Course → Sorting & Searching

- a lot of algorithms to be discussed for sorting & searching
- focus more on problem solving, rather than syntactical sugar
- for each concept we will see different application

Sorting   searching

mix problem

→ for this course → codes & notes will be pushed on a new github link.

https://github.com/codechef-learn-competitive-programming/plus-course-content/tree/main/sorting_and_searching_in_python

link → codes + notes

Sorting $\rightarrow$ as arranging a set of elements in any one particular order or permutation.

$$[\; P_1 \quad P_2 \quad P_3 \; - - - - - - - - - \; P_n \;] \rightarrow \text{n players}$$

arrange the players based on their ranking.

$$[\; P_3 \quad P_1 \quad P_7 \; - - - - - - - \;]$$

one way of sorting

Q ⇒ You have a list of integers and you're supposed to sort them in ascending order of values

→ 2 1 0 3 -1 6

→ -1 0 1 2 3 6 → sorted form

Brute force → In order to sort a list based on any criteria, we can try all the permutations of list, & for every permutation, check if it is the desired result or not.

→ Sorting is one of the most relevant problems in computer science.

Application

→ Ecommerce

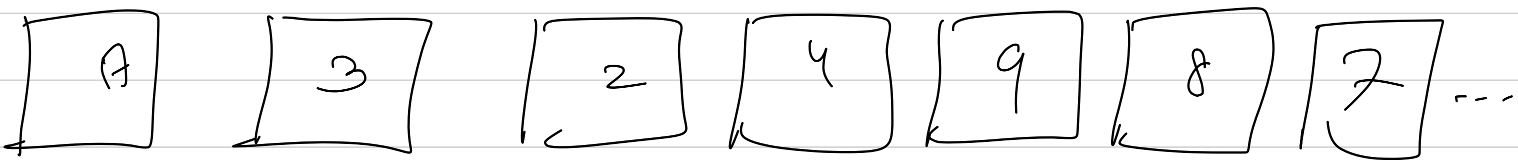
→ Ebooking

→ file management

etc

Criteria of sorting can be any comparable property of the given object.

↳ number of diamond



→ Sorting any object

When we prepare our algorithms for sorting the major things to keep in mind will be

- time complexity
- space complexity

Comparators → Assume list of dictionaries

we will discuss later

list ← [
    {
        'name' : 'macbook'
        'price' : 200000
        'company' : 'Apple'
    },
    {
    }
]

denotes a product

custom objects $\longrightarrow$ ex $\rightarrow$ products, persons, players etc, & you want to sort them based on any property, then we use the concept of comparators, where comparators are functions which compares two complex objects.

Q) What is the default value of reverse parameter in li.sort() function.

a) True

b) False ✓

reverse parameter → default parameter

~~reverse = True~~

asc

→ Type of Sorting algorithms

① Comparison Based → Ex → bubble, selection, insertion etc

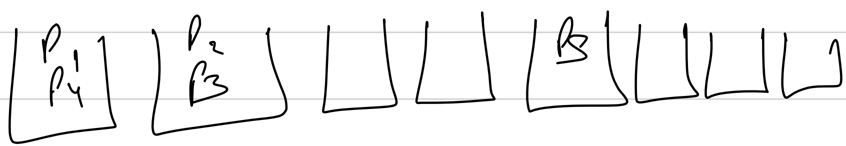
② Counting Based → Ex → radix, counting, bucket, etc

1 2 2 1 5

P_1 P_2 P_3 P_4 P_5

1 → 10

→ range



... certain range of value

P_1 P_4 P_2 P_3 P_5

⇒ While considering any sorting algorithms we need to look for certain criteria

- Space complexity
- Time complexity
- No. of comparisons
- No of swaps
- Inplace or not? → the same list is used for sorting
- Stability of the sorting algo

Stability → Stability means the relative ordering of 2 same elements is maintained. i.e. if 2 objects are equal according to the parameter of sorting then their relative order is maintained.

P_1 { Macbook pro, 200000, Apple } P_2 { asus rog, 200000, asus } P_3 = { macbook air, 100000, Apple }

original order
→ [P_3, P_1, P_2] → sort → based on price (asc) →

→ [P_1 P_2 P_3] → stable sort

[P_2, P_1, P_3] → unstable sort

Selection sort

→ It is a comparison based sorting algorithm

Sort based on ASC order

[5, 2, 6, 7, 2, 1, 0, 3]

all elements in sorted < y

this is the next spot to be handled → you want to bring the right candidate at this position.

