Sorting and Time Complexity - 2

Starting in 2 mins 9.02 pm

Problem Solving - 1 on Sat, 10th
Problem Solving - 2 on Sat, 17th
} optional but recorded

Ayush
=> Elon Algo. (A)

Manju
=> Perfect Algo (B)

Task: Given a list containing N integers, sort the list in ascending order.

Inp : [1, 7, -1, 2, 4, 6]

Output: [-1, 1, 2, 4, 6, 7]

$N = 10^5$ 100000

I/p => [5, 1, 3, 3, 7, 9, 10, ... 100]

Execution Time (i)

15 sec.

Windows 95

10 sec

Macbook Pro.

Dependency on
- device used.
- OS

→ Computer can roughly perform 10^8 operations in 1 sec.

2 GHz

2.5 GHz

2.9 GHz

same Macbook pro (ii)

7 sec

C++ faster

10 sec.

Python. slower

dependency on
Software
↓
Lang. used.

same language and same macbook pro. iii)

12 sec

Top of volcano.

10 sec.

Canada

Cooler the better

dep^n on environment

same environ in Canada. 5 sec.

10 sec. Canada.

Execution time depends on lot of factors

↳ not a good measure

Good measure ⇒ no. of iterations / operations.

Code 1

```
def foo(n):
    a = n
    b = 2 + 3
    c = a * b
    print(n)
```

Constant no. of iterations

a = n → 1st operation

b = 2 + 3 → 2nd operation

c = a * b → 3rd operation

print(n) → 4th operation.

same time per operation.

⇒ 4 operations.

may / may not depend on HW/SW.

independent of HW/SW/temp.

will be given only once.

⇒ C constant no of operations.

C = 4

Code 2.

```
def foo(n):
    x = 1
    y = 2
    i = 0
    while i < n:
        print(i)
        i += 1
```

x = 1 → 1

y = 2 → 2

i = 0, 1, 2, ..., (n-1)

n iterations.

3n + 2

⇒ Time function wrt input n.

T(n) = 3n + 4

n ↑ T(n) ↑

Total operations = $3n + 4$

$y = mx + C$

$T = 3n + C$ — $C = 4$.

```
i = 0                          → 1
while i < n :                  → n times.   +1
    print(n)                   → n times
    i += 1                     → n times.
```

n times

$3n + 2$

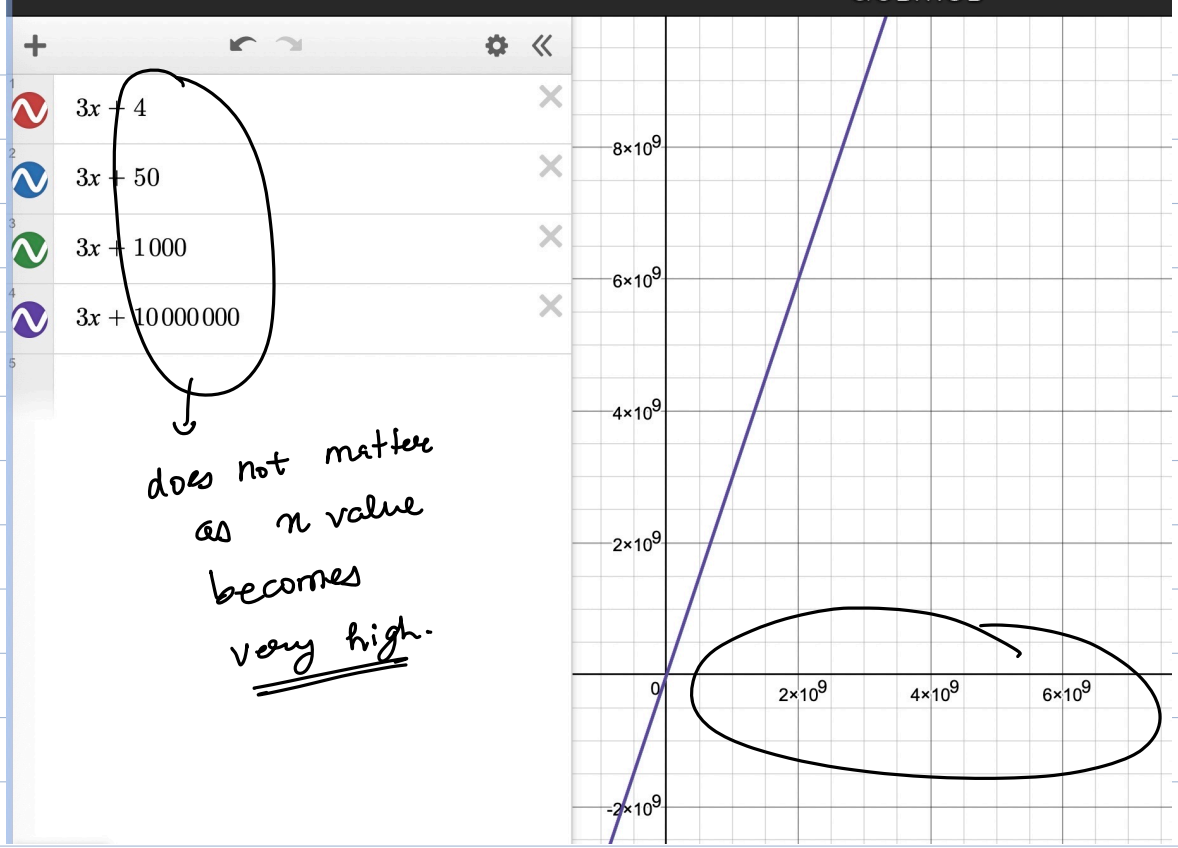
n = 2.

```
i = 0
while i < 2:
    print(i)
    i += 1
```

