Solve ALL 6 PROBLEMS IN THE SPACE PROVIDED.

Read the Problems CAREFULLY!

There are 5 (FIVE) pages this page included
Problem 1 (30 points)

TYPE A.

We have \( n! = \omega((n-1)!) \), and \((n-1)! = \omega(n^3) \), and \( \lg n! = \Theta(n \lg n) \) and we know \( 8^{\lg n} = 2^{3 \lg n} = n^3 \), and thus \( n^3 = \omega(n \lg n) \) and the ranking is as follows if we take into consideration the obvious \( \omega \to \Omega \).

\[
n!, \quad (n-1)!, \quad \{ n^3, \quad 8^{\lg n} \}, \quad \{ n \lg n, \quad \lg (n!) \}
\]

TYPES B.

We have \( (n+1)! = \omega(n!) \), \( n! = \omega(n^2) \), \( \lg n! = \Theta(n \lg n) \) and we know \( n^2 = \omega(n \lg n) \), \( 4^{\lg n} = 2^{2 \lg n} = n^2 \) by definition, and thus \( n \lg n = \omega(n/\lg n) \), and \( 2^{2 \lg n} = \omega(n \lg n) \). The ranking is as follows if we take into consideration the obvious \( \omega \to \Omega \).

\[
(n+1)!, \quad n!, \quad \{ n^2, \quad 4^{\lg n} \}, \quad \lg (n!), \quad \lg (n/\lg n),
\]

Problem 2 (30 points)

Type A.

(a) Use the master method \( a = 1 > 0, \ b = 4 > 1, \ f(n) = n, \ n^{\log_a b} = 1 \) to deduce by case three that \( T(n) = \Theta(n) \).

(b) (As promised last week). \( a = b = 2, \ n^{\log_a b} = n, \) and \( f(n) = \lg n \). By case one of the master method \( T(n) = \Theta(n) \).

(c) Since \( T(n) = 3T(n/2) + T(n/4) + n^2 \geq n^2 \) we have that \( T(n) = \Omega(n^2) \). Since \( T(n) = 3T(n/2) + T(n/4) + n^2 \leq 3T(n/2) + n^2 \) either through master method (case three) or by using the substitution method we deduce that \( T(n) = O(n^2) \). Therefore \( T(n) = \Theta(n^2) \).

Let’s show that \( T(n) \leq 3T(n/2) + n^2 \) has \( T(n) = O(n^2) \) as a solution i.e. \( T(n) \leq cn^2 \) for all \( n \geq n_0 \). We choose the base case ourselves \( T(1) = 1 \) which implies \( c \geq 1 \). By the inductive hypothesis \( T(i) \leq ci^2 \) we have for \( i = n/2 \) that \( T(n) \leq 3T(n/2) + n^2 \leq 3c(n/2)^2 + n^2 = 3cn^2/4 + n^2 \). The latter is at most \( cn^2 \) for \( c \geq 4 \). Therefore for \( c = 4 \) and \( n_0 = 1 \) we have shown \( T(n) = O(n^2) \). If we combine this with the \( T(n) = \Omega(n^2) \) the result follows.

Type B.

(a) Use the master method \( a = 1 > 0, \ b = 2 > 1, \ f(n) = n, \ n^{\log_a b} = 1 \) to deduce by case three that \( T(n) = \Theta(n) \).

(b) (As promised last week). \( a = 1, \ b = 2, \ n^{\log_a b} = 1, \) and \( f(n) = \lg n \). By case two of the master method \( T(n) = \Theta(\lg^2 n) \).

(c) Since \( T(n) = 3T(n/2) + T(n/4) + n^3 \geq n^3 \) we have that \( T(n) = \Omega(n^3) \). Since \( T(n) = 3T(n/2) + T(n/4) + n^3 \leq 3T(n/2) + n^3 \) either through master method (case three) or by using the substitution method we deduce that \( T(n) = O(3^2) \). Therefore \( T(n) = \Theta(3^2) \).

Let’s show that \( T(n) \leq 3T(n/2) + n^3 \) has \( T(n) = O(n^3) \) as a solution i.e. \( T(n) \leq cn^3 \) for all \( n \geq n_0 \). We choose the base case ourselves \( T(1) = 1 \) which implies \( c \geq 1 \). By the inductive hypothesis \( T(i) \leq ci^3 \) we have for \( i = n/2 \) that \( T(n) \leq 3T(n/2) + n^3 \leq 3c(n/2)^3 + n^3 = 3cn^3/8 + n^3 \). The latter is at most \( cn^3 \) for \( c \geq 2 \). Therefore for \( c = 2 \) and \( n_0 = 1 \) we have shown \( T(n) = O(n^3) \). If we combine this with the \( T(n) = \Omega(n^3) \) the result follows.
Problem 3 (30 points)

