COMP 3270

Homework 3

100 points

**Please submit using Canvas by 11:59PM on Friday, July 14th ­­ 2023**

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. (5 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= [6, 8, 6, 10, 12, 9, 15, 13, 14, 19, 18, 17, 16]

**2. (5 points)** Let A be a collection of objects. Describe an efficient O(nlgn) algorithm for converting A into a set. That is, remove all duplicates from A.

Sort the objects of A, then iterate through the sorted sequences and remove all the duplicates form A, sorting the objects take O(nlgn) time and removing the duplicates takes O(n) time, so the overall time complexity is O(nlgn).

**3****. (5 points)** Given a sequence of numbers, S, the mode is the value that appears the most number of times in this sequence. Give an efficient O(nlgn) algorithm to compute the mode for a sequence of n numbers.

Sort the numbers by non-decreasing order, then we scan the sequence to keep track of each run of numbers that are the same, and the length and value of the longest sequence seen so far are stored, after scanning the entire sequence, the value stored is the mode. The running time of this algorithm is dominated by the time to sort the sequence, which can be done in O(nlgn) time using the merge-sort algorithm.

**4. (10 points)** Show that any comparison-based sorting algorithm can be made to be stable, without affecting the asymptotic running time of this algorithm. Hint: Change the way elements are compared with each other.

Change the way elements are compared by replacing each element xi with the pair (xi, i), then perform all comparisons lexicographically to ensure that if xi = xj, the comparison will be resolved using the lexicographic rule (xi, i) < (xj, j) if i < j.

**5. (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.

Quickselect return 4 Partition

A= [10, 3, 9, 4, 8, 5, 7, 6] A = [10, 3, 9, 4, 8, 5, 7, 6]

P= 1, r = 8, k = 2j p =1, r=8j

q = 4 A = [3, 4, 5, 6, 8, 9, 7, 10]

return4



Quickselect return 4 Partition

A= [3, 4, 5, 6, 8, 9, 7, 10] A = [3, 4, 5, 6, 8, 9, 7, 10]

P= 1, r = 3, k = 2k p = 1, r = 3j

q = 3 A = [3, 4, 5, 6, 8, 9, 7, 10]

return4



Quickselect return 2 Partition

A= [3, 4, 5, 6, 8, 9, 7, 10] A = [3, 4, 5, 6, 8, 9, 7, 10]

P = 1, r =2 , k = 2j p = 1, r = 2j

q = 2 A = [3, 4, 5, 6, 8, 9, 7, 10]

(b) (16 points). This algorithm has two base cases.

Explain what the first base case that the algorithm checks for is, in plain English:

It checks for when the starting index = the ending index, first base case means that there is only one element in the array.

List the steps that the algorithm will execute if the input happens to be this base case:

If p = r then return A[p]

Complete the recurrence relation using actual constants:

T(first base case) = \_\_\_\_\_\_\_\_\_\_\_\_7\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Explain what the second base case that the algorithm checks for is, in plain English:

It checks for is when the desired rank = to the distance of the pivot elment, this means that the pivot element itself if the k-th smallest element in the array

List the steps that the algorithm will execute if the input happens to be this base case:

1 If p = r then return A[p]

else

2 q=Partition(A,p,r)

3 pivotDistance=q-p+1

4 if k=pivotDistance then

5 return A[q]

Complete the recurrence relation using actual constants (assume complexity of Partition to be 20n):

T(second base case) = \_\_\_\_\_\_\_\_\_\_\_\_20n + 16\_\_\_\_\_\_\_\_\_\_\_\_\_\_

List the steps that the algorithm will execute if the input is not a base case:

1 if p=r then return A[p]

else

2 q=Partition(A,p,r)

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)

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) = \_\_\_\_\_\_\_\_\_\_\_\_\_T((n-1)(20n + 16 + T(n-1))) + 7\_\_\_\_\_\_\_\_\_\_\_\_\_

How will the above recurrence change if you instead assume the best case input size for the recursive call):

T(n) = \_\_\_\_\_\_\_\_\_\_T(n/2)+20n+15\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6. (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.

A: indexed from 1 to 10 [19, 6, 10, 7, 16, 17, 13, 14, 12, 9]

C: indexed from 0 to 19 [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1]

New C: indexed from 0 to 19 [0, 0, 0, 0, 0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 7, 7, 8, 9, 9, 10]

When finished:

B: indexed from 1 to 10 [6, 7, 9, 10, 12, 13, 14, 16, 17, 19]

Reduced C: indexed from 0 to 19 [0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 7, 7, 8, 9, 9]

**7. (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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4567 | 3210 | 2345 | 4321 | 5678 |
| 3210 | 4321 | 2345 | 4567 | 5678 |
| 3210 | 4321 | 2345 | 4567 | 5678 |
| 3210 | 4321 | 2345 | 4567 | 5678 |
| 2345 | 3210 | 4321 | 4567 | 5678 |

**8. (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.

Bucket0: [0..1/15)

Bucket1: [1/15..2/15)

Bucket2: [2/15..3/15)

Bucket3: [3/15..4/15)

Bucket4: [4/15..5/15)

Bucket5: [5/15..6/15)

Bucket6: [6/15..7/15)

Bucket7: [7/15..8/15)

Bucket8: [8/15..9/15)

Bucket9: [9/15..10/15)

Bucket10: [10/15..11/15)

Bucket11: [11/15..12/15)

Bucket12: [12/15..13/15)

Bucket13: [13/15..14/15)

Bucket14: [14/15..15/15]

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)

**9. (10** 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 without path compression 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 |
| 0 | 1 | 14 | 3 | 3 | 3 | 1 | 1 | 8 | 3 | 3 | 3 | 3 | 1 | 14 | 18 | 16 | 19 | 20 | 1 |

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 |
| -3 | 1 | 14 | 3 | 3 | 3 | 1 | 1 | 8 | 3 | 3 | 3 | 3 | 1 | 14 | 1 | 16 | 16 | 16 | 16 |

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 | 1 | -20 | 3 | 3 | 3 | 1 | 1 | 8 | 3 | 3 | 3 | 3 | 3 | 14 | 3 | 16 | 16 | 16 | 16 |

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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 0 | 1 | 14 | 3 | 3 | 3 | 1 | 1 | 8 | 3 | 3 | 3 | 3 | 1 | 14 | 1 | 1 | 1 | 1 | 1 |

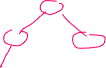
**10. (16 points)** Binomial Queue

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.

Merge B0 tree resulting B2 tree.

22

2 11

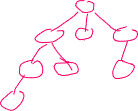


1

Merge trees of order 2 resulting B2 tree

38

22 37 36



2 11 35

1

Merge trees of order 3 resulting B3 tree

34



10



33 32 30 9 8 7

31 29 28 4 5 6

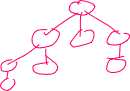


27 3

Final merge tree

38

27 37 36



2 11 35

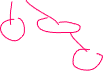
1

34

10 33 30



9 8 7 32 29 28



4 5 6 31 27

3

