

Name:SOLUTIONS.....

ID Number:

Exam Number:

.....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... Total:

SOLVE ALL 6 PROBLEMS IN THE SPACE PROVIDED.

Read the Problems CAREFULLY!

THERE ARE 4 (FOUR) PAGES THIS PAGE INCLUDED

List of useful formulae

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4} \quad n! \approx \left(\frac{n}{e}\right)^n, \quad a^{\log_b n} = n^{\log_b a},$$

For $x \neq 1$, we have that

$$\sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1}, \quad \sum_{i=0}^{n-1} ix^i = \frac{(n-1)x^{n+1} - nx^n + x}{(1-x)^2},$$

B1. $f(n) = \Theta(g(n))$ iff $\exists$ positive constants $c_1, c_2, n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$.

B2. $f(n) = \Omega(g(n))$ iff $\exists$ positive constants $c_1, n_0 : 0 \leq c_1 g(n) \leq f(n) \quad \forall n \geq n_0$.

B3. $f(n) = O(g(n))$ iff $\exists$ positive constants $c_2, n_0 : 0 \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$.

Master Method. $T(n) = aT(n/b) + f(n)$, such that $a > 0, b > 1, f(n) > 0$.

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 $f(n) = \Theta(n^{\lg_b a} \lg^k n)$, then $T(n) = \Theta(n^{\lg_b a} \lg^{k+1} n)$, where $k \geq 0$ is a non-negative constant.

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 $c < 1$ and for large n , then $T(n) = \Theta(f(n))$.

Problem 1. (40 POINTS)

Properties of sorting algorithms (i.e. questions 1 and 2) are described on page 19, Subject 1. Problem 8 was PS1.Problem7.part.a with the 3 of $3n^2$ missing. Problem 7 was PS1.Problem 9.part.a

- (1) Insertion-Sort sorts in-place. **TRUE**
- (2) MergeSort sorts in-place. **FALSE**
- (3) On the input sequence $\langle 1, 2, \dots, n \rangle$, Insertion-Sort is asymptotically faster than MergeSort. **TRUE**, the former is $\Theta(n)$ and the latter is $\Omega(n \lg n)$.
- (4) On the input sequence $\langle n, n-1, \dots, 1 \rangle$, InsertionSort is asymptotically faster than MergeSort. **FALSE** the former is $\Theta(n^2)$ and the latter is $O(n \lg n)$.
- (5) On the input sequence $\langle n, n-1, \dots, 1 \rangle$, InsertionSort has running time that is $\Omega(n)$. **TRUE** since the running time is $\Theta(n^2)$ which is $\Omega(n)$.
- (6) On the input sequence $\langle 1, 2, \dots, n \rangle$, InsertionSort has running time that is $O(n^2)$. **TRUE** since the $\Theta(n)$ running time is also $O(n^2)$.
- (7) The asymptotic solution of $T(n) = T(n/2) + \lg n$ is $T(n) = \Theta(n)$. **FALSE**. Case 2 of the master method shows that it is $T(n) = \Theta(\lg^2 n)$.
- (8) The asymptotic solution of $T(n) = 3T(n/2) + T(2n/4) + n^2$ is $T(n) = \Theta(n^2)$ **FALSE** Case 2 of the master method show that it is $T(n) = \Theta(n^2 \lg n)$. Note that $3T(n/2) + T(2n/4) = 4T(n/2)$.

Problem 2. (30 POINTS)

This is a shortened version of PS1.Problem 6.part.b with several functions replaced.

$$n!, \quad 2^n, \quad 2^{3 \lg n}, \quad n^2 \lg n, \quad 3^{\lg n}, \quad \lg(n!)$$

We have that $n! = \omega(2^n)$. Also $2^n = \omega(n^k)$ for any constant $k > 0$. Since $2^{3 \lg n} = (2^{\lg n})^3 = n^3$, and $n^3 = \omega(n^2 \lg n)$, and $n^2 \lg n = 2^{2 \lg n} \lg n = 4^{\lg n} \lg n = \omega(3^{\lg n})$ the result follows also taking into consideration that $3^{\lg n} = \omega(n \lg n)$, and $\lg(n!) = \Theta(n \lg n)$.

