

Assignment 6 SAMPLE SOLUTION

1. Suppose you have k sorted lists containing a total of n elements. Show how to merge these lists into a single sorted list in $O(n \lg k)$ time. (This is exercise 6.5-8, p 142.)

(Sol'n) Create a min-heap of size k to store the head of each of the k lists. In addition, store a field which says which list the element came from. (This creation can be done in time $O(k)$.) Now, until there are no more elements left, remove the minimum element from the heap and add it to the output stream. Also, look at the list that the min-element came from, take the next element on that list, and insert it into the heap.

Each removeMin and insert takes time $O(\lg k)$, and there are at most n of them.

2. Let H be a max-heap storing n keys. Give an efficient algorithm for listing all the keys in H that are greater than or equal to a given value x (which may or may not be in H). The keys do not need to be listed in sorted order. Ideally, your algorithm will run in time $O(k)$, where k is the number of keys returned.

(Sol'n) The idea would be to *prune* the search - once you see a key smaller than x , there is no need to search it's descendants, all of which will also be smaller than x .

```
searchPrune (i)
    if i > heapsize then return;
    if A[i] <= x
    then
        output A[i];
        searchPrune(2i);    // visit left subtree of heap
        searchPrune(2i+1); // visit right subtree of heap
```

In the worst case, for each node v whose value is in range, we'll visit it's two children whose values are out of range. So at most $3k$ nodes are visited.

3. A **min-max** heap H combines properties of both a min-heap and a max-heap. It is defined as a complete binary tree with the following property: let v be a node of H at depth i . Then
 - if i is even, then the key stored in node v is less than or equal to any other value stored in the subtree of H rooted at v .
 - if i is odd, then the key stored in node v is greater than or equal to any other value stored in the subtree of H rooted at v .
- (a) Illustrate a min-max heap by drawing one containing 20 elements.
- (b) Describe how to perform the following operations in a min-max heap, and give the running time of each.
 - findMin()

- findMax()
- insert(k)

(Sol'n) Sorry, we'll skip part (a).

findMin(): return root

time: $O(1)$

findMax(): return largest child of root (or root if only one element in tree) time: $O(1)$

insert(k):

```

heapsize++;
i = heapsize;
A[i] = k;
parent = (i div 2) div 2;
if (level of i is odd) then
    while parent in range and A[parent]<A[i]
        swap A[i] and A[parent];
        i = parent;
        parent = (i div 2) div 2;
else
    while parent in range and A[parent]>A[i]
        swap A[i] and A[parent];
        i = parent;
        parent = (i div 2) div 2;

```

time: $O(\lg n)$

4. Exercise 19.2-2, p 472

(Sol'n) Sorry again, it's hard to typeset trees in latex.

5. Exercise 19.2-3, p 472

(Sol'n) Ditto.

6. Suppose you are given an array S of n elements, each of which is colored red or blue. Give an in-place linear-time ordering method which will list all the blue elements before the red elements. Can you extend this to handle three colors?

(Sol'n) Think of blue as being small and red large. Then use Partition (one time) to put all the blue elements on the left side and the red ones on the right.

To handle blue-red-green, first view both red and green as the same color, and run Partition once to put the blue elements before the red and green elements. Then run Partition once more on the red/green section to put the red elements before the green ones.

7. Suppose you have an array S of size n , where each element in S represents a different vote for class president, where each vote is given as an integer representing the student ID of the candidate. Without making any assumptions about who is running or how many candidates there are, design an $O(n \lg n)$ algorithm to determine which candidate receives the most votes.

(Sol'n) Sort the list. Then scan left to right, counting the number of votes for each person. When the votes for a person are done being counted (before moving to the next person), compare that person's votes to the max seen so far.

8. Consider a modification to the previous problem to a situation where we know the number $k < n$ of candidates running. Design an $O(n \lg k)$ algorithm to determine which candidate receives the most votes.

(Sol'n) Create a RB tree, where each node contains a studentID and a vote field. Now look at each vote: if the ID receiving the vote is not in the tree, insert it with a vote of 1, otherwise increment its vote field. Since there are n votes, and each vote causes one find/insert (which take $O(\lg k)$ time, as the tree has size at most k), the total time will be $O(n \lg k)$.

9. Suppose you have a set A of n nuts and a set B of n bolts, such that each nut in A has a unique matching bolt in B . The only kind of comparison you can make is to take a nut-bolt pair (a, b) , where $a \in A$ and $b \in B$, and test to see whether the threads of a are larger, smaller, or a perfect match to the threads of b . Give an efficient algorithm to match up all the nut and bolts. What can you say about the run-time of your algorithm?

(Sol'n) Chose a nut $a \in A$ at random. Use it to partition B into B_0 , the set of bolts too small to fit a , and B_1 , the set of bolts too big to fit a . In addition, we will find b , the bolt that fits a . Now, use bolt b to partition A into A_0 and A_1 .

Recursively match the sets (A_0, B_0) and (A_1, B_1) .

Like Quicksort, it's expected time will be $O(n \lg n)$, with a worst case of $O(n^2)$, with low probability.