4, \dots, n/2^{i-1}, \dots$ for n to resolve the 4, $\dots$, $i-1$ -st iteration all the way to the base case, and get similarly stated recurrences. We utilize all the derived recurrences to expand $T(n)$ by observing that $T(n)$ is $2T(n/2) + n$. Then we use the derivation for $T(n/2)$ to formulate the expression in terms of $T(n/2^2)$, then $T(n/2^2)$ is expressed in terms of $T(n/2^3)$ and so on $\dots$ in terms of $T(n/2^i)$. This goes on until we reach the based case of $n = 2$ since $T(2) = 5$ is given. We establish what value of i provides the base case by equating $T(2) = T(n/2^i)$ and solving for i . The unfolding of the recurrence stops then and all the terms unfolded get combined (summed). A closed form solution for $T(n)$ can then be found.

$$\begin{aligned} T(n) &= 2T(n/2) + n = 2(2T(n/2^2) + n/2) + n = 2^2T(n/2^2) + 2n \\ &= 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + n + 2n = 2^3T(n/2^3) + 3n \\ &= \dots = 2^iT(n/2^i) + i \cdot n \end{aligned}$$

From the boundary condition $T(2) = 5$, we decide when to stop the unfolding of $T(n)$. The expansion ends at $n/2^i = 2$ since then $T(n/2^i) = T(2) = 5$. We solve for i by taking logarithms base two of both sides. Then we get that $\lg n - i = 1$ ie $i = \lg n - 1$. Then, for $i = \lg n - 1$, we get that.

$$\begin{aligned} T(n) &= 2T(n/2) + n = \dots = 2^iT(n/2^i) + i \cdot n \\ &= 2^{\lg n - 1}T(n/2^{\lg n - 1}) + (\lg n - 1) \cdot n \\ &= (n/2)T(2) + (\lg n - 1) \cdot n = (n/2)5 + (\lg n - 1) \cdot n \\ &= (3n/2) + n \lg n \end{aligned}$$

Thus we have obtained the following solution to the recurrence: $T(n) = 3n/2 + n \lg n$. ■

Question 1. Is the obtained solution the correct one?

Answer. Yes, unless we missed something. ■

Question 2. How can we be sure?

Answer.

Check the solution i.e. make sure that $T(n) = 3n/2 + n \lg n$ is such that

- (a) $T(2) = 5$ (i.e. boundary condition can be verified), and
- (b) $T(n) = 2T(n/2) + n$ (i.e. recurrence can be verified).

Verify boundary condition.

$$T(2) = 3 \cdot 2/2 + 2 \lg 2 = 3 + 2 = 5.$$

Since $T(2) = 5$ indeed by the boundary condition, our solution satisfies the boundary condition.

Verify recurrence. We obtained the solution

$$T(n) = 3n/2 + n \lg n.$$

Substituting $n/2$ for n we get that

$$T(n/2) = 3n/4 + (n/2) \lg n/2 = 3n/4 + (n/2)(\lg n - 1) = n/4 + (n/2) \lg n.$$

We start from the right hand side of the recurrence using the preceding equality.

$$2T(n/2) + n = 2(n/4 + n/2 \lg n) + n = 3n/2 + n \lg n = T(n)$$

The last term is $T(n)$. We have thus proved that for $T(n) = 3n/2 + n \lg n$, we have that $T(n) = 2T(n/2) + n$, i.e. the recurrence is satisfied.

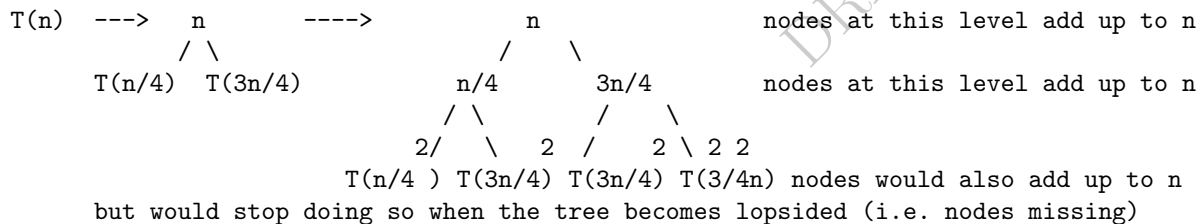
By (a) and (b) the solution satisfies both the recurrence and the boundary condition, i.e. it is indeed a solution to the recurrence. ■

The recursion tree method

A fifth example

We have already solved two recurrences with this method. The first one was the running time of the Fibonacci divide-and-conquer time and space inefficient solution. The second one was the running time (comparison upper bound) of MergeSort.

If the recurrence is complex then the iteration method will not work (easily). Moreover floors and ceilings will complicate things. Suppose $T(n) = T(n/4) + T(3n/4) + n$. Try to use a recursion tree. Start with a tree with one node labeled $T(n)$. Expand the $T(n)$ node by replacing it with a new node that is labeled with the non-recursive part of the recurrence i.e. n . Make that node have two children, the $T(n/4)$ and $T(3n/4)$ recursive parts of the recurrence. (If one had $2T(n/2)$ instead write it as $T(n/2) + T(n/2)$). Then expand similarly (using the recurrence) the two nodes labeled $T(n/4)$ and $T(3n/4)$. Expansion is shown. The recursion tree can be lopsided (Fibonacci case) or not. In this case it does not show after two iterations but it will be lopsided. The left-most path goes $n, n/4, n/4^2$ and so on. It reaches some constant value, say 1, after $n/4^i = 1$ i.e. $\lg(n)/2 = \log_4 n$ iterations and it is $\lg n/2$ levels deep (or high). The rightmost path it goes $n, 3n/4, (3/4)^2 n$, and it reaches some constant value 1 after $(3/4)^i n = 1$ which solves for $i = \log_{4/3} n = \lg n / \lg(4/3)$. (Note that parentheses are dropped and thus the last expression should be read $\lg(n)/\lg(4/3)$.) The right path is longer/deeper/higher by maybe a factor of 4 or more. After i iterations the tree's height is i . The tree is lopsided (the Fibonacci sequence tree for the recursive solution was also lopsided). Deepest subtree is the rightmost one of depth $\lg n / \lg(4/3)$. Total time is thus $O(n \lg n)$. A Θ becomes too cumbersome to derive.



Note 9. The $T(n/4^2)$ and $T(3n/4^2)$ that appear in the third level of the tree in the figure above look awkward. And even more the $T((3^2/4^2)n)$ last term shown.

Notice

Problem Set with Solutions

These Problem Sets are not for credit. They are provided as are for your practice. Several problems are provided with solutions. These can be discussed with the instructor after a student has studied the relevant material. Some other problems are provided with either no solutions at all or a hint of a solution. They are there for your reference and will not be discussed further. For some solutions provided you might have to fill in details related to calculations or intermediate steps.

Before you try these problems: (a) Study the designated textbook, (b) use the Subject notes (typewritten notes available through the course web-page from the password-protected section C) to review your study of the designated textbook material, (c) resolve the examples done in class, (d) then practice with some of these problems PS (Problem Set), and then if you have time, (e) solve some of the problems of the designated textbook, and (f) solve more problems from the PS especially the ones with minimal solution information.