Problem 3. (17 POINTS)

This is PS1.Problem 6.part.a and also similar to a problem done in class.

- (a) O first. $n^3 - 2n^2 + 1 \leq n^3 + 1 \leq 2n^3$. Therefore for $c_2 = 2$ and $n_0 = 1$ we have $n^3 - 2n^2 + 1 = O(n^3)$.
- (b) Ω then. $n^3/2 \leq n^3 - 2n^2 \leq n^3 - 2n^2 + 1$ as long as $n \geq 4$. Therefore for $c_1 = 1/2$ and $n_0 = 4$ we have that $n^3 - 2n^2 + 1 = \Omega(n^3)$.

Combining parts (a) and (b) we have that $c_1 = 1/2$ and $c_2 = 2$ and $n_0 = 4$ are the constants to claim that $n^3 - 2n^2 + 1 = \Theta(n^3)$.

Problem 4. (30 POINTS)

The algorithm determines by divide-and-conquer whether all the keys of array A are the same or not.

Split A into two halves. If $A[0]$ and $A[n-1]$ are not equal output FALSE since we know that all the keys are not equal to the same value since $A[0]$ and $A[n-1]$ differ.

Otherwise recur on the left and the right half. Boundary case is the case where a subarray has only one key; return TRUE by default.

The running time recurrence for the running time $T(n)$ is thus $T(n) = 2T(n/2) + 1$. Case 1 of the master method gives $T(n) = \Theta(n)$.

Problem 5. (30 POINTS)

$$T(n) = 2T(n/2) + 6n, \quad T(4) = 12.$$

$$\begin{aligned}
T(n) &= 2T(n/2) + 6n \\
&= 2^2T(n/2^2) + 2 \cdot 6n \\
&= 2^i T(n/2^i) + i \cdot 6n \\
&= 2^{\lg n - 2} T(n/2^{\lg n - 2}) + (\lg n - 2) \cdot 6n \\
&= (n/4)T(4) + 6n \lg n - 12n \\
&= (n/4)12 + 6n \lg n - 12n \\
&= 3n + 6n \lg n - 12n \\
&= 6n \lg n - 9n
\end{aligned}$$

We used in the fourth equality the facts that $T(4) = 12$ and $2^{\lg n - 2} = n/4$.

Problem 6. (20 POINTS)

Scan A left to right one element at a time. At the same time maintain a sorted X of at most k elements as required. For every element $A[i]$ of A do a binary search into X formed so far. Since X can have at most k elements this takes $O(\lg k)$ time. If $A[i]$ is already in X then nothing is being done; such a key has been dealt with before. If however this is the first $A[i]$ encountered, then and only then do we insert $A[i]$ in the sorted X ; this involves making room for $A[i]$ in X by moving all keys greater than $A[i]$ to the right one position. In the worst case all of X might have to move to the right to make space for the new key, an operation that can take j steps for the j -th insertion into X . We then proceed to the next key $A[i + 1]$.

Total time spent is as follows:

(a) Binary search requires $O(\lg k)$ per input key and since A has n keys and X has at most k keys this is $O(n \lg k)$.

(b) Insertion requires $O(j)$ for the j -th inserted key. Only unique keys are inserted the first time they are encountered. Therefore, we have k insertions for a total of $\sum_{j=1}^k O(j) = O(k^2)$.

(c) The running time altogether for (a), (b) is thus $O(n \lg k + k^2) = O(n \lg k)$ since $k \leq \sqrt{n}$ and thus $k^2 = O(n)$.

The pseudocode corresponding to this description looks like.

```

Keys(A,n,k)
0. allocate space for X[k];
1. j=0 ; // number of keys in X so far
2. for(i=0;i<n;i++) {
3.   t = BinSearch(X,j,A[i]); // Do a BinSearch of A[i] in X ; t is position of A[i]
                               // in X, or position where A[i] can be inserted in X
4.   if X[t] != A[i] {
5.     Insert(X,j,A[i],t); // Insert A[i] at X[t] i.e. X[t]...X[j-1] move to X[t+1] ...X[j]
6.     j++;
7.   } // Note: the case where j=0 and A[i] is the first inserted key is handled within
       // Insert. No moving to the right is required since A[i] is the first key into X.
8. }
9. return(X);

```