

CSCI 303 Homework 3

Problem 1 (9-1):

Given a set A of n numbers, we wish to find the k largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running time of the algorithms in terms of n and k .

- a. Sort the numbers, and list the k largest.
- b. Build a max-priority queue from the numbers, and call EXTRACT-MAX k times.
- c. Use an order-statistic algorithm to find the i th largest number, partition around that number, and sort the k largest numbers.

Solution 1:

a.

```
MERGE-SORT( $A, 1, n$ )  
return  $A[n - k \dots k]$ 
```

This algorithm takes only as long as it takes to MERGE-SORT a list of n numbers, so its running time is $\Theta(n \lg n)$.

b.

```
BUILD-MAX-HEAP( $A$ )  
for  $i \leftarrow 1$  to  $k$   
     $B[i] \leftarrow$  HEAP-EXTRACT-MAX( $A$ )  
return  $B$ 
```

This algorithm first calls BUILD-MAX-HEAP on A , which has worst-case asymptotic complexity $O(n)$. Then it calls HEAP-EXTRACT-MAX k times, each of which has worst-case asymptotic complexity $O(\lg n)$. So the worst-case asymptotic complexity for this algorithm is $O(n + k \lg n)$.

c.

```
 $i \leftarrow$  SELECT( $A, k$ )  
 $A[n] \leftrightarrow A[i]$   
PARTITION( $A$ )  
MERGE-SORT( $A, n - k, n$ )  
return  $A[n - k \dots k]$ 
```

This algorithm first calls SELECT, which has worst-case asymptotic complexity $O(n)$, then calls PARTITION, which also has worst-case asymptotic complexity $O(n)$, then calls MERGE-SORT to sort just the last k elements, which has worst-case asymptotic complexity $O(k \lg k)$. So the worst-case asymptotic complexity for this algorithm is $O(n + k \lg k)$.

Problem 2 (9.3-5):

Suppose that you have a “black-box” worst-case linear-time median subroutine. Give a simple, linear-time algorithm that solves the selection problem for an arbitrary order statistic.

Solution 2:

```

SIMPLE-SELECT( $A, p, r, i$ )
  if  $p = r$ 
    return  $A[p]$ 
   $A[r] \leftrightarrow A[\text{INDEX-OF-MEDIAN}(A, p, r)]$ 
  PARTITION( $A, p, r$ )
   $k \leftarrow \lfloor \frac{r-p+1}{2} \rfloor$ 
  if  $i = k$ 
    return  $A[k]$ 
  else if  $i < k$ 
    return SIMPLE-SELECT( $A, 1, k - 1, i$ )
  else
    return SIMPLE-SELECT( $A, k + 1, r, i - k$ )

```

Problem 3 (9.3-8):

Let $X[1, \dots, n]$ and $Y[1, \dots, n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\lg n)$ -time algorithm to find the median of all $2n$ elements in arrays X and Y .

Solution 3:

```

TWO-LIST-MEDIAN( $X, p, q, Y, s, t$ )
   $mx \leftarrow \text{INDEX-OF-MEDIAN}(X, p, q)$ 
   $my \leftarrow \text{INDEX-OF-MEDIAN}(Y, s, t)$ 
   $X[mx] \leftrightarrow X[q]$ 
   $Y[my] \leftrightarrow Y[t]$ 
  PARTITION( $X, p, q$ )
  PARTITION( $Y, s, t$ )
  if  $X[mx] = Y[my]$ 
    return  $X[mx]$ 
  else if  $X[mx] > Y[my]$ 
    return TWO-LIST-MEDIAN( $X, p, mx - 1, Y, my + 1, t$ )
  else
    return TWO-LIST-MEDIAN( $X, mx + 1, q, Y, s, my - 1$ )

```

Problem 4 (8.1-1):

What is the smallest possible depth of a leaf in a decision tree for a comparison sort?

Solution 4:

Given an array A that contains n elements, the smallest possible depth of a leaf in a decision tree to sort A using a comparison-type sort is $n - 1$. To verify that A is in sorted order takes $n - 1$ comparisons, and if fewer comparisons are used then at least one element was not compared to any of the others, so that element might not be in the correct position.

Problem 5 (Derived from 8.1-4):

You are given a sequence of n elements to sort and a number k such that k divides n . The input sequence consists of n/k subsequences, each containing k elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the k elements in each of the n/k subsequences. Show an $\Omega(n \lg k)$ lower bound on the number of comparisons needed to solve this variant of the sorting problem using a comparison-type sorting algorithm. (*Hint:* It is not rigorous to simply combine the lower bounds for the individual subsequences.)

Solution 5:

We will construct a decision tree for this variant of the sorting problem and show that it has height at least $n \lg k$. Each leaf of the decision tree corresponds to a permutation of the original sequence. How many permutations are there? Each subsequence has $k!$ permutations, and there are n/k subsequences, so there are $(k!)^{n/k}$ permutations of the whole sequence. Thus the decision tree has $(k!)^{n/k}$ leaves. A binary tree with $(k!)^{n/k}$ leaves has height at least $\lg((k!)^{n/k})$.

$$\begin{aligned}\lg\left((k!)^{n/k}\right) &= n/k \lg(k!) \\ &= \Theta(n/k \cdot k \lg k) \\ &= \Theta(n \lg k)\end{aligned}$$

Therefore the lower bound on the number of comparisons needed to solve this variant of the sorting problem using a comparison-type sorting algorithm is $\Omega(n \lg k)$.