Practice Problem Set 3

.

Problem 1

We discussed the both bubble-down and bubble-up in class. Write the pseudocode for bubble-down(i) for index i where the heap is stored in array A. Assume that A.heapsize refers to the number of elements in the heap. Repeat for bubble-up(i).

Problem 2

How do you insert into a heap? Assume that the array that contains the elements is not completely full. Write the pseudo-code and justify the runtime.

Problem 3

Provide a recursive algorithm called BiggerThan(A,x,i) that takes as input a max-heap stored in A and a number x. The parameter i is used to designate the index of root of a sub-heap. The algorithm prints out all numbers from the subheap rooted at index i that are larger than x. Determine the recurrence for the runtime of your algorithm (in the worst-case) and determine the runtime in big-oh notation.

Problem 4

Provide the pseudo-code for an algorithm called HeapDelete(A,,i) that takes as input a max-heap stored in A and an index i. The algorithm deletes the element at index i from the heap and restores the heap property. Justify why the runtime is $O(\log n)$.

Problem 5

Given a max-heap, suppose we would like to output the *minimum* element. Describe a recursive algorithm that solves this problem and provide the pseudo-code. Call your algorithm Find-min(A,i) which returns the minimum element from the sub-heap rooted at index i. Write a recurrence for the worst-case runtime $T(n)$ and determine it's big-oh notation. Is your recursive algorithm asymptotically faster than just a brute-force algorithm?

Problem 6

- Suppose we are given a max-heap stored in array A. Recall that the maximum element is located at $A[1]$. Where can we find the second-largest element? What about the third-largest element?
- Describe an algorithm that returns the *third*-largest element from a max-heap. Justify that your algorithm runs in $O(1)$ time.

Problem 7

A heap can be built using 3 children for each node in the tree instead of 2. How would you represent this heap an array? Describe where to find the children and the parent of node i. How would you update the bubble-down and bubble-up procedures?

Problem 8

- Array A contains elements that are sorted in decreasing order. Is this a heap?
- In class we saw two algorithms for building a heap. Describe the runtime of each of these algorithms when the input array sorted in **decreasing order**

Problem 9

In class we saw that the minimum element of a max-heap is stored in one of the leaves. Suppose we wish to update the heap implementation so that the minimum elements is always stored in the very last index of the heap. In other words, $A[A.heapsize]$ is guaranteed to contain the minimum element. Write the pseudo-code for an algorithm called NewHeapInsert(A, x), where A is the new type of max-heap defined above. You may assume that the input heap has at least one element. The algorithm inserts x into the heap in such a way that after the insert the new minimum is again at index A.heapsize.

Problem 10

The first step of the algorithm Randomized-Select algorithm from class is to partition the elements about a randomly selected element in the array. Write the pseudo-code for Partition(A,s,f) which partitions the elements of array A about a randomly selected pivot, and *returns* the *rank* of the pivot. You may assume that you have access to a function that selects a random number is some range, for example Random(a,b) returns a random integer between the integers a and b inclusively. You may use additional space during the execution of your algorithm if needed.

Problem 11:

Write the pseudo-code for the Randomized-Select algorithm from class. Assume the algorithm takes as input the array A, start and finish indices s and f and the rank k. You may use the partition algorithm from above.

Problem 12:

For each of the following, design an algorithm that takes as input an array of n numbers. (You may make use of the **Select** algorithm from class.) Justify why the runtime of each algorithm is $O(n)$.

- Output all values that are larger than the median element.
- Output the largest 10 elements from the array.
- Output the top $n/4$ elements of the array.
- Output the 2 closest numbers to the median (excluding the median itself).

Problem 13:

Given as input n distinct numbers in an array, describe an algorithm that outputs the k smallest elements in *sorted order*. For example, $\{3, 2, 4, 5, 6, 10, 7, 1, 8, 9\}$ and $k = 4$ would result in $\{1, 2, 3, 4\}$. There are many options for how to approach this. Consider the following 3 approaches and compare the the runtime of each:

1. Sorting the entire array first
2. Building a heap
3. Using the **Select** algorithm to find the kth smallest element and the sorting only a subset of the elements.

Problem 14:

Suppose we carry out a version of the **Select** algorithm where we divide into groups of size $\sqrt{n}$ instead of groups of size 5. Does this alter the complexity of the algorithm? Justify your answer. What if we use groups of size 3?

Problem 15:

The help desk in the IT department receives "*tickets*" representing technical issues that need to be solved by the employees. The tickets are evaluated initially and given a priority number from 1 to 100. When a tech specialist is ready to handle the next ticket, they must be sent the most urgent technical issue right away (urgency is based on the assigned priority number). Describe how you can handle this data, how long it will take you to find the next ticket to be sent to the specialist, and how long it will take you to process a new ticket that arrives.