Several PS problems appeared in past exams or homeworks.

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set

Problem 1.

Rank the following function by order of growth; that is, find an arrangement $g_1, g_2, g_3, \dots$ of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $g_3 = \Omega(g_4)$, $\dots$. Partition your list in equivalence classes such that $g(n)$ and $h(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.

For example for functions $\lg n, n, n^2$, and $2^{\lg n}$ you could write: $n^2, \{n, 2^{\lg n}\}, \lg n$

(a) 1st group of functions to be ranked!

$$n, (n-1)!, (\lg n)^{\lg n}, n \lg n, \lg(n!), 2^{\lg n}, n^{\lg \lg n}, n!.$$

Proof. (a) 1st group

$$n!, (n-1)!, \{n^{\lg \lg n}, (\lg n)^{\lg n}\}, \{n \lg n, \lg(n!)\}, \{n, 2^{\lg n}\}.$$

■

Problem Set Problem 2.

Replace ? with the tightest of o, ω, Θ that best describes the relationship between the two sides.

- (1) $\lg(n!) = ? (n \lg n)$.
- (2) $\lg(n!) = ? (n^2)$.
- (3) $n^n = ? (n!)$.
- (4) $2^n = ? (n!)$.
- (5) $2^{\lg n} = ? (n)$.
- (6) $n/\lg n = ? (n)$.
- (7) $2^{2n} = ? ((\frac{9}{2})^n)$.
- (8) $9^n = ? (n!)$.
- (9) $2^{\lg n} = ? (n^2/\lg n)$.
- (10) $4^{n/2} = ? ((8^{n/3}))$.
- (11) $n! = ? (n \lg n)$.
- (12) $4^n = ? (3^n)$.
- (13) $(n+2)! = ? ((n+1)!)$.
- (14) $2^{n \lg n} = ? (n^n)$.

Proof. (1) $\lg(n!) = \Theta(n \lg n)$.

- (2) $\lg(n!) = o(n^2)$.
- (3) $n^n = \omega(n!)$.
- (4) $2^n = o(n!)$.
- (5) $2^{\lg n} = \Theta(n)$.
- (6) $n/\lg n = o(n)$.
- (7) $2^{2n} = o((\frac{9}{2})^n)$.
- (8) $9^n = o(n!)$.
- (9) $2^{\lg n} = o(n^2/\lg n)$.
- (10) $4^{n/2} = \Theta(8^{n/3})$.
- (11) $n! = \omega(n \lg n)$.
- (12) $4^n = \omega(3^n)$.
- (13) $(n+2)! = \omega((n+1)!)$.
- (14) $2^{n \lg n} = \Theta(n^n)$.

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

■

Problem 3.

- (a) Show that $(n - 3)^2 = \Theta(n^2)$ by providing the n_0, c_1, c_2 in the definition of the Θ notation.
 (b) Show that $(n + 3)^2 = \Theta(n^2)$ by providing the n_0, c_1, c_2 in the definition of the Θ notation.

Proof. Part (a): We show first that $(n - 3)^2 = O(n^2)$ and then that $(n - 3)^2 = \Omega(n^2)$.

Case 1. Show that $(n - 3)^2 = O(n^2)$

For all $n \geq 1$ we have that

$$\begin{aligned} (n - 3)^2 &= n^2 - 6n + 9 \\ &\leq n^2 + 0 \cdot n^2 + 9 \\ &\leq n^2 + 0 \cdot n^2 + 9 \cdot 1 \\ &\leq 10n^2 \end{aligned}$$

Therefore there exist constant $n_2 = 1$ and $c_2 = 10$ such that $(n - 3)^2 \leq c_2 n^2$ for all $n \geq n_2$. This proves Claim 1.

Technique 1. We upper bound $n^2 \pm An \pm B$, for positive A, B as follows: $n^2 \pm An \pm B \leq (1 + A + B)n^2$.

Case 2. Show that $(n - 3)^2 = \Omega(n^2)$

Technique 1 can not be used in this case. However, since $9 \geq 0$ we can eliminate 9 easily.

$$\begin{aligned} (n - 3)^2 &= n^2 - 6n + 9 \\ &\geq n^2 - 6n \end{aligned}$$

The next step is non-trivial. We bound $n^2 - 6n$ from below by $n^2/2$. This is so as long as

$$\begin{aligned} n^2 - 6n &\geq n^2/2 \Leftrightarrow \\ n^2/2 &\geq 6n \Leftrightarrow \\ n/2 &\geq 6 \Leftrightarrow \\ n &\geq 12 \end{aligned}$$

Problem Set Continued

We can do this as long as n is not zero; this is true since for all cases we assume that at least $n \geq 1$. Therefore there exist constant $n_1 = 12$ and $c_1 = 1/2$ such that $(n - 3)^2 \geq n^2/2$ for all $n \geq n_1 = 12$. This proves Claim 2. In order to show that $(n - 3)^2 = \Theta(n^2)$ we need to establish c_1, c_2 and n_0 by combining the results of Claim 1 and Claim 2. Pick $c_1 = 1/2$ and $c_2 = 10$ and $n_0 = 12$.

(b) (*O* part first) To establish an upper bound we observe that,

$$(n + 3)^2 = n^2 + 6n + 9 \leq n^2 + 6n^2 + 9n^2 = 16n^2.$$

(c) (*Omega* part next) To establish a lower bound we observe that since n is non-negative,

$$(n + 3)^2 = n^2 + 6n + 9 \geq 1n^2$$

Therefore for $c_1 = 1, c_2 = 16$ and for all $n \geq 1 = n_0$ we have that $(n + 3)^2 = \Theta(n^2)$. ■

Problem Set

Problem 4.

Give a **TRUE** or **FALSE** for each one of the statements below. Any answer other than a full TRUE or FALSE will be considered wrong.

- (a) The asymptotic solution of $T(n) = 16T(n/2) + n^3$ is $T(n) = \Theta(n^3)$.
- (b) The asymptotic solution of $T(n) = 8T(n/2) + n^3 \lg n$ is $T(n) = \Theta(n^3 \lg n)$.
- (c) The asymptotic solution of $T(n) = 2T(n/2) + \lg n$ is $T(n) = \Theta(n \lg n)$.
- (d) Let $f(n) > 0$ be a positive function for all $n \geq 0$. The following is true for every f : $f(n) = \Theta(f(n/2))$.

Proof. FALSE, FALSE, FALSE, FALSE



Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set Problem 5.

Master Method.

Solve the following recurrences. You may assume $T(1) = \Theta(1)$, where necessary. Make your bounds as tight as possible. Use asymptotic notation to express your answers. Justify your answers.

- a. $T(n) = 4T(n/2) + 3n^2$
- b. $T(n) = 32T(n/4) + 3n^2$
- c. $T(n) = 4T(n/2) + n^3$
- d. $T(n) = 9T(n/3) + n$.
- e. $T(n) = 25T(n/5) + n^2$.

