Name: Yogendra Pokhrel
617585

1. Show that any comparison based algorithm to sort 4 elements requires at least 5 comparisons in the worst case.

To prove that any comparison-based sorting algorithm requires at least 5 comparisons to sort 4 elements in the worst case, let's analyze it step by step.

A. Understanding Comparison-Based Sorting

A comparison-based sorting algorithm determines the order of elements only by comparing pairs of elements (e.g., A[i] < A[j]). Examples include QuickSort, MergeSort, and Insertion Sort.

B. Possible Arrangements of 4 Elements

If we have 4 elements, say A, B, C, and D, they can be arranged in 4! = 24 different ways (since there are 4 factorial permutations).

C. Decision Tree Model

A comparison-based sorting algorithm can be represented as a binary decision tree, where:

- Each internal node represents a comparison (e.g., A < B).
- Each leaf node represents a final sorted order.
- The number of leaf nodes is at least 24 (one for each possible permutation).

D. Minimum Comparisons Required

To sort the elements, we must distinguish among 24 different permutations. The depth of a binary decision tree gives the worst-case number of comparisons.

Since a binary decision tree with height h has at most 2^h leaves, we require:

2^h >= 24

Taking the logarithm on both sides:

h >= $\log_2 24$

Approximating:

Log_2 24 approx to 4.58

Since comparisons must be whole numbers, we round up to get at least 5 comparisons.

E. Example Case: Worst Case with 5 Comparisons

Consider sorting A, B, C, D using Insertion Sort:

1. Compare A and B → Swap if necessary.
2. Compare C with A or B (insertion step).
3. Compare C again if necessary to find its correct place.
4. Compare D with an already sorted subset.
5. Compare D again if necessary.

In the worst case, 5 comparisons are needed.

F. Conclusion

No comparison-based sorting algorithm can sort 4 elements using fewer than 5 comparisons in the worst case. This result follows from the decision tree complexity bound, which shows that the minimum comparisons required for N elements is approximately $\log_2(N!)$

2. Explain how RadixSort can be used to sort the following S={125, 27, 729, 1, 27, 8, 64, 343, 216} using radix = 9

Radix Sort is a non-comparison-based sorting algorithm that sorts numbers by processing individual digits, from the least significant digit (LSD) to the most significant digit (MSD) (LSD Radix Sort).

Given the set:

S = {125, 27, 729, 1, 27, 8, 64, 343, 216}

and radix = 9, this means that we use 9 buckets (0-8) for sorting.

Step 1: Identify Maximum Digits

The largest number in S is 729 (3 digits). Thus, we need 3 passes (one per digit position: units, tens, and hundreds).

Pass 1: Sorting by Least Significant Digit (LSD) – Unit Place

We group numbers based on their unit (rightmost) digit:

Number	Unit Digit	Bucket
125	5	5
27	7	7
729	9	9
1	1	1
27	7	7
8	8	8
64	4	4
343	3	3
216	6	6

After placing numbers in buckets (0-8) and reading them in order:

Sorted order after Pass 1:

1, 343, 64, 125, 216, 27, 27, 8, 729

Pass 2: Sorting by Tens Digit

Now, we sort based on the tens digit:

Number	Tens Digit	Bucket
1	0	0
343	4	4
64	6	6
125	2	2
216	1	1
27	2	2
27	2	2
8	0	0
729	2	2

Reading buckets in order:

Sorted order after Pass 2:

1, 8, 216, 125, 27, 27, 729, 343, 64

Pass 3: Sorting by Hundreds Digit

Now, we sort based on the hundreds digit:

Number	Hundreds Digit	Bucket
1	0	0
8	0	0
216	2	2
125	1	1
27	0	0
27	0	0
729	7	7
343	3	3
64	0	0

Reading buckets in order: Final sorted order:

1, 8, 27, 27, 64, 125, 216, 343, 729

Final Answer:

After 3 passes, we get the sorted sequence:

S = {1, 8, 27, 27, 64, 125, 216, 343, 729}

Thus, Radix Sort successfully sorts the list using radix = 9.

3. Carry out the steps of radixSort to sort the following {80, 27, 72,1,27,8,64,34,16} Hint use 9 for your radix

using radix = 9. Since the largest number is 80 (2 digits), we need 2 passes (one for the unit place and one for the tens place).

Pass 1: Sorting by Unit Place (Least Significant Digit)

We group numbers based on their unit (rightmost) digit:

Number	Unit Digit	Bucket
80	0	0
27	7	7
72	2	2
1	1	1
27	7	7
8	8	8
64	4	4
34	4	4
16	6	6

Reading numbers from buckets in order (0 → 8):

Sorted order after Pass 1:

80, 1, 72, 64, 34, 16, 27, 27, 8

Pass 2: Sorting by Tens Place

Now, we sort based on the tens digit:

Number	Tens Digit	Bucket
80	8	8
1	0	0
72	7	7
64	6	6
34	3	3
16	1	1
27	2	2
27	2	2
8	0	0

Reading numbers from buckets in order:

Final sorted order:

1, 8, 16, 27, 27, 34, 64, 72, 80

Final Answer:

After 2 passes, the sorted sequence is:

S = {1, 8, 16, 27, 27, 34, 64, 72, 80}

Thus, Radix Sort successfully sorts the list using radix = 9.