Gabriel Emerson

COMP 3270

Homework 3

100 points

Please submit using Canvas by 11:59PM on Tuesday, March 16th , 2021

Instructions:

1. This is an individual assignment. You should do your own work. Any evidence of copying will result in a zero grade and additional penalties/actions.
2. Late submissions **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
3. Think carefully; formulate your answers, and then write them out concisely using English, logic, mathematics and pseudocode (no programming language syntax).
4. Type your final answers in this Word document.
5. Don't turn in handwritten answers with scribbling, cross-outs, erasures, etc. If an answer is unreadable, it will earn zero points. **Neatly and cleanly handwritten submissions are acceptable**.

1. (7 points) Heapsort

Show the array A after the algorithm Min-Heap-Insert(A, 6) operates on the Min Heap implemented in array A=[6, 8, 9, 10, 12, 16, 15, 13, 14, 19, 18, 17]. In order to solve this problem you have to do some of the thinking assignment on the Ch.6 lecture slides. But you do not have to submit your solutions to those thinking assignments. Use your solutions to determine the answer to this question and provide the array A below.

A=[19, 18, 17, 16, 15, 14, 13, 12, 10, 9, 8, 6]

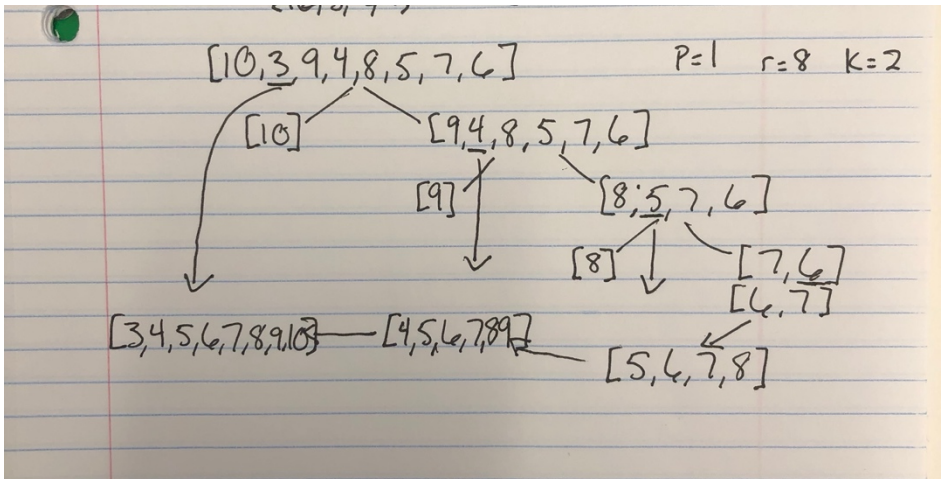
2. (22 points) Quicksort

(a) (6 points)

Quicksort can be modified to obtain an elegant and efficient linear (O(n)) algorithm QuickSelect for the selection problem.

```
Quickselect(A, p, r, k)
{p & r – starting and ending indexes; to find k-th smallest number in non-empty array A; 1≤k≤(r-p+1)}
1 if p=r then return A[p]
else
2       q=Partition(A,p,r)        {Partition is the algorithm discussed in class}
3       pivotDistance=q-p+1
4       if k=pivotDistance then
5               return A[q]
6       else if k<pivotDistance then
7               return Quickselect(A,p,q–1,k)
        else
8               return Quickselect(A,q+1,r, k-pivotDistance)
```

Draw the recursion tree of this algorithm for inputs A=[10, 3, 9, 4, 8, 5, 7, 6], p=1, r=8, k=2. At each non-base case node show all of the following: (1) values of all parameters: input array A, p, r & k; (2) A after Partition. At each base case node show values of all parameters: input array A, p, r & k. Beside each downward arrow connecting a parent execution to a child recursive execution, show the value returned upwards by the child execution.



(b) (16 points). This algorithm has two base cases.
Explain what the first base case that the algorithm checks for is, in plain English:
The first base case is an array with less than two elements because no sorting is required.

List the steps that the algorithm will execute if the input happens to be this base case:
The algorithm will see that the array is not large enough to pass the constraints given, and will then simply return the array.

Complete the recurrence relation using actual constants:
T(first base case) = T(n)

Explain what the second base case that the algorithm checks for is, in plain English:
If K = pivot, (k is not in the array) then return the array.

