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.

First we sort the objects of A using an efficient sorting method with worst-case O(n lg n)

time, like Mergesort. Then, we can linearly go through the sorted sequence and remove all

duplicates. The duplicates will be side-by-side. It takes O(n lg n) time to sort and O(n) time

to remove the duplicates. Overall this is an O(n lg n) algorithm.

**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.

Solution: Merge-sort.

We sort the array and then we scan from left to right for the longest sequence of the same element.

Sorting takes 𝛩(nlgn)

Going through the array will take 𝛩(n) because the input will be n numbers and the algorithm will go through all of the array to see which element has the most out of the entire sequence.

For the most frequently occurring element, it can be put into a counter that can keep track of the “longest sequence” and ‘item in the sequence’.

Then compose the new sequence you have found and if it is longer assign it to the counter.

Another solution: Counting sort but it would only work with a simple range of elements(k) because if k = n^2 then the algorithm runs at k times, which is higher than nlgn. If the data comes from all possible reals, k is infinite, counting just can’t be done.

**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.

Can be modified to be stable by changing the key comparison operation so that the comparison of two keys considers position as a factor for objects with equal keys.

This can be done by replacing each xi with a pair (xi, i) where i is the position where xi is positioned.

This will make the comparisons lexicographically which will guarantee that if xi = xj then the comparison will be resolved by using the lexicographic rule that (xi, i) < (xi, 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]