Proof. One can use the master method for all three of them.

a. $T(n) = 4T(n/2) + 3n^2$. It has $a = 4 \geq 1$, $b = 2 > 1$, $f(n) = 3n^2$. $n^{\log_b a} = n^{\log_2 4} = n^2$. Obviously $n^2 = f(n) = \Theta(n^{\log_b a})$. Case 2 of the Master method applies and thus $T(n) = \Theta(n^2 \lg n)$.

b. $T(n) = 32T(n/4) + 3n^2$. Note that $32 \cdot 32 = 1024 = 2^{10}$, $4^5 = 1024$ i.e. $4^{5/2} = 1024^{1/2} = 32$. It has $a = 32 \geq 1$, $b = 4 > 1$, $f(n) = 3n^2$. $n^{\log_b a} = n^{\lg_4 32} = n^{5/2}$. Obviously $n^2 = f(n) = O(n^{\log_b a - \epsilon}) = O(n^2)$, by choosing $\epsilon = 1/2 > 0$. Case 1 of the Master method applies and thus $T(n) = \Theta(n^{5/2})$.

c. $T(n) = 4T(n/2) + n^3$. It has $a = 4 \geq 1$, $b = 2 > 1$, $f(n) = n^3$. $n^{\log_b a} = n^{\lg_2 4} = n^2$. Obviously $n^3 = f(n) = \Omega(n^{\log_b a + \epsilon}) = \Omega(n^3)$, by choosing $\epsilon = 1 > 0$. Case 3 of the Master method applies since also $af(n/b) = 4(n/2)^3 = n^3/2 \leq cn^3 = (1/2)n^3$, by choosing $c = 1/2 < 1$, and thus $T(n) = \Theta(n^3)$.

d. We use the master method. $a = 9 \geq 1$, $b = 3 > 1$, $f(n) = n$, and $\log_b a = 2$. Then $n^{\lg_b a} = n^2$. As $f(n) = n = O(n^{\lg_b a - 1}) = O(n)$, case one of the master method applies, and thus $T(n) = \Theta(n^2)$.

e. We use the master method. $a = 25 \geq 1$, $b = 5 > 1$, $f(n) = n^2$, and $\log_b a = 2$. Then $n^{\lg_b a} = n^2$. As $f(n) = n^2 = \Theta(n^{\lg_b a}) = \Theta(n^2)$, case two of the master method applies, and thus $T(n) = \Theta(n^2 \lg n)$. ■

Problem Set
Problem 6.

Solve the following recurrence relations by providing asymptotically tight bounds. You only need to provide the bound, intermediate derivations are not required. If no boundary case is given, the choice is yours. You may assume that $T(n)$ is positive and monotonically increasing, if you need to do so.

$$T(n) = T(n/4) + n.$$

$$T(n) = 8T(n/4) + n!.$$

$$T(n) = 2T(n/2) + \lg n.$$

$$T(n) = 2T(n/2) + T(n/4) + n^2.$$

$$T(n) = T(n/2) + n.$$

$$T(n) = T(n/2) + \lg n.$$

$$T(n) = 2T(n/2) + T(n/4) + n^3.$$

Proof. Practice makes perfect! You practice! ■

Problem Set Problem 7.

Recursion Tree/Iteration method.

Let n be a power of two. Solve exactly the following recurrence.

$$T(n) = T(n/2) + n, \text{ where } T(1) = 2.$$

Proof. We could use the master method (case three) to derive that $T(n) = \Theta(n)$ (using $\epsilon = 1/2$ and $c = 1/2$), if boundary condition $T(1) = 2$ was not explicitly stated. Since it is, we use the recursion tree/iteration method, in fact the iteration method.

$$\begin{aligned} T(n) &= T(n/2) + n \\ &= T(n/4) + n/2 + n \\ &= T(n/8) + n/4 + n/2 + n \\ &= T(n/2^i) + n/2^{i-1} + \dots + n/4 + n/2 + n \\ &= T(n/2^{\lg n}) + n/(n/2) + \dots + n/2 + n \\ &= T(1) + 2 + 4 + 8 + \dots + n/2 + n \\ &= 2 + 2 + 4 + 8 + \dots + n/2 + n \\ &= 1 + (1 + 2 + 4 + 8 + \dots + n/2 + n) \\ &= 1 + (2n - 1) = 2n \end{aligned}$$

We used the fact that $1 + 2 + 4 + \dots + n = 2n - 1$.

■

Problem Set
Problem 8.

Solve exactly using the iteration method the following recurrence. You may assume that n is a power of 4, i.e. $n = 4^k$.

$$T(n) = 4T(n/4) + n^2, \text{ where } T(1) = 4.$$

Proof. $T(n) = 4T(n/4) + n^2$. We use the iteration method. **This first example will be done in fine detail showing all calculations.**

$$\begin{aligned}
 T(n) &= 4T(n/4) + n^2 \\
 &= 4 \left(4T\left(\frac{n}{4^2}\right) + \left(\frac{n}{4}\right)^2 \right) + n^2 \\
 &= 4^2 T\left(\frac{n}{4^2}\right) + \frac{n^2}{4^1} + \frac{n^2}{4^0} \\
 &= 4^2 \left(4T\left(\frac{n}{4^3}\right) + \left(\frac{n}{4^2}\right)^2 \right) + \frac{n^2}{4^1} + \frac{n^2}{4^0} \\
 &= 4^3 T\left(\frac{n}{4^3}\right) + \frac{n^2}{4^2} + \frac{n^2}{4^1} + \frac{n^2}{4^0} \\
 &= \dots \\
 &= 4^i T\left(\frac{n}{4^i}\right) + \frac{n^2}{4^{i-1}} + \dots + \frac{n^2}{4^1} + \frac{n^2}{4^0} \\
 &= 4^i T\left(\frac{n}{4^i}\right) + n^2 \left(\frac{1}{4^{i-1}} + \dots + \frac{1}{4^1} + \frac{1}{4^0} \right) \\
 &= 4^i T\left(\frac{n}{4^i}\right) + n^2 \left(\frac{\frac{1}{4^i} - 1}{\frac{1}{4} - 1} \right) \\
 &= 4^i T\left(\frac{n}{4^i}\right) + n^2 \left(\frac{1 - \frac{1}{4^i}}{1 - \frac{1}{4}} \right) \\
 &= 4^i T\left(\frac{n}{4^i}\right) + \frac{4n^2}{3} \left(1 - \frac{1}{4^i} \right)
 \end{aligned}$$

Problem Set Continued

The base case is $T(1) = 4$. We set $n/4^i = 1$, i.e. $n = 4^i$. If we solve this for i , we get $i = \log_4 n = \lg n/2$. Then $T(n/4^i) = T(1) = 4$. In addition $4^i = n$. Therefore the formula for $T(n)$ becomes.

$$\begin{aligned} T(n) &= 4^i T\left(\frac{n}{4^i}\right) + \frac{4n^2}{3} \left(1 - \frac{1}{4^i}\right) \\ &= nT(1) + \frac{4n^2}{3} (1 - 1/n) \\ &= 4n + \frac{4n^2}{3} (1 - 1/n) \\ &= \frac{4n^2}{3} + \frac{8n}{3} \end{aligned}$$

Note. It is easy to verify for example that $T(1) = \frac{4 \cdot 1^2}{3} + \frac{8 \cdot 1}{3} = 4$. This can serve as a checking mechanism that the solution that you have established satisfies at least the base case. ■

Problem Set
Problem 9.