0 1 2 3 4 5 6 7

[0, 1, 2, 2, 3, 7, 5, 6]

y

y → first element from unsorted part

find the min in the unsorted part, swap with y temp = 0

5, 2, 6, 7, 2, 1, 0, 3

0, 2, 6, 7, 2, 1, 5, 3

0, 1, 6, 7, 2, 2, 5, 3

0, 1, 2, 7, 6, 2, 5, 3

0, 1, 2, 2, 6, 7, 5, 3

0, 1, 2, 2, 3, 7, 5, 6

0, 1, 2, 2, 3, 5, 7, 6

0, 1, 2, 2, 3, 5, 6, 7 → since already sorted

$\rightarrow n + n-1 + n-2 \text{ - - - - - - - - - } 1$

$\rightarrow \dfrac{n \times (n+1)}{2} \Rightarrow \dfrac{n^2+n}{2} \longrightarrow \underline{\underline{O(n^2)}}$

Worst Case

$TC \rightarrow \underline{\underline{O(n^2)}}$

$SC \rightarrow O(1)$ Inplace → Yes

No. of swaps → $O(n)$ No. of comparisons → $O(n^2)$

Stability → Not stable

```python
def find_min_element(arr, start):
    """
    arr is the list
    start is the starting index of the uns
    """
    min_index = start

    start += 1

    while(start < len(arr)):
        if arr[start] < arr[min_index]:
            min_index = start

        start += 1

    return min_index
```

[5, 4, 3, 2]

min index = ~~0~~ ~~1~~ 2

start = ~~0~~ ~~1~~ ~~2~~ 3 . . . . . .

arr [1] < arr [0]

4 < 5 → yes

arr, 0

1 2 3 4

Best Case → TC → $\Omega(n^2)$

1 2 3 4 5

Avg Case $\Theta(n^2)$

Stability →

a) True → if stable

b) false → unstable

Not Stable

4 2 3 4' 1

1 2 3 4' 4

HW → Try to make selection sort stable??

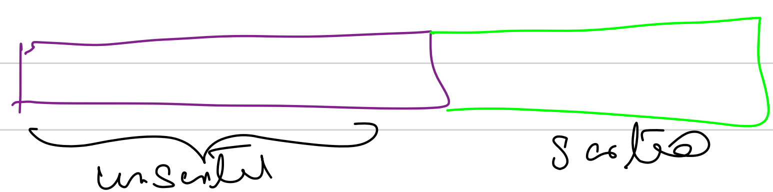
Bubble Sort → Comparison based sort.

after one iteration the biggest element of unsorted region is at the last (correct position)

14, 33, 27, 35, 10

14, 27, 33, 35, 10

14, 27, 33, 10, 35



→ 14, 33, 27, 35, 10

14, 33, 27, 35, 10 Swap

→ 14, 27, 33, 35, 10

→ 14, 27, 33, 35, 10 Swap

14, 27, 33, 10, 35

14, 27, 33, 10, 35

14, 27, 33, 10, 35 Swap

14, 27, 10, 33, 35

14, 27, 10, 33, 35 Swap

14, 10, 27, 33, 35 Swap

10, 14, 27, 33, 35

Sort

$n + (n-1) + n-2 + n-3 - - - - - - -$

$O(n^2)$

TC $\rightarrow$ $\Theta(n^2)$ $O(n^2)$ $\Omega(n)$

SC $\rightarrow$ $O(1)$

Inplace $\rightarrow$ Yes

No of comparison $\rightarrow$ $O(n^2)$

Swaps $\rightarrow$ $O(n^2)$

Stability $\rightarrow$ Yes Stable

5, 4, 3, 2, 1
4, 5, 3, 2, 1
4, 3, 5, 2, 1
4, 3, 2, 1, 5

$\rightarrow$ n swaps

3, 4, 2, 1, 5
3, 2, 4, 1, 5
3, 2, 1, 4, 5

n-1 swaps

2, 3, 1, 4, 5
2, 1, 3, 4, 5

$\rightarrow$ n-2 swaps

⋮

4 2 3 4' 1

2 4 3 4 1

2 3 4 4 1

2 3 4 4 1

2 3 4 1 4'

2 3 4 1 4'

2, 3, 1 4 4'

2, 1, 3 4 4'

1, 2, 3 4 4'

swapped = false

1 , 2 , 3 , 4 , 5

break