**CSCI 4041, Spring 2019, Written Assignment 4**

Due Tuesday, 2/19/19, 1:00 PM (submission link on Canvas)

Group17 member:

Thomas Tang - tang0376

Jackie Ji - ji000011

Song Liu - liux4169

This is a collaborative assignment; you may work in a group of 1-3 students. However, you may not consult or discuss the solutions with anyone other than the course instructor, the TAs, or the other members of your group, nor may you use material found from outside sources as part of your solutions. In addition, if you do choose to work in a group, each group member must participate in coming up with the solution to each problem, and must be able to explain the group’s answer if asked: dividing the problems amongst the group members is not acceptable.

Complete the following problems and submit your solutions in a single pdf file to the Written Assignment 4 submission link on Canvas. If you’re working in a group, only one person should submit your answers, but make sure that you include the name and x500 of each group member at the top of the file, and that you are all in one of the Assignment Groups in Canvas. Typed solutions are preferred, but pictures or scans of a handwritten assignment in pdf form are acceptable so long as your solutions are clearly legible.

Your solutions to these problems must be clearly explained in a step-by-step manner; for most problems, the explanation will be worth far more points than the actual answer.

1. (Adapted from exercise 6.4-3)
   1. Show the operation of Heapsort on the array [7, 6, 5, 4, 3, 2, 1]. You should show the array resulting from the call to Build-Max-Heap, and the array after each call to Max-Heapify within the for loop of Heapsort. You must underline all elements in the array that are currently in the heap at each step.

Build-Max-Heap: [7, 6, 5, 4, 3, 2, 1]

Max-Heapify within the for loop of Heapsort:

[ 6, 4, 5, 1, 3, 2, 7]

[5,4,2,1,3,6,7]

[4,3,2,1,5,6,7]

[3,1,2,4,5,6,7]

[2,1,3,4,5,6,7]

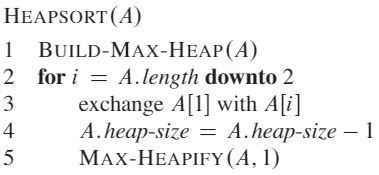
[1,2,3,4,5,6,7]

* 1. What is the asymptotic runtime for Heapsort on an array of distinct elements that is sorted in decreasing order? Justify your answer.

It will be nlgn for the runtime of best case and worst case are the same.

* 1. What is the asymptotic runtime for Heapsort on an array where every element is the same? Justify your answer.

It will be nlgn for the runtime of best case and worst case are the same.



(Exercise 6.4-2 in the textbook) Argue the correctness of the Heapsort algorithm above using the following loop invariant:

At the start of each iteration of the for loop of lines 2-5, the subarray A[1...i] is a max-heap containing the i smallest elements of A, and the subarray A[i+1...n] contains the n - i largest elements of A, sorted.

You may assume that the Build-Max-Heap and Max-Heapify algorithms work correctly. You must make arguments for the Initialization, Maintenance, and Termination of the invariant as part of your answer.

Invariant: Before any given iteration of the for loop, the max value in subarray A[1...i+1] is within A[i+1...A.length] and largest value in subarray A[1...i] is A[1].

Initialization:

Before 1st iteration of the for loop, i = A.length, the statement is vacuously true for A.length+1 does not exist in array A. The max value is A[1] in the subarray A[1...A.length] because of the build-max-heap.

Maintenance:

Assume before kth iteration of the for loop, i = A.length-k+1, the max value in the subarray A[1...A.length-k+2] is within A[A.length-k+2...A.length] and largest value in subarray A[1...A.length-k+1] is A[1].

Proof: before (k+1)th iteration of the for loop, i = A.length-k+1, the max value in the subarray A[1...A.length-k+1] is within A[A.length-k+1...A.length] and largest value in subarray A[1...A.length-k] is A[1].