Solve exactly using the recursion tree method the following recurrence. You may assume that n is a power of 3.

$$T(n) = 9T(n/3) + n^2, \text{ where } T(1) = 1.$$

Proof.

$$\begin{aligned} T(n) &= 9T(n/3) + n^2 \\ &= 9(9T(n/3^2) + n^2/9) + n^2 \\ &= 9^2T(n/3^2) + n^2 + n^2 \\ &= 9^iT(n/3^i) + n^2 + \dots + n^2 + n^2 \\ &= 9^iT(n/3^i) + i \cdot n^2 \end{aligned}$$

A first observation is that the number of n^2 's we have is equal to the power of 9 (or equivalently, of 3) in the coefficient of $T(\cdot)$ (or the denominator of $T(n/\cdot)$ respectively). In the base case $T(1) = 1$, therefore we choose $n/3^i = 1$, which gives $i = \lg n / \lg 3$. Then, for this i , $T(n/3^i) = T(1) = 1$. We note that $9 = 3^2 = (2^{\lg 3})^2 = 2^{2 \lg 3}$. We therefore get that

$$\begin{aligned} T(n) &= 9^i T(n/3^i) + in^2 \\ &= 9^{\lg n / \lg 3} T(1) + n^2 \lg n / \lg 3 \\ &= 3^{2 \lg n / \lg 3} + n^2 \lg n / \lg 3 \\ &= 2^{2 \lg 3 \lg n / \lg 3} + n^2 \lg n / \lg 3 \\ &= 2^{2 \lg n} + n^2 \lg n / \lg 3 \\ &= n^2 + n^2 \lg n / \lg 3 \end{aligned}$$

It is easy to verify that this solution satisfies both the recurrence and the boundary condition. ■

Problem Set
Problem 10.

Solve the following recurrences. Make your bounds as tight as possible. You may assume that n is a power of two.

$$T(n) = T(n/2) + \lg n, \quad \text{where } T(2) = 1.$$

Proof. $T(n) = T(n/2) + \lg n$. We use the iteration method, noting that $T(2) = 1$.

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + \lg n \\ &= \left(T\left(\frac{n}{2^2}\right) + \left(\lg \frac{n}{2}\right)\right) + \lg n \\ &= T\left(\frac{n}{2^2}\right) + \lg \frac{n}{2^1} + \lg \frac{n}{2^0} \\ &= \dots \\ &= T\left(\frac{n}{2^i}\right) + \lg \frac{n}{2^{i-1}} + \dots + \lg \frac{n}{2^1} + \lg \frac{n}{2^0} \end{aligned}$$

The base case is $T(2) = 1$. We set $n/2^i = 2$, i.e. $n = 2^{i+1}$. If we solve this for i , we get $i = \lg n - 1$. Then $T(n/2^i) = T(2) = 1$. In addition $2^i = n/2$. Therefore the formula for $T(n)$ becomes

$$\begin{aligned} T(n) &= T\left(\frac{n}{2^i}\right) + \lg \frac{n}{2^{i-1}} + \dots + \lg \frac{n}{2^1} + \lg \frac{n}{2^0} \\ &= T\left(\frac{n}{2^{\lg n - 1}}\right) + \lg \frac{n}{2^{\lg n - 2}} + \dots + \lg \frac{n}{2^1} + \lg \frac{n}{2^0} \\ &= T(2) + (\lg n - (\lg 2^{\lg n - 1})) + (\lg n - (\lg 2^{\lg n - 2})) + \dots + (\lg n - \lg 2^1) + (\lg n - \lg 2^0) \\ &= 1 + (\lg n - (\lg n - 2)) + (\lg n - (\lg n - 3)) + \dots + (\lg n - 1) + (\lg n - 0) \\ &= 1 + 2 + 3 + \dots + (\lg n - 1) + (\lg n) \\ &= \sum_{j=1}^{\lg n} j \\ &= \frac{\lg n (\lg n + 1)}{2} \end{aligned}$$

Problem Set
Continued

Therefore, $T(n) = \lg n(\lg n + 1)/2$. It is easy to verify that $T(2)$ is indeed 1. ■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 11.

Solve exactly, assuming n is a power of two, $T(n) = 2T(n/2) + \lg n$ where $T(1) = 1$.

Proof.

$$T(n) = 2T(n/2) + \lg n = \dots = 2^{\lg n} T(1) + 2^{\lg n - 1} \lg \frac{n}{2^{\lg n - 1}} + \dots + 2^i \lg \frac{n}{2^i} + \dots + 2^1 \lg \frac{n}{2^1} + 2^0 \lg \frac{n}{2^0}.$$

Therefore, by substituting for $T(1) = 1$ we get

$$T(n) = 2^{\lg n} + \lg n(1 + 2 + \dots + 2^{\lg n - 1}) - \sum_{k=0}^{\lg n - 1} k 2^k = n + \lg n(n - 1) - \lg n n + 2n - 2 = 3n - \lg n - 2.$$

■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 12.

The recurrence to solve is the following one for n a power of two.

$$T(n) = 2T(n/2) + (n \lg n), \text{ where } T(1) = 1.$$

Proof. We solve the recurrence using the iteration method. Below, we set $n/2^i = 1$ to satisfy the boundary condition $T(1)$. Then, $i = \lg n$.

$$\begin{aligned} T(n) = T(2^k) &= 2T(n/2) + (n \lg n) \\ &= 2^2T(n/4) + 2(n/2) \lg(n/2) + n \lg n \\ &= 2^2T(n/4) + n \lg(n/2) + n \lg n \\ &= 2^i T(n/2^i) + n \lg(n/2^{i-1}) + \dots + n \lg(n/2^0) \\ &= 2^i T(n/2^i) + n \sum_{k=0}^{i-1} \lg(n/2^k) = 2^{\lg n} T(1) + n \sum_{k=0}^{\lg n - 1} (\lg(n) - k \lg 2) \\ &= n + n \sum_{k=0}^{\lg n - 1} \lg n - n \sum_{k=0}^{\lg n - 1} k \\ &= n + n \lg^2 n - n(\lg n - 1)(\lg n)/2 \\ &= n + n \lg n (\lg n + 1)/2 \end{aligned}$$

Therefore, $T(n) = \Theta(n \lg^2 n)$. ■

Problem Set

Problem 13.

Solve the following recurrences; make your bounds as tight as possible. Use strong induction if necessary.

$$T(n) = T(n/2) + T(n/3) + 32n, \quad \text{where } T(1) = 100.$$

Proof. Since $T(n) = T(n/2) + T(n/3) + 32n$ we have for $n = 1$ from the base case that $T(1) = 100$. We show that $T(n) = \Omega(n)$. By the recurrence we have that $T(n) \geq 32n$ for all $n > 1$ since $T(n/2), T(n/3)$ are non-negative and the other term is $32n$. In addition, $T(1) = 100 \geq 32 \cdot 1 = 32$. Therefore $T(n) \geq 32n$ for $n = 1$ as well. Therefore $T(n) = \Omega(n)$.

We now prove that $T(n) = O(n)$ or equivalently that there exist positive constants n_0, c_2 such that $T(n) \leq c_2n$ for all $n \geq n_0$. Proof is by induction on n .

