Time Complexity - II

Today's Content

→ Comparing two Algos
- → Using exectution time
- → Using iteration & graphs

→ Why Big O is needed?
- → Why lower order terms are ignored?
- → Why const. coffiecients terms are neglected?
- → Issues in Big O
- → Worst Case

→ Space Complexity

→ TLE (Time limit Exceeded error)
- → Why TLE occurs?
- → How to approach any given problem?
- → Importance of constraints

Comparing Algos using execution time

Given N elements, sort them in increasing order.

$N = 10^4$ (input size)

Algo1 (Dilip)

Exec. time

15 sec

↓

(Windox XP)

↓

(Macbook M_2)

↓

7 sec

↓

(C++)

↓

7 sec

(Top of hot volcano)

↓

Mt. Everset

↓

5 sce

Algo2 (Hrishikesh)

10 sec

↓

(Macbook M_2)

↓

10 sec

↓

(Python)

↓

(C++)

5 sec

Mt. Everest

↓

5 sec

Exection time : Since it depends on so many external factors, we generally don't compare 2 algos using execution time.

Comparing using Iterations & Graphs

Algo1 (Prasant)

Algo 2 (Sai)

iterations : $100 \log_2(N)$ $\quad N/10$

for $N=10$

$$\boxed{\begin{array}{lcl} 100\log(10) & \longrightarrow & \sim 300 \\ 10/10 & \longrightarrow & 1 \end{array}} \Rightarrow$$ Algo 2 is better

for $N <= 3900$ Algo 2 is better

for $N > 3900$ Algo 1 is better

Google results : 1M +

Baby shark : 12B + views

Compare 100+ Algos ?

Algo1 Algo2 Algo3 . . . . . .

Comparing all using graph method is tideous & time taking.

Asymptotic analysis of Algorithms

↳ Performance analysis of Algo(s) for very large inputs.

Use Big O notation

1. Calculate iterations
2. Take highest order term
3. Ignore const. coefficient

Why neglect lower-order terms?

N^2+10N

input size	total iterations	% of lower order terms
$N=10$	200	$\frac{10N}{N^2+10N} = \frac{100}{200} = 50\%$
$N=100$	10^4+10^3	$\frac{10^3}{10^4+10^3} = \frac{1}{11} \sim 9\%$
$N=10^3$	10^6+10^4	$\frac{10^4}{10^6+10^4} = \frac{1}{101} \quad <1\%$
$N=10^6$		Highly insignificant

Why ignore const. coefficients

$10N$

$N=10^5$

$N \rightarrow 10^5$

$10N \rightarrow 10^6$

N^2

10^{10}

Claim 1 : for all inputs, we can decide which Algo is better. ✗

Claim 2 : for all inputs $>= x$, we can decide which Algo is better. ✓

final Claim : When we compare 2 Algos using Big O, Algo1 will always be better than Algo2 for all input values above a certain point. ↖ threshold

→ After threshold, Big O holds.

→ Please don't worry about threshold.

Issues in Big O

iteration : $2N^2+4N$

$O(N^2)$

$3N^2$

$O(N^2)$

$2N^2+4N-2N^2$

$3N^2-2N^2$

$4N$ is always better than N^2

→ If we have same Big O for 2 Algos, then Big O will fail.

↳ It can't tell which is better.

Worst Case

Ques : search of an element = K

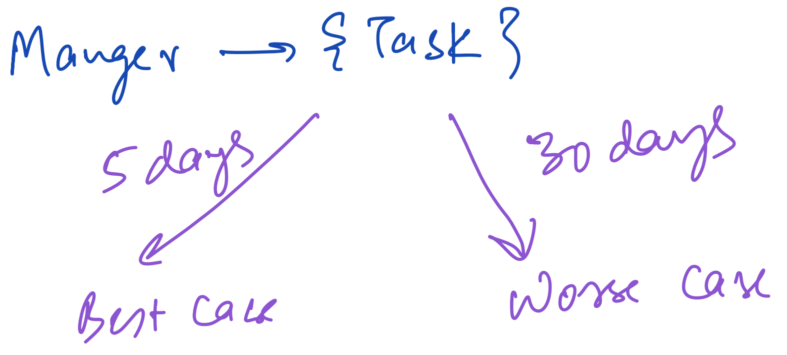
```
bool search( arr[], K) {
    for (i=0; i< N → n = array size ; ++i) {
        if (a[i] == K)
            return true

    }
    retur false
}
```

total iterations = N
O(N)

best-case scenario iteration = 1

worst-case scenario iteration = N



Manger → {Task}

5 days

Best Case

30 days

Worse Case

BREAK : 9:30 - 9:40 PM

Space complexity

Code → Time complexity → no. of iterations
↳ Big O notation

Code → Space complexity
↳ total space → Big O notation

```
def func(N) {
    int  x=0
    int  y=N
    long p=5
}
```

int → 4 Bytes
long → 8 Bytes

total = 16 Bytes → O(1)

```
def func (int a[]) {
    int m = a[0]
    for(i=1; i< a.size(); ++i ) {
        m= max( m ,a[i])
    }
    return m
}
```

m → 4B
i → 4B
a[] → 4×n B

↗ no. of element in a

total Space =

$8 + \cancel{4n}$ Bytes

↓

this is not created by the function

total space = 8B

= O(1)

```
def func ( int a[] , int n ) {
    int pf[n]  ────────────→  4n B
    pf[0] = a[0]
   for ( i=1; i<n; ++i ){       i → 4B
       pf[i] = a[i] + pf[i-1]
   }
}
```

total space =
4n+4
$O(n)$

```
for ( i=0; i<n; ++i ) {
      int x;
      . . . .
   }
```

⇒ $O(1)$

~~$O(N)$~~

Time Limit Exceeded - TLE

Prashant → (Amazon) → Hiring Challenge → 3Q (1.5 hrs)

optimize

Q → idea → code → submit → TLE

→ better idea → code → submit → TLE

Online Editors/Compilers → 1 GHz → 10^9 instruction/sec

```
s=0  +1
for(i=0; i<n; ++i){
    +1   +1×n  +1×n
    s=s+i
        +1×n
}
```

total instruction = $2+3N$

total iterations = N

Approx :

1 iteration → 10 instructions

Approx2 :

1 iteration → 100 instructions

1 sec → 10^9 instruction

1sec → $[10^7 - 10^8]$ iterations

Importance of Constraints

$1 <= N <= 10^6$

int a[n][n] → $O(10^{12})$
MLE space

Algo → $O(N^2)$

iterations = $(10^6)^2 = 10^{12}$ iterations TLE

Algo → $O(N \log N)$

iteration $= 10^6 \log_2 10^6$

$$\boxed{10^3 \approx 2^{10}}$$

$= 10^6 \log_2 2^{20}$ ✓

$= 10^6 \times 20 = 2 \times 10^7$ iterations

$1 <= N <= 100$

Algo → $O(N^3)$ iteration $(100)^3 = 10^6$ ✓

$$\boxed{\text{no need to optimize further}}$$