	i	i<2
0	0	✓
1	1	✓
	2	✗

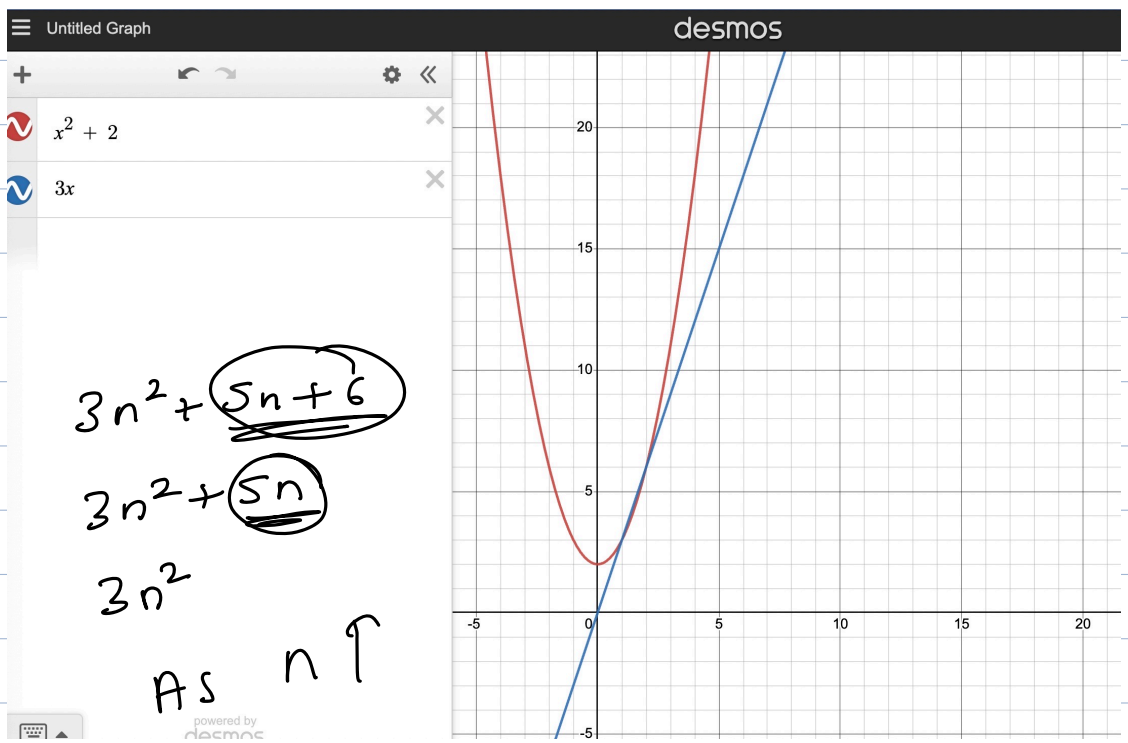
```
i = 0                                  → 1.
sum = 0                                → 1.
while (i < N) {                        → N + 1.
    sum = sum + number[i]              → N
    i += 1                             → N
}
```

$3N + 3$



does not matter as n value becomes very high.

Big-O Notation



$3n^2 + 5n + 6$

$3n^2 + 5n$

$3n^2$

As n ↑

n is the input

Code 3

```
ans=0
i=0
while(i<n):
    j=0
    while(j<n):
        print(i+j)
        j+=1
    i+=1
```

ans=0 → C
i=0 → C
while(i<n): → n
j=0 → n
while(j<n): → n^2
print(i+j) → n^2
j+=1 → n^2
i+=1 → n

$3n^2 + 3n + C$

$\Rightarrow c_1 n^2 + c_2 n + c$

has higher growth rate

vs

$\Rightarrow an + b$

As N↑

→ Observation 1 : Constant does not matter

→ Observation 2 : Lower order terms of N do not matter. Only the highest degree term matters.

