

Name _____ Student No. _____

Tutor: Travis _____ Kleoni _____ Babak _____ Mohammad _____ Spyros _____

NO AIDS ALLOWED

Answer ALL questions on test paper. Use backs of sheets for scratch work.

Total Marks: 50

- (1) (a) (8 points) Solve the following recurrence exactly. Show your work.

$$T(n) = \frac{1}{n} \sum_{i=1}^{n-1} (T(i) + 1),$$

where $T(1) = 0$.**Solution:** We can use the same trick as with the recurrence from the average-case complexity of quicksort:

$$nT(n) - (n-1)T(n-1) = \sum_{i=1}^{n-1} (T(i) + 1) - \sum_{i=1}^{n-2} (T(i) + 1) \quad (1)$$

$$= T(n-1) + 1 \quad (2)$$

Therefore,

$$T(n) = T(n-1) + 1/n.$$

This recurrence we can solve by substitution to get:

$$T(n) = 1/2 + 1/3 + \dots + 1/n.$$

- (b) (2 points) Give the simplest
- Θ
- expression for this quantity.

Solution:

$$\Theta(\log n)$$

(2) Definitions:

(a) (5 points) Define a *dictionary abstract data type (ADT)*.

Solution: A dictionary represents a set S of elements with key-values that supports the following operations:

- * INSERT(S, x) for x an element.
- * DELETE(S, x)
- * SEARCH(S, k) for k a key-value.
- * ISEMPY(S)

(b) (5 points) Let A be a randomized algorithm that takes its inputs from the set S . A makes a random choice from the sample space P . Let $t(x, p)$ be the running time of A on input $x \in S$ using random choice $p \in P$. Write down the expression that defines the *expected worst-case running time* of A .

Solution:

$$\max_{x \in S} E_{p \in P}(t(x, p)).$$

- (3) Let A be a (fixed) sorted array of $n = 2^m - 1$ distinct integers. Consider the following algorithm for searching for integer k between positions a and b (including positions a and b themselves) of the array:

```

SEARCH(A, k, a, b)

  if ( a = b AND A[a] != k ) then          /* Base cases */
    return -1
  else if ( a = b AND A[a] = k ) then
    return a

  c := (a-b)/2                               /* SHOULD BE: c := (b-a)/2 */
  if ( A[a+c] = k ) then
    return a+c
  else if ( A[a+c] < k ) then               /* SHOULD BE: A[a+c] > k */
    return SEARCH(A, k, a, a+c-1)
  else if ( A[a+c] > k ) then               /* SHOULD BE: A[a+c] < k */
    return SEARCH(A, k, a+c+1, b)
END

```

We will measure the complexity of this algorithm in terms of the number of calls to SEARCH that get executed.

- (a) (3 points) Write a recurrence for the worst-case running time of SEARCH($A, k, 1, n$), assuming that k is in the array. What is the solution to the recurrence?

Solution:

$$T(n) = T((n-1)/2) + 1,$$

where $T(1) = 0$. Solving by substitution, we get:

$$T(n) = m.$$

- (c) (7 points) Assuming k is equally likely to be any integer in the array, what is the average-case running time of SEARCH (you can leave your answer in Σ notation).

Solution: Let S be the set of key-values in array A , and let $t(k)$ be the number of calls it takes to find key value k in A . Then, the average-case running time is, by definition:

$$\sum_{k \in S} \Pr(k) t(k).$$

Let A_i be the set of key-values that are found in exactly i calls to SEARCH. There are 2^{i-1} such values. Therefore, the average running time is equal to:

$$\sum_{i=1}^m \Pr(A_i) t(A_i),$$

where $t(A_i)$ is the number of calls it takes to find a key from A_i . This is:

$$\sum_{i=1}^m (2^{i-1}/n)i.$$

(4) Augment the binary search tree (BST) data structure to support the query $\text{DEPTH}(R, x)$, which, given a node x in the BST rooted at R , returns the depth of that node in time $O(1)$. The depth of the root is 0, the depth of the root's children is 1, etc.

(a) (2 points) Describe the extra information you need to store with the tree. How will it be stored?

Solution: For each node x , store its depth in a field $x.depth$.

(b) (2 points) Write the pseudocode for the operation $\text{DEPTH}(R)$ (**NB: should be $\text{DEPTH}(R, x)$**) given the extra information from (a).

Solution:

```
DEPTH( $R, x$ )  
    return  $x.depth$ 
```

- (c) (6 points) Describe how to modify INSERT and DELETE to maintain the information from (a) without affecting their running times.

Solution:

“Without affecting their running times” means that the new INSERT and DELETE must run in time $O(\text{height of tree } R)$, not $O(n)$.

INSERT(R, x): set $x.depth$ to $parent(x).depth + 1$.

```
DELETE( $R, x$ )
  if ( left( $x$ ) != NIL ) then          /* x has a left child */
    key( $x$ ) := key(predecessor( $x$ ))
    DELETE( $R$ , predecessor( $x$ ))
  else if ( right( $x$ ) != NIL ) then /* x has no left; x has a right child */
    key( $x$ ) := key(successor( $x$ ))
    DELETE( $R$ , successor( $x$ ))
  else remove  $x$                       /* x has no children */
END
```

DELETE just traces a path from x to a leaf, performing constant-time operations along the way. Hence it runs in time $O(\text{height of tree } R)$.

- (d) (10 points) Can you augment a Red-Black tree in the same way without affecting the running times of any of its operations? Why or why not?

Solution: No. When rotating right about $x-y$ in the fix-up process, the depth of every node in the left subtree of x and the right subtree of y must be updated. This could take much longer than $O(\text{height of tree } R) = O(\log n)$. The same problem arises when rotating left.