Name:________________________________ ID:________________________________

# CSE 246 Analysis of Algorithms

## Spring 2012 Midterm Exam

12.04.2011 Thursday, Duration: 90 minutes

| Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | B1 | SUM |
|---|---|---|---|---|---|---|---|
| /12 | /10 | /20 | /20 | /20 | /18 | /6 | /100 |

**Q-1.** (12 pts) Solve the following recurrences. Express your answer using $\Theta(\cdot)$ notation.
(a) (6 pts) T(n)=2T(n-2), T(0)=1, T(1)=1.

By backward substitution,

$T(n)=2^{n/2}$ if n is even
$T(n)=2^{n/2}2^{-1/2}$ if n is odd.
$T(n) \in \Theta(2^{n/2}) = \Theta((\sqrt{2})^n)$

(b) (6 pts) $T(n)=4T(\lceil n/2 \rceil)+n$, T(1)=1.

By backward substitution or Master's theorem

$T(n) \in \Theta(n^2)$

**Q-2.** (10 pts) Design a decrease-by-half algorithm for computing $\lfloor \log_2 n \rfloor$ and determine its time efficiency.

Algorithm LogFloor (n)
//Input: A positive integer n
//Output: Returns $\lfloor \log_2 n \rfloor$
if n = 1 return 0
else return LogFloor ($\lfloor n/2 \rfloor$) + 1

The recurrence relation for the number of additions is
$A(n) = A\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$ for $n > 1$, $A(1) = 0$
Its solution is $A(n) = \lfloor \log_2 n \rfloor \in \Theta(\log n)$

Name:______________________________ ID:______________________________

**Q-3. (20 pts) Consider the following variant of MergeSort: instead of splitting the list into two halves, we split it into three thirds. Then we recursively sort each third and merge them. This is called three-way MergeSort.**

**a. (6 pts) Write a pseudocode for three-way MergeSort. You may assume that you are given an algorithm, Merge(B,C,A) which merges two sorted arrays (B,C) into one sorted array (A).**

**Mergesort3(A[0..n-1]):**
**if $n \leq 1$, then return (A[0..n-1]).**
**Let $k = \lceil n/3 \rceil$ and $m = \lceil 2n/3 \rceil$.**
**Return Merge3(Mergesort3(A[0..k-1]), Mergesort3(A[k..m-1]), Mergesort3(A[m..n-1]),A[0..n-1]).**

**Merge3(B,C,D,X):**
**Merge(B,C,E); Merge(E,D,X).**

**b. (4 pts) What is the total number of key comparisons performed in the worst case, while merging three sorted lists, each of length n/3, to one sorted list? Also express your answer using O(·) notation.**

**$n/3+n/3 - 1 = 2n/3 - 1$ for Merge(B,C,E);**

**$2n/3 + n/3 - 1 = n - 1$ for Merge(E,D,X);**

**Total: $5n/3 - 2 \in O(n)$**

**c. (3 pts) Let T(n) denote the worst-case running time of three-way MergeSort on an array of size n. Write a recurrence relation for T(n).**

**$T(n)=3T(n/3)+O(n)$**

**d. (3 pts) Solve the recurrence relation in part (c). Express your answer using O(·) notation.**

**By Master theorem, $T(n)=O(n\log n)$**

**e. (2 pts) Is the three-way MergeSort asymptotically faster than insertion sort? (<u>Yes</u> or No)**

**f. (2 pts) Is the three-way MergeSort asymptotically faster than ordinary MergeSort? (Yes or <u>No</u>)**

Name:______________________________ ID:______________________________

Q-4 (20 pts) We have two input arrays, an array $\boldsymbol{A}$ with $\boldsymbol{m}$ elements, and an array $\boldsymbol{B}$ with $\boldsymbol{n}$ elements, where $\boldsymbol{n} \geq \boldsymbol{m}$. There may be duplicate elements. We want to decide if every element of $\boldsymbol{B}$ is an element of $\boldsymbol{A}$.