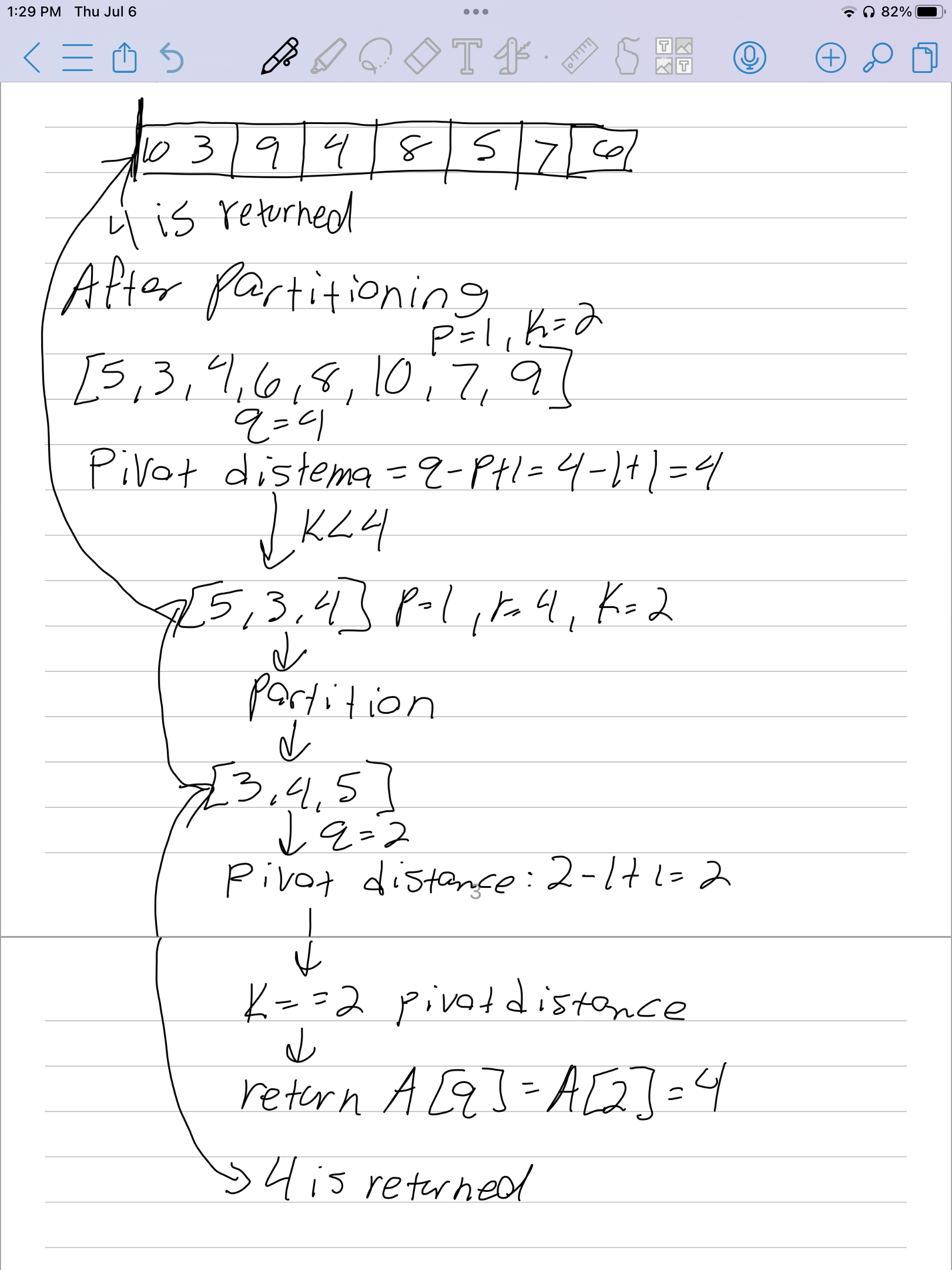
6 else if k<pivotDistance then

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.



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

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

First base case:

if (p == r) return A[p]

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

When the starting and ending indexes in the quick select are same then kth smallest number is nothing but the value at the starting index(i.e ending also) A[p]. We just return the A[p]

Complete the recurrence relation using actual constants:

T(first base case) = \_\_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

if the pivotElement itself is the kth smallest number then we can return the value at the pivot index in the Array. i.e A[q].

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

Second base case:

else

q = partition(a,p,r);

pivotDistance = q-p+1

if(k==pivotDistance)

return A[q];

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

T(second base case) = \_\_\_\_\_20n+6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

not base case

else

q = partition(a,p,r);

pivotDistance = q-p+1

else if(k<pivotDistance) then

return quickselect(A,p,q-1,k)

else

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 + 6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

T(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ T((n-1)/2)+20n+6\_\_\_\_\_\_\_\_\_\_\_\_\_

**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**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **19** | **6** | **10** | **7** | **16** | **17** | **13** | **14** | **12** | **9** |

