Analysis of Algorithms

Homework 2

PROBLEM 1

Argue the correctness of HEAP-INCREASE-KEY using the following loop invariant: at the start of each iteration of the **while** loop of lines 4 – 6, the subarray $A[1 \ldots A.\textit{heap-size}]$ satisfies the max-heap property, except that there may be one violation: $A[i]$ may be larger than $A[\text{PARENT}(i)]$.

The pseudocode for HEAP-INCREASE-KEY is reproduced for your convenience:

HEAP-INCREASE-KEY(A, i, key):

1 **if** $key < A[i]$

2   **error** "new key is smaller than current key"

3 $A[i] = key$

4 **while** $i > 1$ and $A[\text{PARENT}(i)] < A[i]$

5   exchange $A[i]$ with $A[\text{PARENT}(i)]$

6     $i = \text{PARENT}(i)$

You may assume that the subarray $A[1 \ldots A.\textit{heap-size}]$ satisfies the max-heap property at the time HEAP-INCREASE-KEY is called.

PROBLEM 2

(a) Give an $O(n \lg k)$-time algorithm to merge k sorted lists into one sorted list, when n is the total number of elements in all the input lists.

(b) Write a k-way Merge Sort algorithm to using the procedure in **(a)**. What is the running time as a function of n and k? What is the best value of k to use?

PROBLEM 3

In the deterministic selection algorithm, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into:

(a) groups of 7?

(b) groups of 3?

For each case, provide the worst-case running time.

PROBLEM 4

Does the following pseudocode produce a uniform random permutation?

(a) SHUFFLE(A):

```
for i = 1 to n
    swap A[i] with A[RANDOM(1, n)]
```

(b) SHUFFLE(A):

```
New array B[1 ... n]
offset = RANDOM(1, n)
for i = 1 to n
    dest = i + offset
    if dest > n
        dest = dest − n
    B[dest] = A[i]
return B
```

For **(b)** show that each element $A[i]$ has a $1/n$ probability of getting assigned to any particular position in B.

PROBLEM 5

For n distinct elements $x_1, \dots, x_n$ with positive weights $w_1, \dots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the weighted median is the element x_k satisfying the following:

$$\sum_{x_i < x_k} w_i < \frac{1}{2} \text{ and } \sum_{x_i > x_k} w_i \leq \frac{1}{2}$$

For example, if the elements are $\langle 0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2 \rangle$ and each element equals its weight, the median is 0.1, but the weighted median is 0.2.

(a) Argue that the median of $x_1, \dots, x_n$ is the weighted median of the x_i with weights $w_i = 1/n$ for $i = 1, 2, \dots n$.

(b) Show how to compute the weighted median of n elements in $O(n \lg n)$ worst-case time using sorting.

(c) Show how to compute the weighted median in $\Theta(n)$ worst-case time.

PROBLEM 6

In this problem, we use indicator random variables to analyze the RANDOMIZED-SELECT procedure. As in the quicksort analysis, we assume that all elements are distinct, and we rename the elements of the input array A as $z_1, \dots z_n$ where z_i is the i-th smallest element. (In other words, the call RANDOMIZED-SELECT$(A, 1, n, k)$ returns z_k.) For $1 \le i < j \le n$, the indicator random variable

$$X_{ijk} = \mathbf{1}\{z_i \text{ is compared with } z_j \text{ sometime during the execution of the algorithm to find } z_k\}$$

(a) Give an exact expression for $E[X_{ijk}]$. *Hint:* It depends on the values of i, j, k.

(b) Let X_k denote the total number of comparisons between elements of array A when finding z_k. Show that

$$E[X_k] \le 2\left(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + \sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1} + \sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1}\right)$$

(c) Show that $E[X_k] \le 4n$.

(d) Conclude that, assuming all elements of A are distinct, RANDOMIZED-SELECT runs in expected time $O(n)$.

For your reference, the pseudocode for RANDOMIZED-SELECT and RANDOMIZED-PARTITION:

```
RANDOMIZED-SELECT(A, p, r, i):
    if p == r
        return A[p]
    q = RANDOMIZED-PARTITION(A, p, r)
    k = q - p + 1
    if i == k                 //the pivot value is the answer
        return A[q]
    else if i < k
        return RANDOMIZED-SELECT(A, p, q - 1, i)
    else    return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

```
RANDOMIZED-PARTITION(A, p, r):
    i = RANDOM(p, r)
    exchange A[r] with A[i]
    return PARTITION(A, p, r)
```

```
PARTITION(A, p, r):
    x = A[r]
    i = p - 1
    for j = p to r - 1
        if A[j] < x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```