**TYPE A.** \( T(n) = 2T(n/2) + 18n, \quad T(8) = 6. \)

\[
T(n) = 2T(n/2) + 18n \\
= 2^2T(n/2^2) + 2 \cdot 18n \\
= 2^iT(n/2^i) + i \cdot 18n \\
= 2^{\lg n - 3}T(n/2^{\lg n - 3}) + (\lg n - 3) \cdot 18n \\
= (n/8)T(8) + 18n \lg n - 54n \\
= (n/8)6 + 18n \lg n - 54n \\
= 0.75n + 18n \lg n - 54n \\
= 18n \lg n - 53.25n
\]

We used in the fourth equality the facts that \( T(8) = 6 \) and \( 2^{\lg n - 3} = n/8. \)

**TYPE B.** \( T(n) = 2T(n/2) + 6n, \quad T(8) = 18. \)

\[
T(n) = 2T(n/2) + 6n \\
= 2^2T(n/2^2) + 2 \cdot 6n \\
= 2^iT(n/2^i) + i \cdot 6n \\
= 2^{\lg n - 3}T(n/2^{\lg n - 3}) + (\lg n - 3) \cdot 6n \\
= (n/8)T(8) + 6n \lg n - 18n \\
= (n/8)18 + 6n \lg n - 18n \\
= 2.25n + 6n \lg n - 18n \\
= 6n \lg n - 15.75n
\]

We used in the fourth equality the facts that \( T(8) = 6 \) and \( 2^{\lg n - 3} = n/8. \)
**Problem 4 (30 points)**

For the running time of MergeSort below sometimes we use $\Omega(n \lg n)$ to stress that this is a lower-bound (i.e. it takes at least that much) and sometime we use $O(n \lg n)$ to stress the upper bound. In the former case we compare performance to a linear algorithm and in the latter to a quadratic algorithm.

**TYPE A.**

1. Insertion-Sort sorts in-place. **TRUE.**
2. MergeSort sorts in-place. **FALSE.**
3. On the input sequence $(1, 2, \ldots, n)$, Insertion-Sort is asymptotically faster than MergeSort. **TRUE**, InsertionSort is $\Theta(n)$ on the input, MergeSort still takes $\Omega(n \lg n)$. 
4. On the input sequence $(n, n - 1, \ldots, 1)$, InsertionSort is asymptotically faster than MergeSort. **FALSE**, InsertionSort is $\Theta(n^2)$ but MergeSort still takes $O(n \lg n)$. 
5. On the input sequence $(n, n - 1, \ldots, 1)$, InsertionSort has running time that is $O(n^3)$. **TRUE**, InsertionSort is $\Theta(n^2)$ on the input, and $\Theta(n^2) = O(n^3)$. 
6. On the input sequence $(1, 2, \ldots, n)$, InsertionSort has running time that is $O(n^2)$. **TRUE** InsertionSort is $\Theta(n)$ on the input, which is $O(n^2)$.

**TYPE B.**

1. Insertion-Sort is stable. **TRUE.**
2. MergeSort is stable. **TRUE.**
3. On the input sequence $(1, 2, \ldots, n)$, Insertion-Sort is asymptotically faster than Selection-sort. **TRUE**, InsertionSort is $\Theta(n)$ but SelectionSort is still $\Theta(n^2)$. 
4. On the input sequence $(n, n - 1, \ldots, 1)$, InsertionSort is asymptotically slower than MergeSort. **TRUE** since InsertionSort’s $\Theta(n^2)$ is slower than MergeSort’s $\Omega(n \lg n)$ time. 
5. On the input sequence $(n, n - 1, \ldots, 1)$, MergeSort has running time that is $O(n^2)$. **TRUE** since MergeSort is $O(n \lg n)$ which is $O(n^2)$.
6. On the input sequence $(1, 2, \ldots, n)$, MergeSort has running time that is $\Omega(n)$. **TRUE** since MergeSort is $\Omega(n \lg n)$ which is $\Omega(n)$.

**Problem 5 (30 points)**

**TYPE A.**

(a) The first 3 `Enqueue` have a cost of 3. The following 2 `Dequeue` have a cost of $2 \cdot 3 = 6$ to move 3 elements from $A$ to $B$ and 2 to actually remove, i.e. a total of 8. As of this point $A$ is empty and $B$ has one element and the total cost so far is 11. The next 3 `Enqueue` have a cost of 3. The final 2 `Dequeue` operations have the following cost: the penultimate `Dequeue` just removes the only element of $B$ at cost 1, and the ultimate `Dequeue` moves the 3 elements from $A$ to $B$ at a cost of 6 and then removes 1 element from $B$ at a cost of additional 1. Total cost for these `Dequeue` operations is $1 + 6 + 1 = 8$. Total cost for the second set of operations is 11.

Total cost is 22 and at the end $A$ is empty and $B$ has two elements.

(b) Suppose we have $n$ `Enqueue` followed by $n$ `Dequeue` operations. The first $n$ operations cost 1 each. The first `Dequeue` (the $n$-th overall operation) moves the $n$ elements from $A$ to $B$ and then removes one element. Its total cost is $2n + 1$.