**C**

**First count**

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

**C**

**After adding A[j] + A[j-1]**

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

**C**

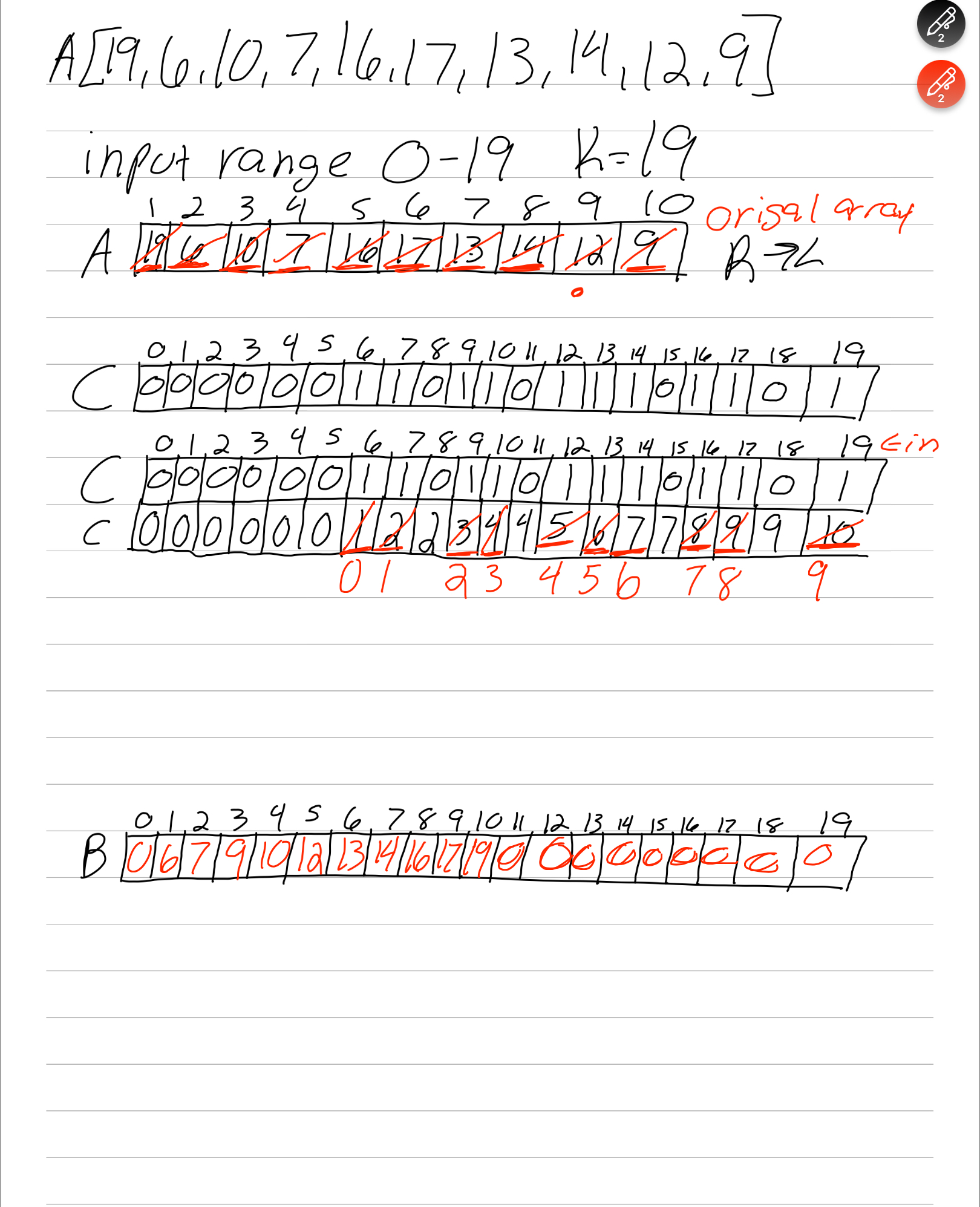
**After subtracting c[A[j] = a[A[j]]-1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **2** | **2** | **3** | **4** | **4** | **5** | **6** | **7** | **7** | **8** | **9** | **9** |