(a) (6 pts) Describe a brute-force algorithm. What is the worst-case time complexity?

We compare each element of B with each element of A. If there is no match for any element of B, algorithm stops returning false. Worst-case time complexity is O(nm).

(b) (14 pts) Describe an algorithm to solve this problem in $\boldsymbol{O}(\boldsymbol{n}\log\boldsymbol{m})$ worst case time. (Hint: You may apply instance simplification.)

First we sort A by MergeSort (in O(mlog m) time). Then for each element of B we do a binary search in the sorted list of A (in O(n log m) time).

The total worst-case running time is O((m + n) log m) = O(n log m).

Q-5 (20 pts) We have an input array $\boldsymbol{A}$ with n (n > 1) elements.

(a) (10 pts) Describe a O(n) worst-case time algorithm to find two elements $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{A}$ such that $|\boldsymbol{x} - \boldsymbol{y}| \geq |\boldsymbol{u} - \boldsymbol{v}|$ for all $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{A}$.

For this, we have to find minimum and maximum of the list. We store a temporary variable (max or min, initially -∞ or +∞). We compare it with the elements one by one, and update the value of the temporary variable after each comparison. This algorithm makes n-1 comparisons to find each. 2n-2 ∈ O(n)

There is also a divide and conquer algorithm that finds min and max simultaneously using at most 3n/2 comparisons (As we described in the class).

(b) (10 pts) Describe a O(n log n) worst-case time algorithm to find two elements $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{A}$ such that $|\boldsymbol{x} - \boldsymbol{y}| \leq |\boldsymbol{u} - \boldsymbol{v}|$ for all $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{A}$.

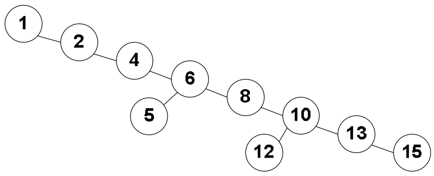
For this, firstly we sort the numbers using Mergesort (O(nlogn)). Then x and y must be consecutive elements in the sorted order. We go through the sorted list and find the smallest difference between two neighboring elements (this is O(n)).

O(nlogn)+O(n)=O(nlogn)

Name:_______________________________ ID:_______________________________

Q-6 (18 pts) Consider the following almost sorted list:
L = 1, 2, 4, 6, 5, 8, 10, 13, 12, 15

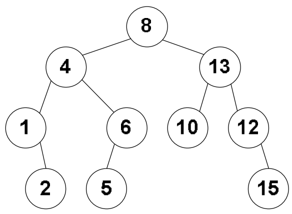
(a) (5 pts) Construct a Binary Search Tree by inserting the elements of L from left to right, one by one. In the worst-case, how many comparisons is needed for searching a key in the constructed tree.




8 comparisons.

(b) (8 pts) In order to decrease the worst-case complexity of searching a key, describe an alternative algorithm for BST construction by changing the insertion sequence of the elements. In the new BST, how many comparisons is needed for searching a key (in the worst-case)?

First insert the medium element. Extract that element from the list, then divide the remaining list into two sub-lists, and insert the medium element of each sub-lists. And so on, insert all the elements recursively.




4 comparisons.

(c) (5 pts) Describe another alternative way to decrease worst-case complexity of searching a key, by transforming ordinary BST to another data structure.

We may transform unbalanced search tree to a balanced one, such as using AVL tree or red-black tree structures.

B-1 (Bonus Question - 6 pts, no partial credit): Solve the following recurrence relation:

$T(n)=2T(\sqrt[3]{n})+1$, $T(3)=1$.

Put $n=3^k$. Accordingly, $T(3^k)=2T(3^{k/3})+1$, $T(3^1)=1$.
Let $G(k)$ denote $T(3^k)$. Accordingly, $G(k)=2G(k/3)+1$, $G(1)=1$.

Using any technique (Master theorem, backward subs., etc), $G(k)=O(2^{\log_3 k})$, from which it follows that $T(n)=O(2^{\log_3 \log_3 n})$.