Sorting + Time Complexity

Agenda :

→ Time complexity
→ What is sorting ?
→ Why sorting ?
→ Bubble sort

Algorithm Analysis → space / memory
→ time / processing

Algorithm → logical steps to solve a problem

⇒ Now a days space is not much of a problem.

⇒ Time complexity is still a cause of concern for us.

⇒ GHz : 2.4 GHz , 2.8 GHz

★ 2.4 GHz : 2.4 × 1024 × 1024 op/ sec

★ 2.8 GHz : 2.8 × 1024 × 1024 ops/sec

★ Time Complexity : How much time does it takes your CPU to execute an algo .

i) Time taken by a cpu
ii) How many operatⁿ ✓✓

★ We will find time complexity of an algo in terms of no. of operatⁿ so that it becomes generic to all the computers.

⇒ We represent time complexity using : Big oh.

# $O(f(x))$ # time complexity of $f(x)$

↳ Notatⁿ for time complexity

```
for i in range(n) :          -> 3 times
    for j in range(n) :      -> 3 times
        print(1)
```

(n = 3)

=> 3 + 3 + 3 => 3×3

=> 4^2

=> 5^2

★ For very big values we can ignore constants

★ $C(foo()) = n \cancel{+2}$

★ time complexity = $O(n)$

$O(f(x))$

- Constant
- lower power polynomial
- factor

Ex : $\cancel{2}n^2 + \cancel{n} + \cancel{4}$ => Cost function

Ans : $O(n^2)$

Ex : 4 ⇒ cost functn

Ans : $O(1)$

Ex : $n^3 + 2n^2 + n + 4$ ⇒ cost function

Ans : $O(n^3)$

Quiz :

i)

```
ans = 0                              → 10
for i in range (1, 11) :
        for j in range ( 1, N+1) :
                ans += 1
```

Ans = 10 x N

Ans = $O(N)$

ii)

```
ans = 0                          → N
for i in range (0, N)                   → N
        for j in range (0, N) :
                ans += 1
```

Ans = N x N → N^2

Ans = $O(N^2)$

⭑ Sorting :

⇒ Arranging data in increasing or decreasing arrangement .

Amazon :

→ sort : price
→ sort : brands
→ Ranking system

Sorting Algo :

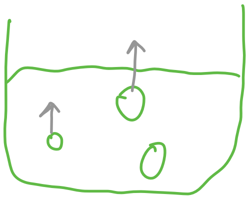
- → Bubble sort
- → Selectⁿ Sort
- → Insertion Sort
- → Merge Sort

⭑ Bubble Sort

heights = [5, 1, 2, 4, 7, 3]

goal = [1, 2, 3, 4, 5, 7]

Bubble sort :



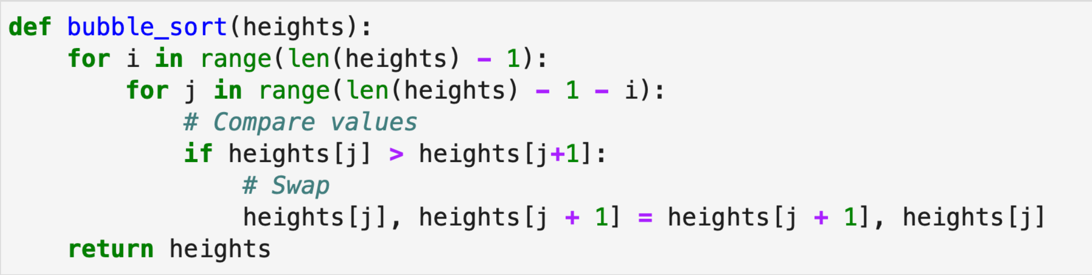
```
#   heights = [5, 1, 2, 4, 7, 3]
            j=0
# 1st pass
    -> if   heights [j]  >  heights [j+1]
                 # swap
                  0  1  2  3  4  5
        #        [1, 5, 2, 4, 7, 3]
                     j
    -> if   heights [j]  >  heights [j+1]
                 # swap
                  0  1  2  3  4  5
        #        [1, 2, 5, 4, 7, 3]
                        j
            heights [j]  >  heights [j+1]
                  0  1  2  3  4  5
        #        [1, 2, 4, 5, 7, 3]
                           j
                 no swap
                  0  1  2  3  4  5
        #        [1, 2, 4, 5, 7, 3]
                              j
                 swap
                  0  1  2  3  4  5
        #        [1, 2, 4, 5, 3, 7]
                                 j

#  2nd pass
        #        [1, 2, 4, 3, 5, 7]
```

3rd pass

[1, 2, 3, 4, 5, 7]

★ **Time Complexity Analysis :**



```python
def bubble_sort(heights):
    for i in range(len(heights) - 1):
        for j in range(len(heights) - 1 - i):
            # Compare values
            if heights[j] > heights[j+1]:
                # Swap
                heights[j], heights[j + 1] = heights[j + 1], heights[j]
    return heights
```

⇒ $n \;,\; n-1 \;,\; n-2 \;.\;.\;.\;.\; 1$

⇒ Cost ⇒ $n + n-1 + n-2 + \ldots\ldots + 1$

⇒ Cost $= \frac{n(n+1)}{2}$

$= (n^2+n)/2$

$= \frac{n^2}{2} + \frac{n}{2}$

Time complexity : $\boxed{O(n^2)}$

$100 + 99 + 98 + 97 - - - 1$

$1 + 2 + \ldots\ldots 100$

⇒ $\frac{n(n+1)}{2}$ ⇒ $\frac{100(101)}{2}$ ⇒ $50 \times (101)$

רב סוף סוף