CMSC 351 Summer 2025 Homework 5

Due Friday 1 August 2025 by 11:59pm on Gradescope.

Directions:

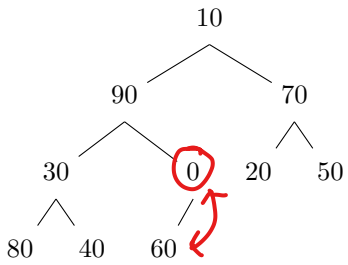
- Homework must be done on printouts of these sheets and then scanned properly, or via Latex, or by downloading, writing on the PDF, and uploading. If you use Latex please do not change the Latex formatting.
- Do not use your own blank paper!
- The reason for this is that Gradescope will be following this template to locate the answers to the problems so if your answers are organized differently they will not be recognized.
- Tagging is automatic, you will not be able to manually tag.

---

1. Here is the pseudocode for `convertomaxheap` with a `print` statement added: [20 pts]

```
function converttomaxheap(A,n)
    for i = floor(n/2) down to 1:
        maxheapify(A,i,n)
        print(A)
    end for
end function
```

Considering the following complete binary tree:



Suppose we represent this as a 1-indexed list and we run `converttomaxheap`. What will `A` look like each time it is printed?

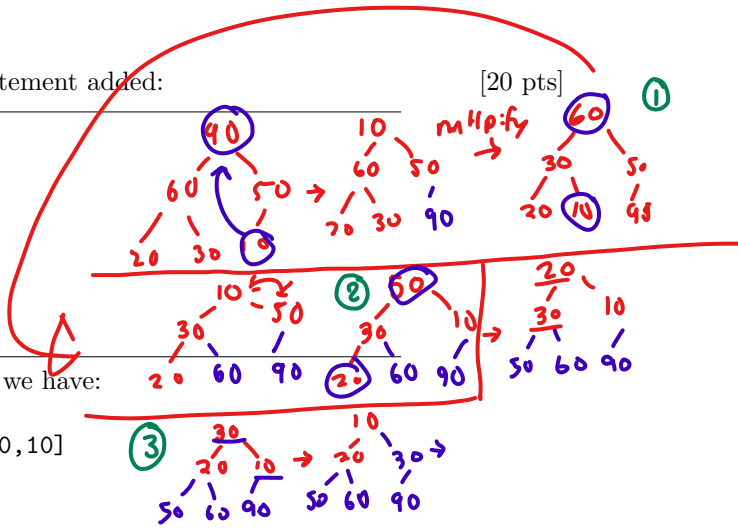
Start	10	90	70	30	0	20	50	80	40	60
The first time, `A =`	10	90	70	30	60	20	50	80	40	0
The second time, `A =`	10	90	70	80	60	20	50	30	40	0
The third time, `A =`	10	90	70	80	60	20	50	30	40	0
The fourth time, `A =`	10	90	70	80	60	20	50	30	40	0
The fifth time, `A =`	90	80	70	40	60	20	50	30	10	0

no swap

no swap

2. Here is the pseudocode for `heapsort` with a `print` statement added: [20 pts]

```
function heapsort(A,n)
    converttomaxheap(A,n)
    for i = n down to 2 inclusive:
        swap(A[1],A[i])
        maxheapify(A,1,i-1)
        print(A)
    end for
end function
```



Suppose we have just finished `converttomaxheap` and we have:

`A = [90,60,50,20,30,10]`

What will `A` look like each time it is printed?

Starting `A =`	90	60	50	20	30	10
The first time, `A =`	60	30	50	20	10	90
The second time, `A =`	50	30	10	20	60	90
The third time, `A =`	30	20	10	50	60	90
The fourth time, `A =`	20	10	30	50	60	90
The fifth time, `A =`	10	20	30	50	60	90

Put scratch work below; Scratch work is not graded:

3. One of the problems with implementing Heap Sort in real life is its unfriendliness with the cache. One result is that in real computers it is much faster to swap list entries whose indices are close. [20 pts]

   Denote by C the *cache limit*. Suppose that when we swap entries in indices x and y it takes time H if $|x - y| \leq C$ (called a *cache hit*) and time M otherwise (called a *cache miss*). The assumption is that $M > C$ but that is not important for the problem.