As $N \to \infty$ (very very large values)

↓

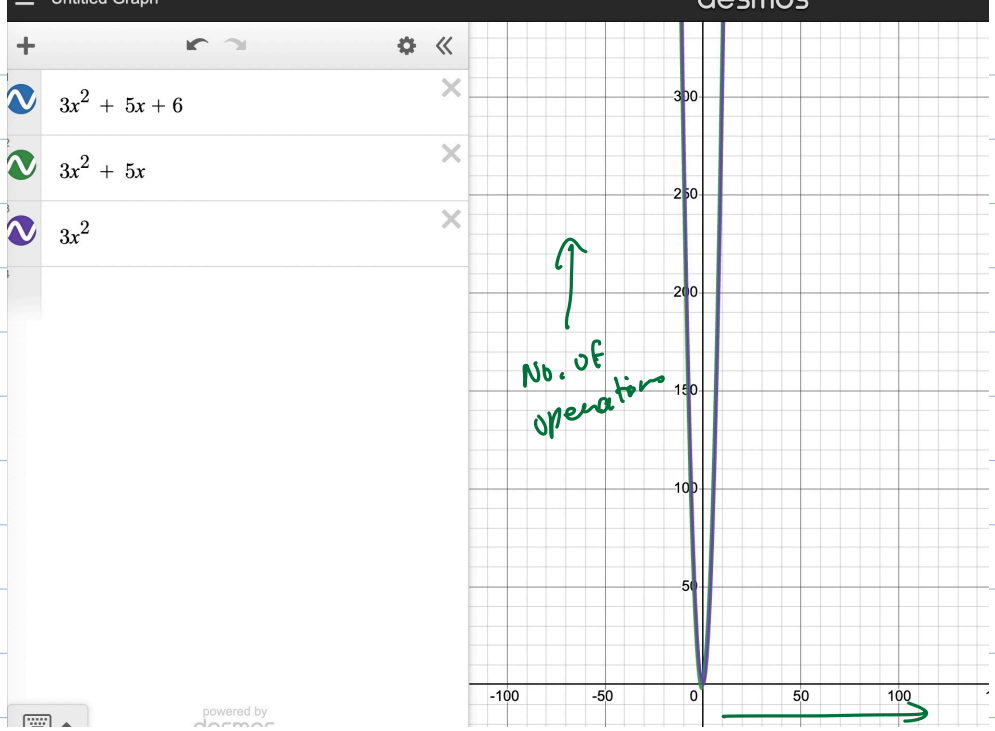
Asymptotic Analysis

⇒ Big-O notation (mathematical notation) Standard notation

↓

$f(N) = 3N + 4$

rate of growth of time wrt input.



$T(N) = 3N + 4.$

Find Big-O for $T(N)$

3N
4N
5N

1 → $O(T(N)) = 3N$

Drop constant

$O(T(N)) = N$

$3N^2$ is order N^2

order N

2 → Take highest power of N.

