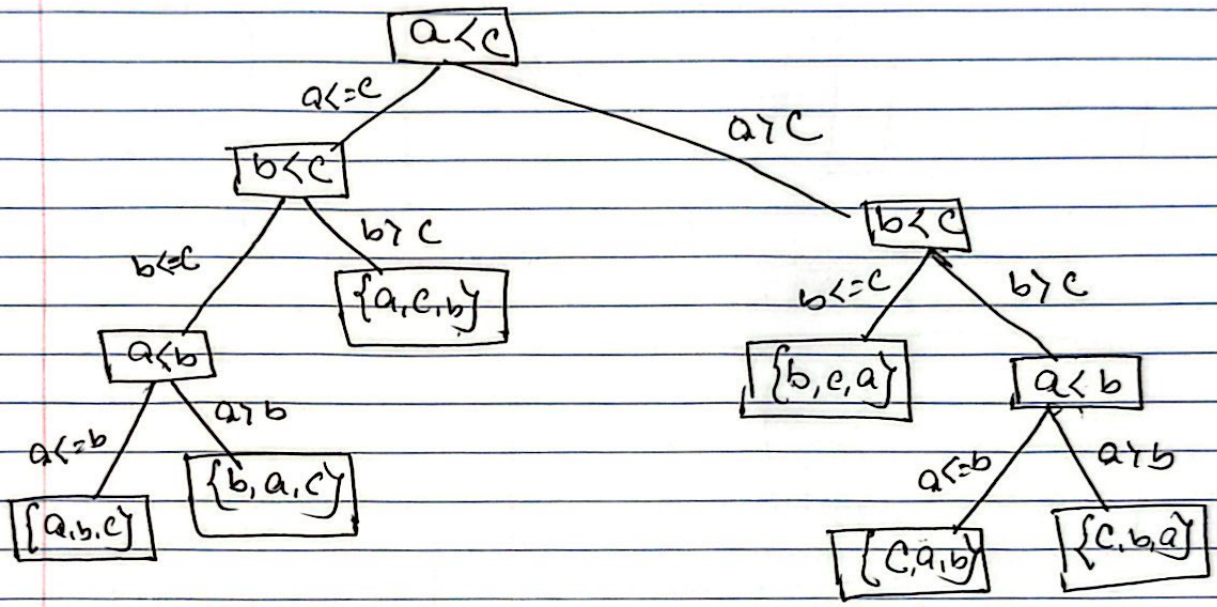
Analysis of Algorithms

Assignment 4

Pratik Dey

Ans: to the Ques: No: 1

Let the 3 inputs be a, b, c



Ans: to the Ques: No: 2

646	196	619	920
920	167	920	541
619	541	541	661
853	582	646	582
864	646	853	853
541 →	619 →	167 →	864
196	678	661	646
582	661	864	196
167	853	678	167
678	864	582	678
661	920	196	619
↑	↑	↑	

Here the outcome of the array is not sorted. Doing radix sort on MSD can cause incorrect output as the order may not be handled at each step, which is a crucial part.

To overcome this problem we pair each element in 3 ~~passes~~. ~~and~~ steps

Step 1: Group and sort using the MSD:

Group 1: 196, 167
Group 2: 541, 582
Group 3: 646, 619, 661, 678
Group 4: 853, 864
Group 5: 920

Step 2: Sort within groups by the next most significant digit

Group 1: 167, 196
Group 2: 541, 582
Group 3: 619, 646, 661, 678
Group 4: 853, 864
Group 5: 920

Step 3: Sort within group by the least significant digit.

Group 1: 167, 196
Group 2: 541, 582
Group 3: 619, 646, 661, 678
Group 4: 853, 864
Group 5: 920

Next we merge the groups and get the final sorted array:

167, 196, 541, 582, 619, 646, 661, 678, 853, 864, 920.

—

Ans: to the Ques: No: 3

For an array of n values in the ranges 0 to 2n, counting sort will have the best worst-case running time possible.

Counting sort algorithm is ideal for sorting integers when the range of the values is known and relatively small compared to the size of the array. Counting sort is optimal here because the values are integers in the range 0 to 2n, meaning the number of unique values is at most 2n+1. Counting sort sorts the array occurrences of each value, then reconstructing the sorted array based on those counts. Here knowing the range of the array makes it ideal for counting sort as its time complexity depends on the range.

The worst case time complexity is $O(n+k)$ where n is the number of elements and K is the range of input. So the time complexity is $O(n+2n) = O(n)$ which is linear time.