List the steps that the algorithm will execute if the input happens to be this base case:
It will get through how large the array is, then it will check the parameters to see where the quicksort pivot is. This will find the pivot is outside of the quicksort array, and it will return the array.

Complete the recurrence relation using actual constants (assume complexity of Partition to be 20n):
T(second base case) = T(nlogn-1)

List the steps that the algorithm will execute if the input is not a base case:
The algorithm will check the array size and parameters, then move on to begin sorting around the pivot, and then will call itself until the array is sorted. It will work through each pivot until the end, and then return the sorted array.

Complete the recurrence relation using actual constants (assume complexity of Partition to be 20n and the worst case input size for the recursive call):
T(n) = This will continuously call itself until the array is too small to call a pivot. This will give us T(N^N)

How will the above recurrence change if you instead assume the best case input size for the recursive call):
T(n) = If it was the base case and we got 1, then T(N^N) will be equal to T(1)

3. (10 points) Counting Sort
Show the B and C arrays after Counting Sort finishes on the array A [19, 6, 10, 7, 16, 17, 13, 14, 12, 9] if the input range is 0-19.
C = [0, 0, 0, 0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 7, 7, 8, 9, 9, 10]
B = [6, 7, 9, 10, 12, 13, 14, 16, 17, 19]

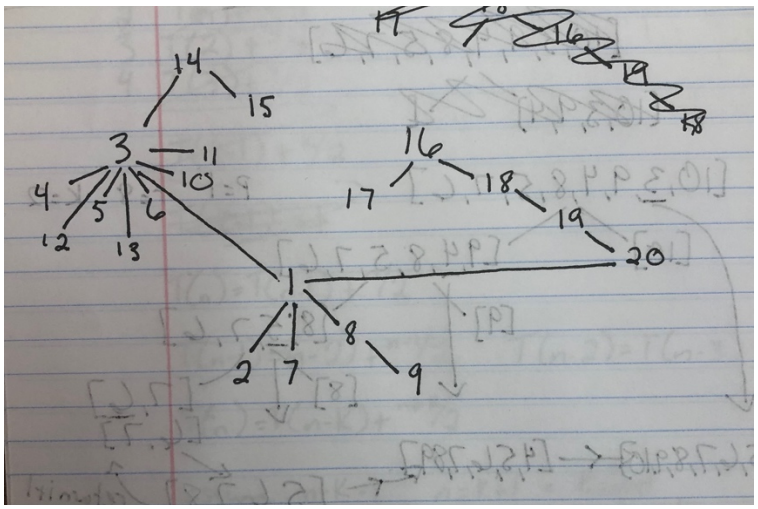
4. (5 points) Radix Sort
If Radix Sort is applied to the array of numbers [4567, 3210, 2345, 4321, 5678], show how these numbers will get rearranged after each of the four passes of the algorithm.
= [3210, 4321, 2345, 4567, 5678]
= [3210, 4321, 2345, 4567, 5678]
= [3210, 4321, 2345, 4567, 5678]
= [2345, 3210, 4321, 4567, 5678]

5. (12 points) Bucket Sort
Consider the algorithm in the lecture slides. If length(A)=15 then list the range of input numbers that will go to each of the buckets 0…14. A = [78, 17, 39, 26, 72, 94, 21, 12, 23, 68]
[0]Bucket0: 12, /
[1]Bucket1: 17, 26, 21, 23, /
[2]Bucket2: 39
[3]Bucket3: /
[4]Bucket4: 72, 68, /
[5]Bucket5: 78, /
[6]Bucket6: 94, /
[7]Bucket7: /
[8]Bucket8: /
[9]Bucket9: /
[10]Bucket10: /
[11]Bucket11: /

[12]Bucket12: /
[13]Bucket13: /
[14]Bucket14: /

Now generalize your answer. If length(A)=n then list the range of input numbers that will go to buckets 0,1,…(n-2), (n-1).
Bucket0: 0 → 1/n
Bucket1: 1/n → 2/n
Bucket(n-2): n-2/n → n-1/n
Bucket(n-1): n-1/n → n/n

