Sorting → decreasing / increasing

arranging things in order

→ numerical
→ alphabetical
→ based on properties of elements.

① Array (integers) → asc / desc

② Array of measurements of rectangle length / breadth.

[[l₁, b₁], [l₂, b₂], ... [lₙ, bₙ]]

→ dependent on one property

① $l_i < l_j$
② $b_i < b_j$
③ $A[i] < A[j]$ for properties
④ permutation

Q given an array of integers, which would be the 2 nd smallest and 2nd largest in it?

Sol

Sorted version of an array is nothing but permutations of elements of array

Eg: 2 3 1 4

sort → 1 2 3 4

(asc)

→ Find all permutations of given array and then print 2nd smallest & largest.

TC → $n \times n!$

BS / IS / SS → $O(n^2)$ >>> $O(n \times n!)$

Why is sorting required ??

→ Benefits of sorted array vs unsorted array

① Search for a specified value.
② make min or max smallest etc.
③ remove duplicate etc.
④ merge etc.
⑤ count the freq of elements etc.

Q → Racing Horses

eg → 4 9 1 32 13

↓

sort (asc order)

↓

4 9 13 32

min = 4 2nd 9 3rd 13

I (sorted) sort

pick 2 cards

sorted

left hand

sorted

right hand

I

II

III

IV

V

In each iteration, I was taking every element from the unsorted region & placing it at right position in the sorting position.

A → [4 2 5 2]

sorted unsorted

Size → 4

n-1 iterations

Pass

4 < 2

[4 5 7 2]

sorted unsorted

1st

7 > 5
7 < 5

[4 5 7 2]

sorted unsorted

2nd

swap

7 > 2
5 > 2
4 > 2

[2 4 5 7]

sorted

3rd

Best case

[4 2 5 9]

left

[6 7 5 9]

2nd

[5 6 7 9]

n-1 iter

3rd

[4 5 6 7]

(n-1) iterations

(n-1 depth)

(n-1) × (n-1) ≅ $O(n^2)$

Time Complexity

① worst case → $O(n^2)$

② best case → $O(n-1) \cong O(n)$

Best case

[1 2 3 4]

1st

[1 2 3 4]

2nd

[1 2 3 4]

3rd

[1 2 3 4]

Space Complexity

→ $O(1)$ [In-place sorting algo]

constant extra space

Insertion Sort Applications

→ ① when you need to sort on stream of data

[1 4 3 9 7]

→ min no of swaps
→ 2 swaps

→ ② online sorting

1 3 5 6 7 ...

→ ③ In-place sorting algo need

key = arr[i] → 1

I Bubble sort

→ works by repeatedly swapping the adjacent elements if they are in wrong order

n = 5

I iteration

(n-1) swaps

II iteration

(n-2) swaps

III iteration

(n-3) swaps

IV iteration

(n-4) swaps

n-1 terms

TC → (n-1) + (n-2) + (n-3) + ... 1

$= \frac{n(n-1)}{2}$ swaps

→ $\cong O(n^2)$

Time Complexity

worst → $O(n^2)$

Best → $O(n)$ (adaptive)

Avg → $O(n^2)$

$O(n)$

Space Complexity

→ $O(1)$

In-place sorting algo

Applications

① simple & easy to understand
→ short code

② complexity doesn't matter