COMP 3270

Homework 3 Solution

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

**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]

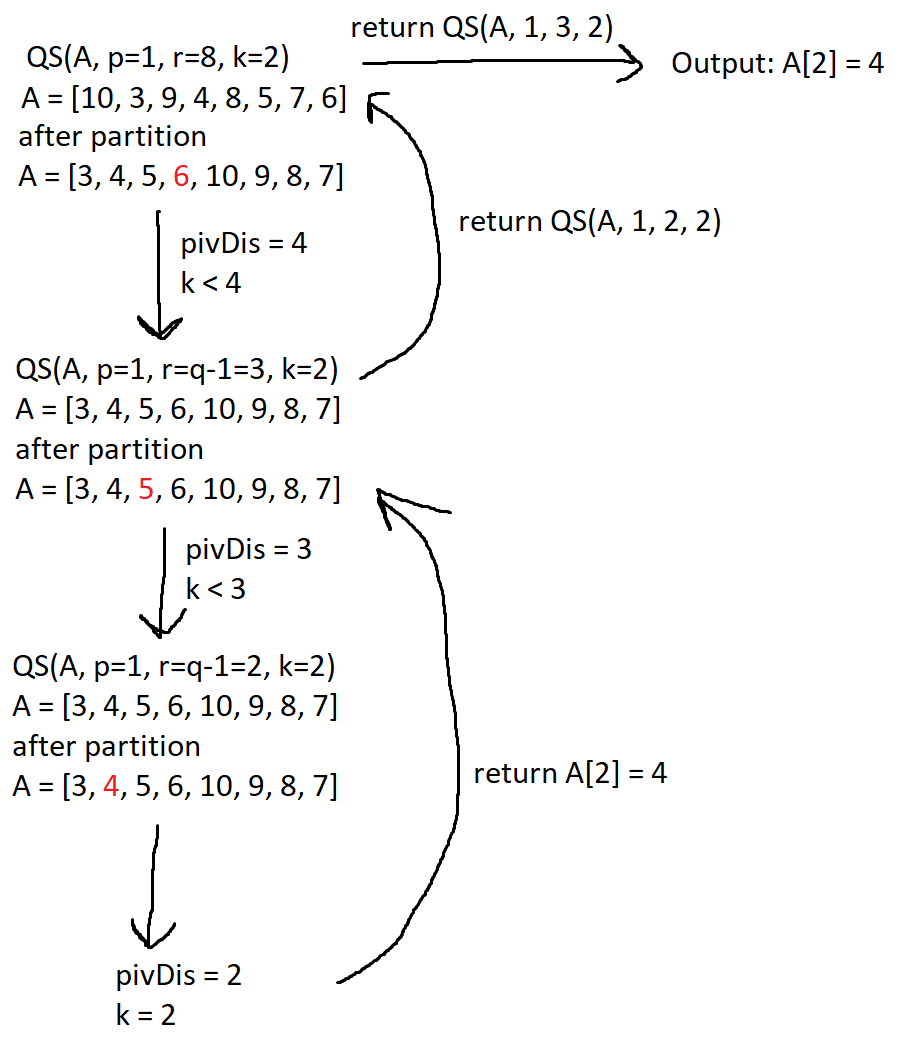
6 else if k<pivotDistance then

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:

If the array is one element long, return that element

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

Only step 1 is executed

Complete the recurrence relation using actual constants:

T(first base case) = 3 + 4 = 7 (3 for if, 4 from return)

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

If the distance the pivot is from the start is equal to k, return the value of the pivot

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

The if statement evaluates p != r, so the else block executes. Lines 2-5 execute as the base case is the first case of the next if statement. Partition is called to find q, pivotDistance is calculated, k is found equal to pivotDistance, then the algorithm returns A[q].

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

T(second base case) = 3 + 20n + 1 + 5 + 3 + 4 = 20n + 16 (3 from first if, 20n from Partition call, 1 from setting the Partition to q, 5 for calculating pivotDistance, 3 for next if statement, 4 to return A[q])

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

When the algorithm does not have a base case input, all that changes is where the second if statement ends. It begins the same where the first if statement jumps to the else block. Lines 2 and 3 then find q and pivotDistance. Line 4 finds that k != pivot Distance, so it then finds if k < pivotDistance. If so, Quickselect is run on the side less than the partition. If not, then k > pivotDistance and Quickselect runs on the upper portion.

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) = 3 + 20n + 1 + 5 + 3 + 3 + T(n-1) = 20n + 15 + T(n-1) (3 from first if, 20n from Partition call, 1 from setting the Partition to q, 5 for calculating pivotDistance, 3 for next if statement, 3 for else if, T(n-1) as the worst case is when the partition is all the way to one side and the rest of the elements go in to the recursive call)

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

T(n) = 3 + 20n + 1 + 5 + 3 + 3 + T(1) = 20n + 15 + T(1) = 20n + 15 + 7 = 20n + 22 (3 from first if, 20n from Partition call, 1 from setting the Partition to q, 5 for calculating pivotDistance, 3 for next if statement, 3 for else if, T(1) is the best case call as only one element will be lift to consider. The next call is guaranteed to be the base case so we can substitute the previously calculated value)

**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.

B[6, 7, 9, 10, 12, 13, 14, 16, 17, 19]

C[0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 7, 7, 8, 9, 9]

**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.

1: [3210, 4321, 2345, 4567, 5678]

2: [3210, 4321, 2345, 4567, 5678]

3: [3210, 4321, 2345, 4567, 5678]

4: [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.

Bucket0: [0, 1/15) or [0, 0.0667)

Bucket1: [1/15, 2/15) or [0.0667, 0.1333)

Bucket2: [2/15, 3/15) or [0.1333, 0.2)

Bucket3: [3/15, 4/15) or [0.2, 0.2667)

Bucket4: [4/15, 5/15) or [0.2667, 0.3333)

Bucket5: [5/15, 6/15) or [0.3333, 0.4)

Bucket6: [6/15, 7/15) or [0.4, 0.4667)

Bucket7: [7/15, 8/15) or [0.4667, 0.5333)

Bucket8: [8/15, 9/15) or [0.5333, 0.6)

Bucket9: [9/15, 10/15) or [0.6, 0.6667)

Bucket10: [10/15, 11/15) or [0.6667, 0.7333)

Bucket11: [11/15, 12/15) or [0.7333, 0.8)

Bucket12: [12/15, 13/15) or [0.8, 0.8667)

Bucket13: [13/15, 14/15) or [0.8667, 0.9333)

Bucket14: [14/15, 15/15) or [0.9333, 1)

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, 1)

**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 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 |
| -4 | 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 |

A picture containing diagram

Description automatically generated

**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.

Answer on next page