Base Case. $T(1) = 100 \leq c_2 \cdot 1$ is true provided that $c_2 \geq 100$. This completes the base case.

Inductive Step. Let $T(i) \leq c_2i$ be true for all $1 \leq i < n$. We want to show that $T(n) \leq c_2n$. Since $T(i) \leq c_2i$ for all $i < n$, it is also true for $i = n/2$ and $i = n/3$ as well, and thus by the inductive hypothesis we obtain $T(n/2) \leq c_2n/2$ and $T(n/3) \leq c_2n/3$. Then we use these two inequalities in the recurrence to obtain.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + 32n \\ &\leq c_2n/2 + c_2n/3 + 32n \\ &= 5c_2n/6 + 32n \end{aligned}$$

In order to complete the inductive step we need to prove that $T(n) \leq c_2n$ or equivalently $5c_2n/6 + 32n \leq c_2n$. The latter is equivalent to $32n \leq c_2n/6$ or $c_2 \geq 192$. Therefore the inductive step is completed and $T(n) \leq c_2n$ for any $c_2 \geq 192$. In conclusion, combining the inductive step and the base case we obtain that $T(n) \leq c_2n$ for all $n \geq 1$ as long as $c_2 \geq 192$. We can fix $c_2 = 192$ and thus show that $T(n) = O(n)$.

Therefore since $T(n) = O(n)$ and $T(n) = \Omega(n)$ we obtain that $T(n) = \Theta(n)$. The constants in the definition are 1, 192 and for n_0 we can still use 1.

■

Problem Set Problem 14.

Solve the recurrence $T(n) = T(n/2) + T(2n/3) + n$, where $T(1) = \Theta(1)$.

Proof. Attempt 1. We try $T(n) = O(n^2)$, i.e. $T(n) \leq cn^2$ for some constant $c, n_0 > 0$ and for all $n \geq n_0$. Using by the inductive hypothesis $T(n/2) \leq c(n/2)^2$ and $T(2n/3) \leq c(2n/3)^2$ and

$$T(n) \leq cn^2/4 + 4cn^2/9 + n = c(25/36)n^2 + n \leq cn^2$$

for large enough n and $c > 0$. The bound, however, is not the best possible one.

Attempt 2. We try $T(n) \leq cn^\alpha$ where α is a constant such that $1 < \alpha < 2$. Similarly as before,

$$T(n) \leq cn^\alpha((1/2)^\alpha + (2/3)^\alpha) + n \leq cn^\alpha$$

The latter inequality holds asymptotically for large n provided that $(1/2)^\alpha + (2/3)^\alpha < 1$. If this value of $(1/2)^\alpha + (2/3)^\alpha$ is 0.99 then $0.99cn^\alpha + n$ is indeed smaller than cn^α asymptotically for large n .

Attempt 3. We try $T(n) \leq cn^\alpha - 6n$ where α are constant such that $1 < \alpha < 2$ and $\beta > 1$. (There is no magic in choosing a 6: it relates to $1/2$ and $2/3$ after originally choosing a β instead.) Similarly as before,

$$T(n) \leq cn^\alpha((1/2)^\alpha + (2/3)^\alpha) - 2n/3 + n \leq cn^\alpha - 6n$$

The latter inequality holds asymptotically for large n provided that $(1/2)^\alpha + (2/3)^\alpha < 1$. If this value of $(1/2)^\alpha + (2/3)^\alpha$ is 0.99 then $0.99cn^\alpha - 6n$ is indeed smaller than $cn^\alpha - 6n$ asymptotically for large n . A numerical computation of an α yields one in the range $[1.293, 1.294]$. The exact value is of no interest; only the fact that it is between 1 and 2! ■

Problem Set
Problem 15.

Substitution Method.

There are more examples with the substitution method in the pages to follow.

Solve the following recurrences. You may assume $T(1) = \Theta(1)$, where necessary. Make your bounds as tight as possible. Use asymptotic notation to express your answers. Justify your answers.

$$T(n) = T(n/4) + T(3n/4) + n \lg n.$$

Proof. We use the substitution method to solve this recurrence. We guess as a solution $T(n) = O(n \lg^2 n)$ ie $T(n) \leq cn \lg^2 n$, for some positive constant $c > 0$ and for all $n \geq n_0$. We use induction to prove that this is indeed a solution to the recurrence.

$$\begin{aligned} T(n) &= T(n/4) + T(3n/4) + n \lg n \\ &\leq (cn/4) \lg^2 (n/4) + (3cn/4) \lg^2 (3n/4) + n \lg n \\ &= c(n/4)(\lg n - 2)^2 + c(3n/4)(\lg n + \lg 3 - 2)^2 + n \lg n \\ &= c(n/4)(\lg n - 2)^2 + c(3n/4)(\lg n)^2 + n \lg n \\ &\leq cn \lg^2 n + n - cn \lg n \\ &\leq cn \lg^2 n \end{aligned}$$

for any $n \geq 4$ and $c \geq 1$. We choose for boundary condition $T(2) = 2$ so that $T(n) \leq cn \lg^2 n$ would result in $2 \leq c2 \lg^2 2 = 2c$, ie $c \geq 1$. Therefore for any $n \geq 4$ and any $c \geq 1$, we have $T(n) \leq cn \lg^2 n$. ■

Problem 16.

The Fast vs the (furiously) Good. For some problem P we have implemented two algorithms that solve P . An algorithm **Good** runs in time $g(n)$ and an algorithm **Fast** runs in time $f(n)$, where n is problem size of P . Answer the following question Q for each one of the following three cases C1, C2 and C3. Justify your answer.

Q. Is *Fast* faster than *Good* for all $n \geq n_0$, for some constant n_0 ?

C1. $g(n) = \Omega(f(n) \lg n)$.

C2. $g(n) = \Theta(f(n) \lg n)$.

C3. $g(n) = O(f(n) \lg n)$.

Proof. Asymptotic notation roundup: Think!

■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 17.

Solve the following recurrences associated with three problems introduced in Subject 0. First name the problem associated with a recurrence and then solve the recurrence as best as you can (i.e. tightest possible bound) asymptotic or not! You may assume n is a power of 2. Formulate appropriate base case as well.

- (a) $T(n) = T(n/2) + 1$.
- (b) $T(n) = 2T(n/2) + (n - 1)$.
- (c) $T(n) = T(n - 1) + (n - 1)$.

Proof. (a) $T(1) = 1$, Binary search and $T(n) = \lg n + 1$ for (upper bound on) number of comparison in worst-case thus determining running time also but using a rougher analysis ($n/2$ should have been $(n - 1)/2$).

(b) $T(2) = 1$, Merge sort and $T(n) = n \lg n - (n - 1)$ for number of comparisons in worst-case thus determining running time also.

(c) $T(1) = 0$, Insertion sort (recursive formulation) $T(n) = n(n - 1)/2$ for number of comparisons in the worst-case determining running time also. ■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 18.

Solve the recurrence $T(n) = T(an) + T(bn) + n$ where $a + b < 1$ and $a > 0$ and $b > 0$ are constant. In the following problem, we address the case $a + b = 1$.