6. (20 points**)** Disjoint Set
Assume a Disjoint Set data structure has initially 20 data items with each in its own disjoint set (one-node tree). Show the final result (only show the array P for parts a, b & c below; no need to draw the trees) of the following sequence of unions (the parameters of the unions specified in this question are data elements; so assume that the find operation <u>without path compression</u> is applied to the parameters to determine the sets to be merged): union(16,17), union(18,16), union(19,18), union(20,19), union(3,4), union(3,5), union(3,6), union(3,10), union(3,11), union(3,12), union(3,13), union(14,15), union(14,3), union(1,2), union(1,7), union(8,9), union(1,8), union(1,3), union(1,20) when the unions are:



a. Performed arbitrarily. Make the second tree the child of the root of the first tree.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
16	17	18	19	20	1	3	14	15	4	5	6	10	11	12	13	2	7	8	9

b. Performed by height. If trees have same height, make the 2nd tree the child of the root of the 1st tree.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
16	18	19	20	1	3	14	15	8	9	4	5	6	10	11	12	13	2	7	17

c. Performed by size. If trees have the same size, make the second tree the child of the root of the first tree.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
3	4	5	6	10	11	12	13	14	15	16	17	18	19	20	1	2	7	8	9

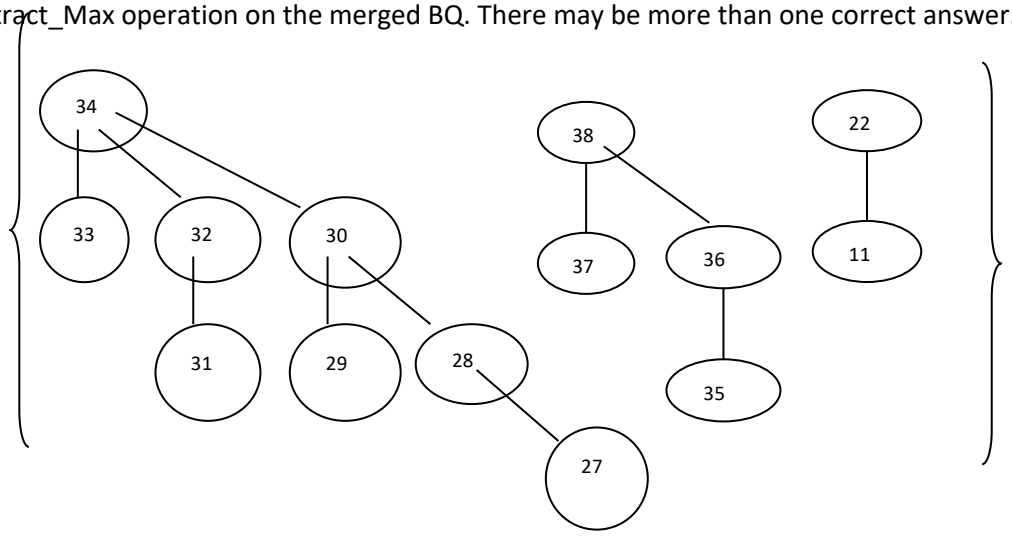
d. For the solution to part a, perform a find with path compression on the deepest node and show the array P after find finishes.

