Wednesday, 8 September 2021 8:10 PM

Sorting → ascending / descending

arranging things in order
→ ascending
→ descending
→ based on properties of elements

① Array (integers) → asc / desc

② Array of measurements & weights
[[h_1, w_1], [h_2, w_2], ... [h_n, w_n]]
① $h_i < h_j$
② $w_i < w_j$
③ $h[i] < h[j]$ → properties
④ permutation → computation
→ dependent on the

Q. Given an array of integers, what would be the most easy way (most stable way) to sort it?
→ ascending

→ Sorted version of an array i.e. nothing but permutations of the original array

Eg. 2 3 1 4

ans → 1 2 3 4 n → n! possibilities

→ Find all permutations of given array and pick the sorted one.

TC → (n!) × n → worst case

Q.S | I.S | S.S → $O(n^2)$ <<< $O(n \times n!)$

Why is sorting required??
→ Benefits of sorted array over unsorted array.

1. Search for a specified value.
2. max/min or k^{th} smallest etc.
3. remove duplicate values
   1 2 3 3 4 5 6
4. count the freq of etc.
   1 2 3 3 3 4 5 6
   freq of 3

4 9 1 32 12

↓

sort (asc / desc)

↓

[4 9 12 32]

3 6 9 19

sorted

I → Insertion sort

sorted | unsorted

9 6 11 8

left hand | right hand

I

9 | 6 11 8

L | R

6 < 9

II

6 9 | 11 8

L | R

9 < 11

III

6 9 11 | 8

L | R

8 > 6
8 > 9
6 < 8

IV

6 9 11 8

L | R

In each iteration, elements are picked up from the unsorted region and placed at their right position in the sorted region.

A → [1 | 2 5 2] (sorted | unsorted) size → 1st elt

4 < 7

n-1 iterations

A → [5 | 3 2] (sorted | unsorted) → 2nd itr

3 > 5
4 < 5

A → [2 | 4 5 2] (sorted | unsorted) → 3rd itr

3 > 2
5 > 2
4 > 2

A → [2 4 5 | 2] (sorted)

Done

→ Implementation

worst case → [3 2 5 4]

[2 3 5 4]
[2 3 4 5]
[2 2 3 4 5] 4th

(n-1) iterations
→ (n-1) swaps

(n-1) × (n-1) ≈ $O(n^2)$

Time Complexity

① worst case → $O(n^2)$
② best case → $O(n)$

Best case → [1 2 3 4]
[1 2 3 4] 1st
[1 2 3 4] 2nd
[1 2 3 4] 3rd

Space Complexity
→ $O(1)$ [In-place sorting algo]
↳ constant extra space

Insertion Sort Applications

① when you need to sort an almost sorted array
[4 7 3 8 9]
→ min 4 steps
→ 2 steps

② online queues

1 3 5

③ In-place sorting algo

II Bubble sort

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

n = 5

I → 2 5 3 6 1

2 6 5 3 4 (n-1) swaps
2 5 6 3 4
2 5 3 6 4

I

2 5 3 6 4
2 5 3 6 4 . (n-2) swaps
2 3 5 6 4

II

2 3 5 6 4
2 3 5 6 4 (n-3) swaps
2 3 4 5 6

III

2 3 5 6 4
2 3 4 5 6 (n-4) swaps

(n-1) items

TC → (n-1) + (n-2) + (n-3) 1
swaps

= n(n-1)/2

TC → ≈ $O(n^2)$

Time Complexity

Worst case → $O(n^2)$
Best case → $O(n^2)$ $O(n)$
Avg case → $O(n^2)$

Space Complexity

→ $O(1)$
↳ In-place sorting algo

Applications

① Simple & easy to understand
② complexity doesn't matter.