$T(N)$ is $O(N)$

order of N

$T(N) = 3N^2 + 3N + C$ is $O(N^2)$

1 → $N^2 + N$

Drop constant

2 N^2

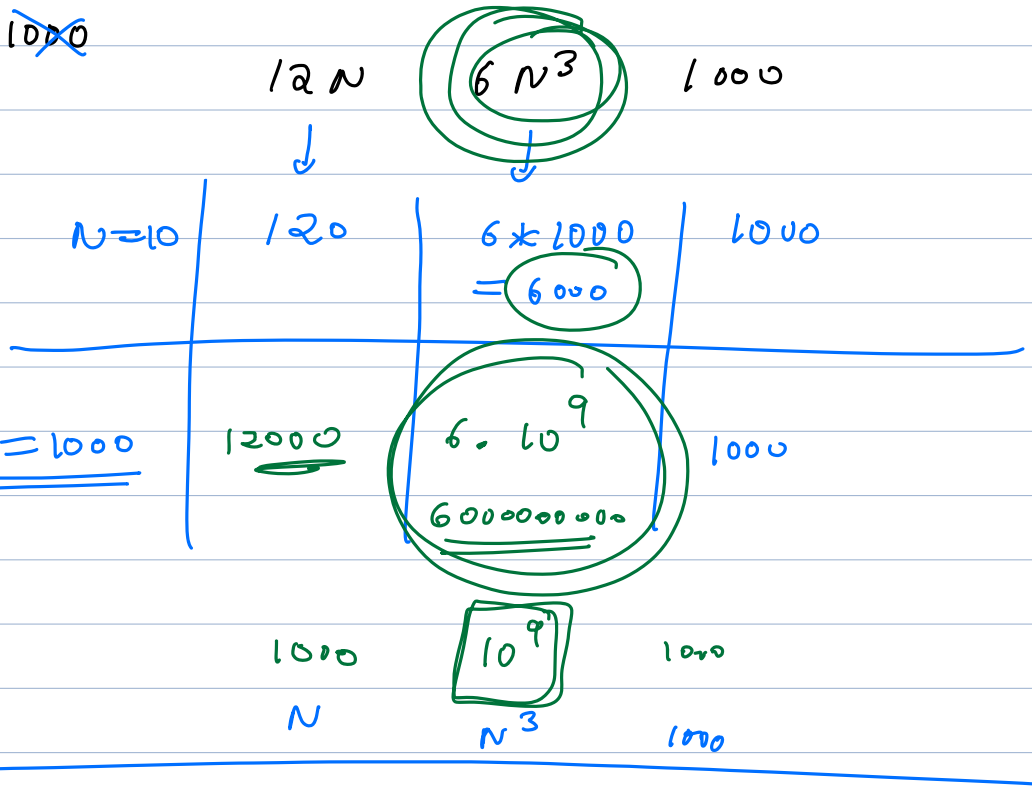
Drop lower power of N

$N = 10$

~~12~~ $N + ~~6~~ N^3 + ~~1000~~$

↳ $N + N^3$

↳ $O(N^3)$



Linear search

① Iterations

```
def search(nums, target):
    i = 0
    while i < len(nums):
        if (nums[i] == target)
            return i
        i = i + 1
    return None
```

$aN + b$

		Iterations
Best Case.	[1, 5, 7, 2, 3], target = 1	1
Worst Case	[1, 5, 7, 2, 3], target = 20	N

Worst-case by default. $O(N)$

Space Complexity

```
def search(nums, target):
    i = 0
    while i < len(nums):
        if (nums[i] == target)
            return i
        i = i+1
    return None
```

Variables

- nums — Input
- target — Input
- i → Created by me. (Non-input)
- C ⇒ Constant
- O(1) → does not depend on input size!

Auxiliary / Non-input space required.
(Extra)

N is size of arr.

```
def test(arr):
    res = [i*i for i in arr]
    return res
```

Space

arr (input)

res ⇒ N int
i ⇒ 1 int.
(N+1) int.

arr = [1, 2, 3]
res = [1, 4, 9]

arr = [1, 2, 3, 4, 5, 6]
res = [1, 4, 9, 16, 25, 36]

```
def test(arr):
    res = []
    i = 0
    while i < len(arr):
        res.append(arr[i]*arr[i])
        i+=1
```

N

Non-input space

(i) ⇒ 1
res ⇒ N
N+1

Space Complexity : O(N) Linear

Time Complexity : O(N)

Quiz 2

list = [1, 3, 5, 7, 9]

```
GetEvens(list) {
   i = 0
   evensList = Create empty list
   while (i < len(list)) {
      if (list[i] % 2 == 0)
         Add list[i] to evensList
      i = i + 1
   }
   return evensList
}
```

evenslist = [ ]

S(N) = k

O(1)

What is the best case auxiliary space complexity?

93 users have participated

✓ A	S(N) = k (k is constant)	61%
B	S(N) = N + k (N is size of the list, k is constant)	39%

[2, 4, 6, 8, 10, 12]

Worst-case

evenslist = [2, 4, 6, 8, 10, 12]

S(N) = N + k.

Linear search has best case for N = 0 True / False

Best case is O(1) independent of N value.

Worst case is O(N).

What is the time complexity of the below code:

```
FindMin(x, y) {
   if (x < y) {
      return x
   }
   else {
      return y
   }
}
```

Chose the correct answer

A	O(N)
B	O(1)

FindMin(3, 5)

TC : O(1)

SC : O(1)

independent of input value.

```
def foo():
    a = 1
    b = 2
    c = 3
    d = 4
    e = 5
```

SC: O(1)

O(5)

$\cancel{5} * 1$

1, N, N^2, logN, ——— ——— ———

Mid-Module Test

Live from next week, Wed (14th) until next to next week Tue. (20th)

Fail => Repeat the 1 month Intermediate

Pass → Continue.

Will cover everything covered till Wed, 14th Class in Intermediate module including Python Refresher.