Proof. We could use the substitution method to solve this recurrence. (1.) Ω part: $T(n) = \Omega(n)$ is straightforward as $T(n) = T(an) + T(bn) + n \geq n$.

(2.) O part: We show $T(n) = O(n)$ i.e. we show that there exist c, n_0 positive and constant such that $T(n) \leq cn$ for $\forall n \geq n_0$. Proof by induction. Since $an < n$ and $bn < n$ the induction hypothesis for $T(an)$ and $T(bn)$ gives

$$\begin{aligned} T(n) &= T(an) + T(bn) + n \\ &\leq c(an) + c(bn) + n \\ &= cn(a + b) + n \end{aligned}$$

For the last expression to be at most cn we need to have

$$\begin{aligned} cn(a + b) + n &\leq cn \\ n &\leq c(1 - a - b)n \\ 1 &\leq c(1 - a - b) \end{aligned}$$

Since $a + b < 1$ we have $1 - a - b > 0$. Therefore if we choose c such that $c > 1/(1 - a - b)$ the last inequality is true and the inductive step is proven. ■

Problem Set Problem 19.

Consider the following variation of MergeSort: Instead of an even split, we split 10-percent on one-size and 90-percent on the other. Then the recurrence for the running time becomes

$$T(n) = T(0.1n) + T(0.9n) + \Theta(n)$$

In general we might have an an split and $(1-a)n$ split where $0 < a \leq 1/2$. The recurrence then becomes after simplifying the last term (one may use cn instead for some constant $c > 0$).

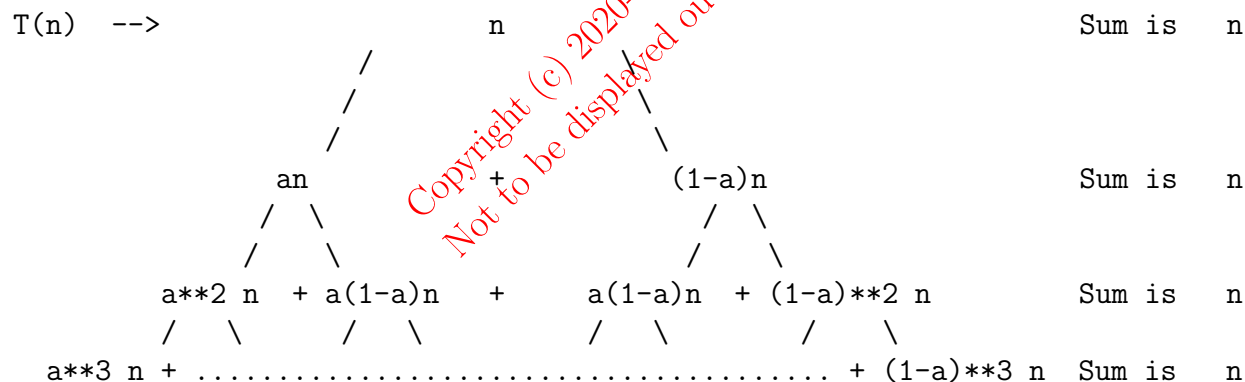
$$T(n) = T(an) + T((1-a)n) + n$$

Proof. A solution using the Recursion tree method. Before you go back to Problem 12 of this Problem Set, note the catch there with $a + b < 1$ but in our case $a + (1-a) = 1$! This recurrence is easier solved by the recursion tree approach (used in Merge-Sort and Fibonacci sequence in class). A very flat tree is shown in textual/graphic form below. Isolate the leftmost and the rightmost paths of this tree.

The leftmost path is $n, an, a^2n, a^3n, \dots$ and the rightmost path is $n, (1-a)n, (1-a)^2n, (1-a)^3n, \dots$

How long is the leftmost path? It is L long if $a^L n = 1$. Taking logarithms base two $L \lg a + \lg n = 0$, i.e. $L = -\lg n / \lg a$. Substitute 0.1 for a and we get the desired answer.

How long is the rightmost path? It is R long if $(1-a)^R n = 1$. Taking logarithms base two $R \lg(1-a) + \lg n = 0$, i.e. $R = -\lg n / \lg(1-a)$. Substitute 0.1 for a or 0.9 for $1-a$ and we get the desired answer.



Problem Set Continued

Remark. As a byproduct of this problem, we conclude that in the context also of the QuickSort algorithm of Subject 6, if there is a PARTITION function (we shall call the implementation of the designated textbook GTP) that always causes a 10-percent and 90-percent split or in the more general case an an -($1-a$) n split, then the running time of quicksort that uses this partition is going to be $O(n \lg n)$. This is because we just proved that the depth of the recursion is $O(\lg n)$ and at each level, partition spends total time $O(n)$.

A solution using the Substitution method. We could use the substitution method to solve this recurrence. (1.) Ω part: $T(n) = \Omega(n \lg n)$ uses a line of argument similar to the O part to follow. (2.) O part: We show $T(n) = O(n \lg n)$ i.e. there exist positive constant c, n_0 such that $T(n) \leq cn \lg n$, for all $n \geq n_0$ by utilizing strong induction. Since $an < n$ and $(1-a)n < n$ the induction hypothesis applies to $T(an)$ and $T((1-a)n)$.

$$\begin{aligned} T(n) &= T(an) + T((1-a)n) + n \\ &\leq cn \lg(an) + c(1-a)n \lg((1-a)n) + n \\ &= cn \lg n + cn(a \lg a + (1-a) \lg(1-a)) + n \leq cn \lg n - cnA + n \\ &\leq cn \lg n + (1-c_2A)n \leq cn \lg n \quad \text{for } c > 1/A. \end{aligned}$$

where we used that the entropy function i.e. the expression $a \lg a + (1-a) \lg(1-a)$ is negative for $0 < a < 1$. Let that negative value be called $-A = a \lg a + (1-a) \lg(1-a)$, where $A > 0$. We obtain the final answer (i.e. inductive step) by making sure that the term $(1-cA)n$ is negative which is so for $1-cA < 0$ i.e. $c > 1/A$. Note that A is constant because a is constant. **If a were not a constant, then A would not have been a constant either, and c would not have been bounded by a constant either.**

■

Problem Set
Problem 20.

a) Solve exactly using the iteration method the following recurrence. You may assume that n is a power of 3, ie $n = 3^k$.

$$T(n) = 3T(n/3) + 9n, \text{ where } T(1) = 1.$$

Proof. a) This problem is similar to the recurrence of PS2. Let $n = 3^k$ and thus $k = \log_3 n$. As $T(n) = 3T(n/3) + 9n$, then $T(n/3) = 3T(n/9) + 9n/3$ and similarly for $T(n/9) = \dots$ and so on.

$$\begin{aligned} T(n) &= 3T(n/3) + 9n \\ T(3^k) &= 3T(3^{k-1}) + 9 \cdot 3^k \\ &= 3(3T(3^{k-2}) + 9 \cdot 3^{k-1}) + 9 \cdot 3^k \\ &= 3^2 T(3^{k-2}) + 9 \cdot 3^k + 9 \cdot 3^k \\ &= 3^2 (3T(3^{k-3}) + 9 \cdot 3^{k-2}) + 9 \cdot 3^k + 9 \cdot 3^k \\ &= 3^3 T(3^{k-3}) + 9 \cdot 3^k + 9 \cdot 3^{k-1} + 9 \cdot 3^k \\ &\dots \\ &= 3^k T(3^{k-k}) + 9 \cdot 3^k + \dots + 9 \cdot 3^k + 9 \cdot 3^k \\ &= 3^k T(1) + 9(3^k + \dots + 3^{k-1} + 3^k) \\ &= 3^k \cdot 1 + 9 \cdot 3^k k \\ &= n + 9n \log_3 n \end{aligned}$$

■

Problem Set Problem 21.