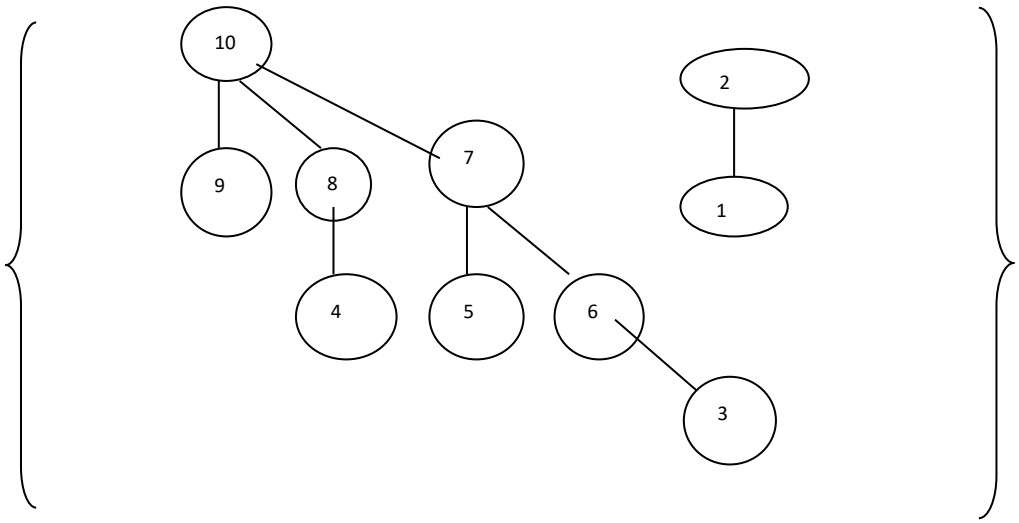
Deepest node is 15

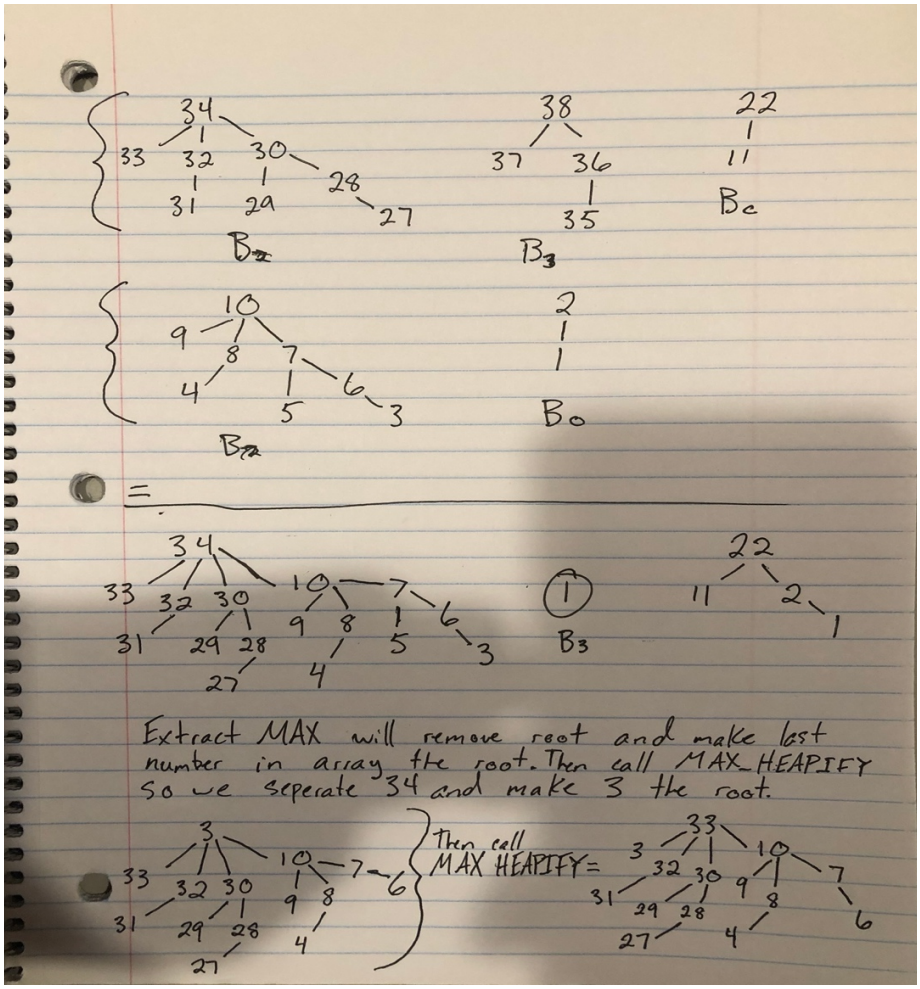
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
16	15	14	3	1	20	19	18	17	4	5	6	10	11	12	13	2	7	8	9

7. (24 points) Binomial Queue

First show the Binomial Queue that results from merging the two BQs below. Then show the result of an Extract_Max operation on the merged BQ. There may be more than one correct answer.




10
9
8
7
2
1
4
5
6
3


34
33
32
30
31
29
28
27
38
37
36
35
22
11
Bc
B3
10
9
8
7
4
5
6
3
2
1
Bo
=
Extract MAX will remove root and make last number in array the root. Then call MAX_HEAPIFY so we seperate 34 and make 3 the root.
Then call MAX HEAPIFY=