Every element is inserted in $A$ at a cost of 1 and overall cost (for all $n$ such `Enqueue` operations) is thus $n$. Every element is inserted initially in $A$ but deleted from $B$. The cost assigned to a single element is 2 when it gets moved from $A$ to $B$ and 1 more when it gets popped off $B$. Therefore the per element cost of a `Dequeue` is 3. We have $n$ `Dequeue` operations whose overall cost is thus $3n$. Therefore the overall cost of all 2$n$ operations is $n + 3n = 4n$, no matter what the order of the operations is.

If the $n$-1-st operation costs $2n + 1$ and the total cost is $4n$ no other operation can cost more than $4n - (2n + 1) = 2n - 1$. Each of the remaining $2n - 1$ operations costs 1, so they collectively account for this $2n - 1$. The results follows.

**TYPE B.**

(a) The first 4 `Enqueue` have a cost of 4. The following 3 `Dequeue` have a cost of $2 \cdot 4 = 8$ to move 4 elements from $A$ to $B$ and 3 to actually remove, i.e. a total of 11. As of this point $A$ is empty and $B$ has one element and the total cost so far is 15. The next 2 `Enqueue` have a cost of 2. The final 2 `Dequeue` operations have the following cost: the penultimate `Dequeue` just removes the only element of $B$ at cost 1, and the ultimate `Dequeue` moves the 2 elements from $A$ to $B$ at a cost of 4 and then removes 1 element from $B$ at a cost of additional 1. Total cost for these `Dequeue` operations is $1 + 4 + 1 = 6$.

Total cost is 23 and at the end $A$ is empty and $B$ has one element.

(b) Same answer as in case (b) of the **TYPE A** exam.
Problem 6 (17 points)

Type A (jingly-sorted sequences).

"Sort the \(2n+1\) keys" to separate the \(n+1\) smallest from the \(n\) largest. Call these two sequences \(S\) (for smallest) and \(L\) (for largest) respectively.

Then interlace the two chains together picking one key from \(S\) and then one from \(L\). The resulting sequence is jingly sorted because for an arbitrary key \(A[2i+1]\) since \(A[2i+1]\) comes from \(L\), and \(A[2i]\) and \(A[2i+2]\) come from \(S\), and all elements of \(L\) are larger than those from \(S\) we deduce the fact that odd-indexed keys are well-placed in the output. We can repeat the same argument similarly for even-indexed keys. Obviously the running time is determined by the sorting operation to form \(S\) and \(L\) since interlacing takes linear time. Therefore the running time of jingle-sorting is \(O(n \log n)\).

The algorithm can be performed in place as follows.

```plaintext
JinglySort (A,n) // n is the problem size but the input consists of
// 2n+1 keys i.e. input size is 2n+1
1. MergeSort(A,2n+1);    // O(nlogn) time
2. for(i=1;i<=n;i+=2)    //
3. swap(i,2n-i+1);        //swap A[i] and A[2n-i+1]    // Linear time in lines 2-3
```

Note. Step 1 is generally overkill we donot need to have the elements in \(L\) or \(S\) sorted, just separated. If we perform (and we will show how this can be done so in class) this step in linear time, jingle-sorting requires linear time as well.

TYPE B. ("close-pair")

Find \(m,M\) in at most \((n-1) + (n-1)\) comparisons (we know from PS1 how we can do it faster in about \(3n/2\) comparisons). No matter how we do it, this step is \(\Theta(n)\).

Split the interval \([m,M]\) into \(n-1\) sub-intervals of width \(d = (M-m)/(n-1)\), i.e.

\[
[m,m+d], \\
(m+d,m+2d], \\
(m+2d,m+3d], \\
\ldots, \\
(M-d,M].
\]

Attach a linked list to each interval and call the lists \(L_0, \ldots, L_{n-2}\).

Go through the elements of \(A\) one by one. In linear time decide which linked list \(A[i]\) belongs to by finding the index of the linked list for \(A[i]\) which is \(\lfloor (A[i] - m)/d \rfloor\).

There are \(n\) keys and \(n-1\) linked lists. By the pigeonhole principle there will be a linked list with at least two keys (since otherwise \(n-1\) linked lists would have a maximum of one key, which is impossible since we placed \(n\) keys into them). If there are more lists, pick one arbitrarily. Pick two of the keys from such a list that contains two or more keys and call them \(x,y\) (if there more than two such keys, pick two arbitrarily as well). The two keys \(x,y\) are at most \(d\) apart because they belong to the same \(d\)-wide interval. The result follows.

An issue that might cause complications if not properly implemented is insertion into a linked list. We also insert at the head of such list so that the running time is \(O(1)\) in the worst-case. Running time of the algorithm is \(\Theta(n)\). The splitting into \(n-1\) intervals, the assignment (initialization) of \(n-1\) linked lists, and the scanning of the \(n\) keys to assign them to the linked lists all take \(\Theta(n)\) steps. Going through the linked lists to find one of length greater than 1 also takes \(\Theta(n)\) time. The result follows.

This is the end of the exam.