Divide-and-Conquer MIN-MAX. Give a divide-and-conquer algorithm that finds both the MINIMUM and the MAXIMUM of n keys $x_1, x_2, \dots, x_n$ by performing at most $3n/2 - 1$ comparisons. You may assume that n is a power of two.

Proof.

```
MinMax(A,1,n)

// MinMax(A,l,r) returns (min,max) , min=min(A[l..r]), max=max(A[l..r]),
// for the subsequence of A between A[l] and A[r] (inclusive) of size r-l+1
// A a sequence of size n, n is a power of two ie n=2**k.
MinMax(A,l,r)
1. if lengthof(A[l..r]) is 2
2.   if (A[l] <= A[r]) return (A[l],A[r])
      else return (A[r],A[l]);
   }
   // Split A into two halves as in mergesort
3. (min1,max1) = MinMax(A,l,(l+r)/2); // Call this half A[l...(l+r)/2] A1
4. (min2,max2) = MinMax(A,(l+r)/2+1,r); // and this one A2
   // and recursively find min,max of two halves.
   // Compare minima of two halves to find minimum
5. if min1 <= min2 then min= min1;
   else min= min2;
   // Compare maxima of two halves to find maximum
6. if max1 <= max2 then max= max2;
   else max= min
```

Split the keys into two equally-size halves. Find the MIN-MAX of each half recursively. The in two comparisons you find the minimum of the two minima and the maximum of the two maxima. They give the global minimum and maximum respectively. The top-level recursive call is MinMax(A,1,n).

Problem Set Continued

The number of **key** comparisons $T(n)$ of **MinMax(A, 1, n)** is derived as follows. Step 2 requires one comparison; A is split as in lines 2-4 of Merge-Sort (page 13). Step 3 requires $T(n/2)$ comparisons and so does Step 4. We call the half of A in step 3, A_1 and that of step 4 A_2 . In steps 5 and 6 we perform a total of two (2) comparisons. Note that the minimum of A is the minimum of the minima of A_1 and A_2 . and the minimum of A is the maximum of the maxima of A_1 and A_2 . We don't need to compare the minima to the maxima. Therefore, the recurrence for the number of comparisons becomes:

$$T(n) = 2T(n/2) + 2 = 2(2T(n/4) + 2) + 2 = 2^2T(n/2^2) + 2^2 + 2 = 2^{\lg n - 1}T(2) + 2^{\lg n - 1} + \dots + 2 = 3n/2 - 2.$$

with $T(2) = 1$ by way of lines 1-2. By the iteration/recursion tree method we obtain that $T(n) = 3n/2 - 2$, as required. [Note that the solution of the recurrence was compressed into one line.]

■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 22.

MIN-MAX revisited. Previously you used a divide and conquer approach to solve this problem. Now you are asked NOT to use such an approach and moreover not to assume that n is a power of two. Solve this problem **again** but for a $\lceil 3n/2 \rceil - 2$ bound.

Proof. Distinguish two cases : odd n and even n respectively, i.e. $n = 2k$ and $n = 2k + 1$. I'll do the easy one. I am skipping details: pair the first two keys , then the next two and so on. As $n = 2k$ we have k pairs. With one comparison per pair we know the min and the max per pair. Group the k mins in A and the k max in B . Find the minimum of A in $k - 1$ comparisons and return it and the max of B in $k - 1$ comparisons and return it. Done (the returned values are the global min and max respectively). Total number of comparisons is $k + (k - 1) + (k - 1) = 3k - 2$. Since $n = 2k$, this is also $3k - 2 = 3n/2 - 2 = \lceil 3n/2 \rceil - 2$ as $3n/2$ is an integer! You do the odd case! ■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

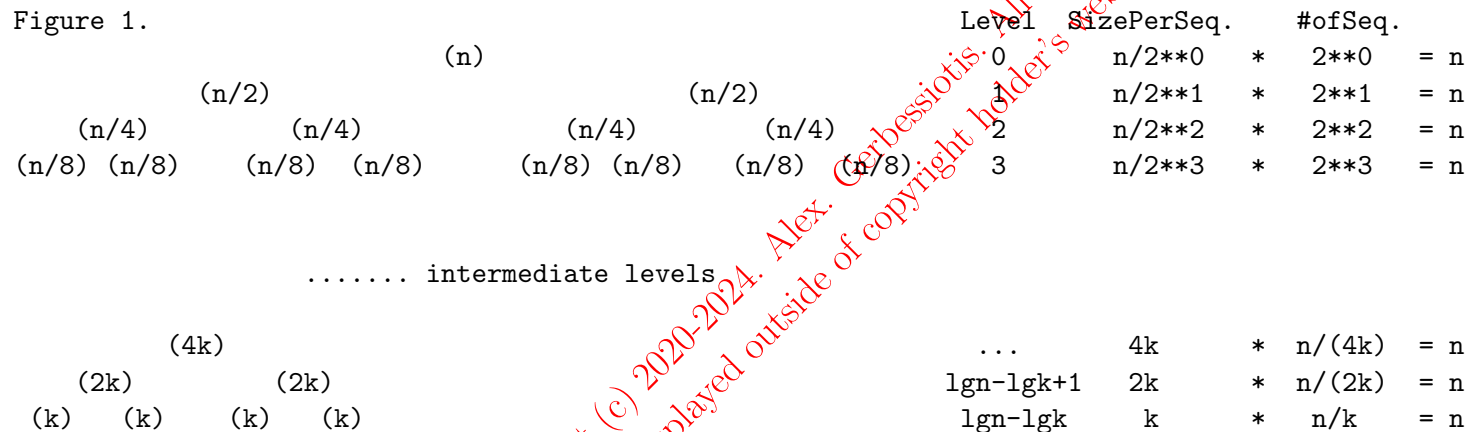
Problem Set Problem 23.

Consider a modification of merge-sort in which out of n keys n/k subsequences of length k are formed and each one is sorted by insertion sort and then all subsequences are merged. The optimal k will be determined after analyzing the worst case performance of the algorithm. In other words, this new algorithm works as follows.

- (i) Form n/k subsequences each of length k . Sort each one with insertion sort. What is the total (worst-case) running time for sorting all k subsequences using insertion sort in asymptotic notation? Explain.
- (ii) The k -long sorted subsequences can then be merged in $\Theta(n \lg(n/k))$ time. Show how.
- (iii) Show that the (worst-case) running time of the whole algorithm (steps (i) and (ii)) is $O(nk + n \lg(n/k))$.
- (iv) What is the (asymptotically) largest value of k for which this algorithm is as good as merge-sort? Explain.

Proof.

Figure 1.