**B**

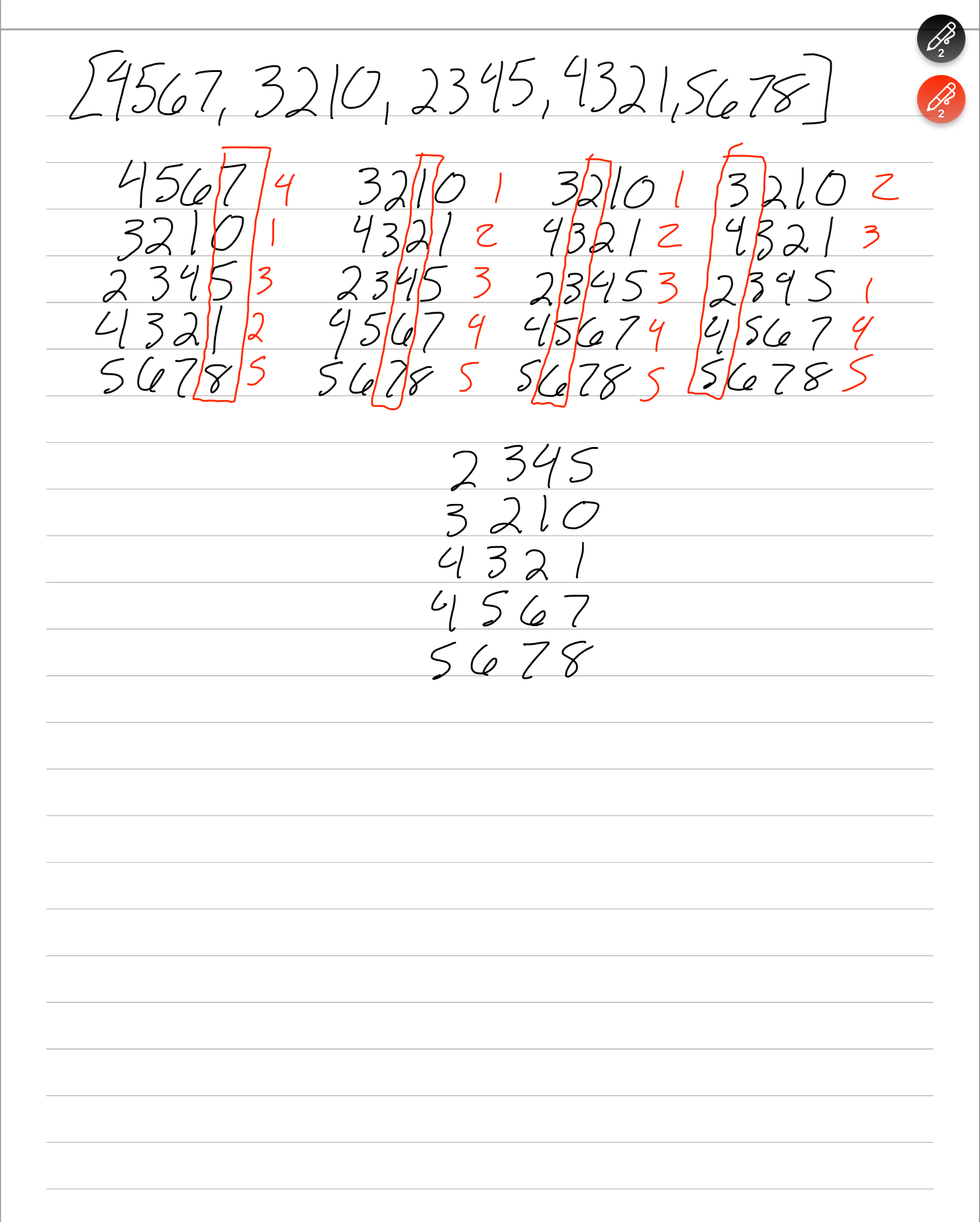
**After B[c[A[j]]] = A[j]**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **6** | **7** | **9** | **10** | **12** | **13** | **14** | **16** | **17** | **19** |  |  |  |  |  |  |  |  |  |

****

**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.



**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:[0.0, .10)

Bucket1: [0.10, 0.20)

Bucket2: [0.20, 0.30)

Bucket3: [0.30, 0.40)

Bucket4: [0.40, 0.50)

Bucket5: [0.50, 0.60)

Bucket6: [0.60, 0.70)

Bucket7: [0.70, 0.80)

Bucket8: [0.80, 0.90)

Bucket9: [0.90, 1.0)

Bucket10: [1.0, 1.1)

Bucket11: [1.1, 1.2)

Bucket12: [1.2, 1.3)

Bucket13: [1.3, 1.4)

Bucket14: [1.4, 1.5)

For each bucket the range of numbers is ith index bucket [i/10 to (i+1)/10) excluding (i+1)/10 while including i/10

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: 0/10, (0+1)/10 = [0.0, 0.1)

Bucket1: [1/10, (1+1)/10) = [.10, .20)

Bucket(n-2): n-2/10, (n-2 +1) /10 = [n-2/10, n-1/10)

Bucket(n-1): n-1/10, (n-1+1)/10 = [n-1/10, n/10)

**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 |
| 1 | 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 |
| 1 | 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 | 3 | 3 | 3 | 3 | 1 | 1 | 7 | 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 |
| 1 | 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.