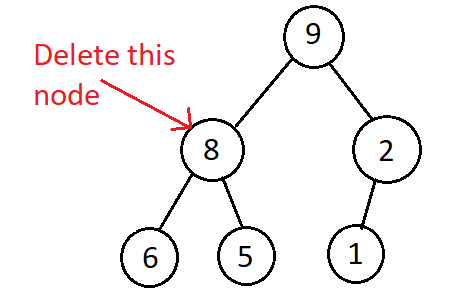
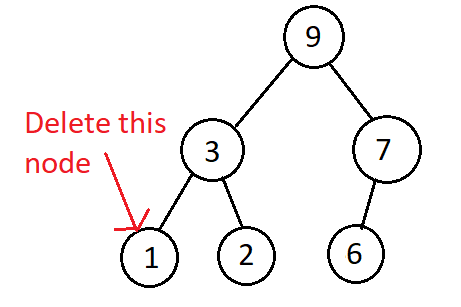
According to the assumption, A[A.length-k+2] is max value in the subarray A[1...A.length-k+2] and the largest value in subarray A[1...A.length-k+1] is A[1]. During the (k+1)th iteration, it will swap A[1] with A[A.length-k+1]. Then the max value in the subarray A[1...A.length-k+1] is within A[A.length-k+1...A.length] which is A[A.length-k+1]. Then, the max-value in A[1...A.length-k] will become A[1] by using Max-Heapify.

Termination:

When the loop is over, the max value in subarray A[1...i] is A[i] and largest value in subarray A[1...i] is A[1].

1. (Exercise 6.5-8 in the textbook) The operation Heap-Delete(A,i) deletes the item at index i from heap A, and ensures that A retains heap structure. Give a pseudocode implementation of Heap-Delete the runs on O(lg n) time for an n-element max-heap. Give a single sentence explanation for why your implementation runs in O(lg n) time.

This may look similar to Heap-Extract-Max, but make sure your code can handle both of the following scenarios correctly:



1:heap-delete(A,i):

2: If A.heap-size<1

3: Error “heap underflow”

4: Exchange A[i] with A[A.heap-size]

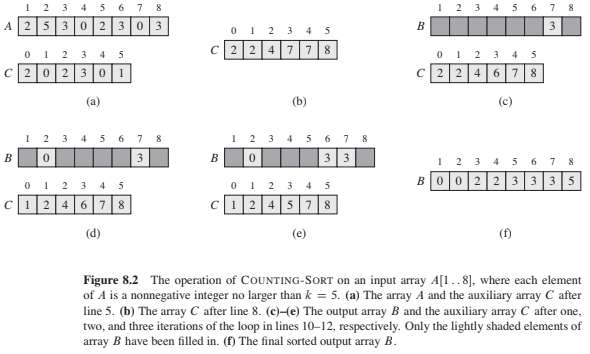
5: A.heap-size--

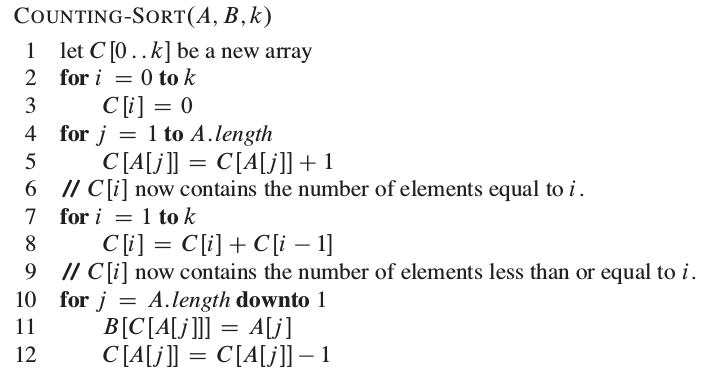
6: Max-Hapify(A,i)

Each line from line 2 to 5 has 1 time complexity, while the line 6 has lg(n) time complexity.

1. (Adapted from Exercise 8.2-1 in the textbook) Using Figure 8.2 from the textbook as a model (see below), illustrate the operation of Counting-Sort on the array

A = [2, 0, 1, 1, 4, 0, 4, 1]. Similar to Figure 8.2, you must show (a) The array A and auxiliary array C after line 5, (b) The array C after line 8, (c)-(e) The output array B and the auxiliary array C after one, two, and three iterations of the loop in lines 10-12, respectively, and (f) The final sorted output array B. You are not required to show the index numbers above each element of the array (though this may help you understand the problem): the format A = [2, 0, 1, 1, 4, 0, 4, 1] will suffice.





1. A = [2, 0, 1, 1, 4, 0, 4, 1]

C = [2,3,1,0,2]

1. C = [2,5,6,6,8]
2. B = [x,x,x,x,1,x,x,x]

C = [2,4,6,6,8]

1. B = [x,x,x,x,1,x,x,4]

C = [2,4,6,6,7]

1. B = [x,0,x,x,1,x,x,4]

C = [1,4,6,6,7]

1. B = [0,0,1,1,1,2,4,4]