- i. Each sequence of length k can be sorted by insertion sort in $O(k^2)$ worst case time. We have n/k sequences, each one requires $O(k^2)$ time, therefore total time for all sequences is

$$(n/k) \cdot O(k^2) = O(nk).$$

Problem Set Continued

- ii. With reference to Figure 1., at level i of the recursion we have 2^i sequences each one consisting of $n/2^i$ keys. The base case is for $i = \lg(n/k) = \lg n - \lg k$, where n/k subsequences have k keys each. Note that the number of subsequences and the number of elements in an individual subsequence have n as their product. The 2^i subsequences of level i are paired and merged. Each pair is merged in $O(2 \cdot (n/2^i))$ comparisons (forget about the -1) and since the number of pairs (half the number of sets) is $(1/2)2^i$ the *total cost of merging at level i* is $O(n)$. There are $\lg(n/k)$ levels thus the total cost of merging is $O(n \lg(n/k))$.
- iii. The running time of parts (i) and (ii) give a total running time of $O(nk + n \lg(n/k))$.
- iv. We choose k so that $nk = n \lg(n/k)$ i.e. $k = \lg(n/k)$, which gives that $k = \Theta(\lg n)$. Then the performance of the algorithm is $O(n \lg n)$. ■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 24.

A generalized merge-sort algorithm, instead of splitting a sequence of length n into two subsequences of size $n/2$ each, it splits the input sequence into k equal size subsequences (k may not be a constant). In the combining phase, a merging of k sorted sequences needs to be performed. Suppose the recurrence for the running time of this modified merge-sort algorithm is as follows.

$$T(n) = kT(n/k) + n \lg k,$$

with $T(1) = 1$. For simplicity you may also assume that n is a power of k .
Is there a value of k that minimizes $T(n)$? Justify your answer.

Proof. Note that since k may not be a constant **the master method cannot be used** as $a = b = k$ may not be constant. We solve the recurrence with the iteration method.

$$\begin{aligned} T(n) &= kT(n/k) + n \lg k \\ &= k(kT(n/k^2) + (n/k) \lg k) + n \lg k \\ &= k^2T(n/k^2) + n \lg k + n \lg k \\ &= k^i T(n/k^i) + n \lg k + \dots + n \lg k \end{aligned}$$

As $T(1) = 1$ and n is a power of k , we choose i such that $k^i = n$, ie $i = \lg n / \lg k = \log_k n$. Note also, that $k^{1/\lg k} = 2$ below, and there are i terms $n \lg k$ below (equal to the exponent of k^i). Therefore,

$$\begin{aligned} T(n) &= k^i T(n/k^i) + i \cdot n \lg k \\ &= k^{\lg n / \lg k} T(1) + (\lg n / \lg k) n \lg k \\ &= 2^{\lg n} + (n \lg k)(\lg n / \lg k) \\ &= n + n \lg n \end{aligned}$$

Therefore, the answer to the problem is : NO, as $T(n)$ is independent of k , there is no value of k that minimizes $T(n)$. ■

Problem Set Problem 25.

Merge-sort splits a sequence of length n into two subsequences of size $n/2$ each, sorts the two subsequences recursively and then merges the two sorted subsequences. A generalized merge-sort algorithm splits the original sequence into k equal-size subsequences (k may not be a constant and is greater than 2), sorts the k subsequences recursively and then combines the results. In the combining phase, a merging of k sorted sequences needs to be performed (what we call a k -way merging). This takes $O(n \lg k)$ time. Suppose the recurrence for the running time of this modified merge-sort algorithm is as follows.

$$T(n) = kT(n/k) + n \lg k,$$

with $T(1) = 1$. For simplicity you may also assume that n is a power of k . Is there a value of k that minimizes $T(n)$? Justify your answer.

Continuing this Problem, show how to merge k sorted subsequences each of length n/k in time $O(n \lg k)$.

Proof. We solve the recurrence using the iteration method ($T(n) = kT(n/k) + n \lg k$).

$$\begin{aligned} T(n) &= kT\left(\frac{n}{k}\right) + n \lg k \\ &= k\left(kT\left(\frac{n}{k^2}\right) + \frac{n}{k} \lg k\right) + n \lg k \\ &= k^2T\left(\frac{n}{k^2}\right) + n \lg k + n \lg k \\ &= k^2T\left(\frac{n}{k^2}\right) + 2n \lg k \\ &= \dots \\ &= k^iT\left(\frac{n}{k^i}\right) + in \lg k \end{aligned}$$

This is similar to the merge-sort recurrence described in class. The only difference is that the 2 has become a k . The base case is $T(1) = 1$. For the base case to occur $\frac{n}{k^i} = 1$ i.e. $n = k^i$, i.e. $i = \log_k n$ or $i = \lg n / \lg k$. For that value of i we have $n = k^i$ (the first term in the expression above), $T\left(\frac{n}{k^i}\right) = T(1) = 1$ (the second term multiplied by the first)

Problem Set Continued

and $in \lg k = (\lg n / \lg k)n \lg k = n \lg n$ (the third term). Therefore

$$\begin{aligned} T(n) &= k^i T\left(\frac{n}{k^i}\right) + in \lg k \\ &= nT(1) + \left(\frac{\lg n}{\lg k}\right) n \lg k \\ &= n + n \lg n \end{aligned}$$

$T(n)$ is thus independent of k ; it only depends on n . Therefore, there is no value of k that minimizes $T(n)$; the solution is independent of k . Complete the missing part. ■

Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web-page

Problem Set
Problem 26.

What is the order of growth of

$$1 + x + x^2 + \dots + x^n$$

for constant x , and integer n ($n \geq 1$) for the following three cases? Asymptotics are for n going to ∞ . Justify your answer(s) and provide simplest possible answer(s).

1. x is a constant greater than 1,
2. $x = 1$,
3. $0 < x < 1$.

Proof. $\Theta(n)$, $\Theta(1/(1-x)) = \Theta(1)$, $\Theta(x^n)$, in no particular order. Match questions with answers.



Copyright (c) 2020-2024. Alex. Gerbessiotis. All rights reserved.
Not to be displayed outside of copyright holder's web page

Problem Set
Problem 27.

TRUE or FALSE? Do not provide any justification.

- (1) $2^{\lg n} = o(n^2)$.
- (2) $2^n = o(3^n)$.
- (3) $2^n = \omega(3^{n/2})$.
- (4) $n = \Theta(n \lg^* n)$.
- (5) $\lg^* \lg n = \Theta(\lg^* n)$.
- (6) $2^{3 \lg n} = o(n^4)$.
- (7) $3^n = \omega(2^n)$.
- (8) $2^n = \omega(2^{n/2})$.
- (9) $n / \lg^* n = \Theta(n)$.
- (10) $2^{\lg n} = \Theta(n^2)$.
- (11) $n / \lg n = \Theta(\lg(n!))$.
- (12) $3^n = o(4^{n/2})$.
- (13) $n! = \Theta((n+1)!)$.
- (14) $2^{\lg n} = o(n^2)$.
- (15) $n \lg n = \Theta(\lg(n!))$.
- (16) $2^n = o(3^n)$.
- (17) $n! = \Theta((n-1)!)$.

Proof. Practice makes perfect.

■