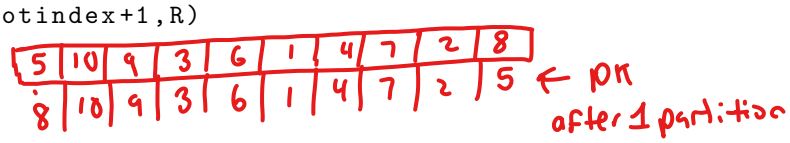
   Suppose we have a complete binary tree with levels indexed $0, ..., L$ and we call `maxheapify` on the node with index 2^i for some i. Moreover suppose that each swap is to the left, the node is swapped all the way to the bottom, and that there are sufficient levels that both cache hits and cache misses occur (this eliminates annoying special cases).

   Calculate the total swap time as a function of L, i, C, H, M. Simplify.

   Solution:

4. Here is the pseudocode for `quicksort` and `partition` with a `print` statement added. Note that this version of `partition` is a little bit different than usual. [14 pts]

```
function quicksort(A,L,R)
    if L < R:
        resultingpivotindex = partition(A,L,R)
        print(A)
        quicksort(A,L,resultingpivotindex-1)
        quicksort(A,resultingpivotindex+1,R)
    end if
end function
function partition(A,L,R)
    swap A[L] and A[R]
    pivotkey = A[R]
    t = L
    for i = L to R-1 inclusive:
        if A[i] <= pivotkey:
            swap A[t] and A[i]
            t = t + 1
        end if
    end for
    swap A[t] and A[R]
    return t
end function
```



Suppose we call `quicksort([5,10,9,3,6,1,4,7,2,8],0,9)`. The `print` statement will run seven times. What is printed each time?

First Time										
Second Time										
Third Time										
Fourth Time										
Fifth Time										
Sixth Time										
Seventh Time										

5. Here is the pseudocode for `quicksort` and `partition` with two `print` statements added, one of which is commented out with a `#`: [12 pts]

```
function quicksort(A,L,R)
    if L < R:
        resultingpivotindex = partition(A,L,R)
        quicksort(A,L,resultingpivotindex-1)
        quicksort(A,resultingpivotindex+1,R)
    end if
end function
function partition(A,L,R)
    print(A)
    pivotkey = A[R]
    # print(pivotkey)
    t = L
    for i = L to R-1 inclusive:
        if A[i] <= pivotkey:
            swap A[t] and A[i]
            t = t + 1
        end if
    end for
    swap A[t] and A[R]
    return t
end function
```

Suppose the output is:

```
[87, 92, 12, 2, 27, 34, 84, 90, 83, 5]
[2, 5, 12, 87, 27, 34, 84, 90, 83, 92]
[2, 5, 12, 87, 27, 34, 84, 90, 83, 92]
[2, 5, 12, 27, 34, 83, 84, 90, 87, 92]
[2, 5, 12, 27, 34, 83, 84, 90, 87, 92]
[2, 5, 12, 27, 34, 83, 84, 90, 87, 92]
```

If the `# print(pivotkey)` were uncommented what would this `print` statement print, in order?

6. We know the best case for Quick Sort occurs when the pivot key always ends up in the middle of the list. Suppose instead it always ends up 1/4 of the way through the list. Assume that `partition` takes time exactly n.

   (a) What would the resulting recurrence relation be? [4 pts]

$T(n) =$	

   (b) There is a generalization of the Master Theorem called the Akra-Bazzi method which can solve this. The solution for this recurrence relation is given by: [10 pts]

   $$T(n) = \Theta\left(n\left(1 + \int_1^n \frac{1}{u}\,du\right)\right)$$

   Evaluate this expression to get the time complexity.

   Note: This shouldn't frighten you, it's very straightforward! I'm only including it so that you can see the value in Calculus as applied to CS.

   Solution: