05/12/2021

Time and Space Complexity

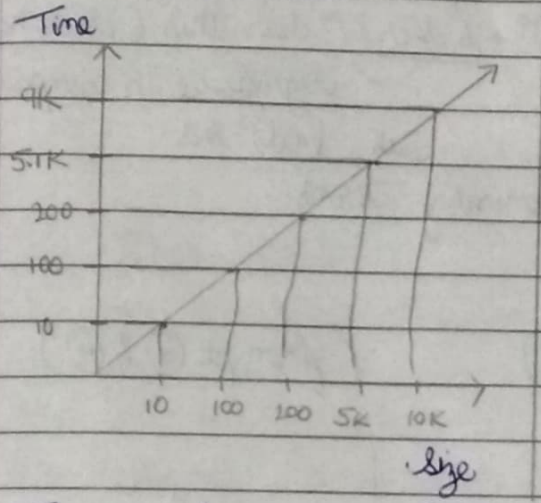
Time Complexity

Old Computer	M1 Macbook (very fast)
data : 1 million elements in an array	data and algorithm are kept same for this also
Algorithm :- Linear search for an element that does not exist	
Time Taken - 10 secs	Time Taken :- 1 secs

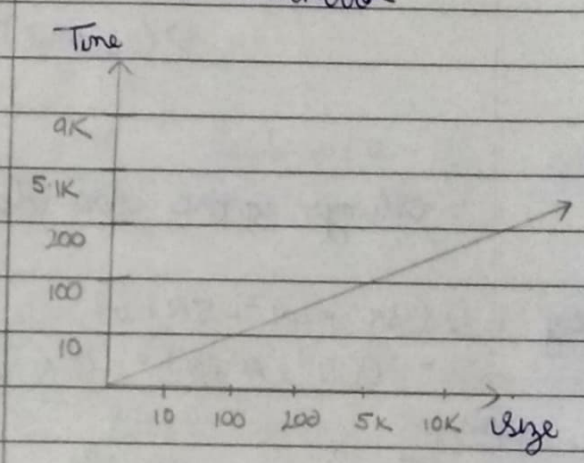
Both machines have same time complexity as

Time Complexity != Total time taken to execute

Old Machine

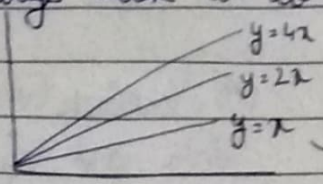


M1 Macbook



⇒ Time complexity is a function that gives us the relationship about how the time will grow as the input grows.
In the above case of old computer & M1 macbook time was growing linearly as input was growing

⇒ What do we consider when thinking about complexity??

1) Always look for worst case complexity?
2) Always look at complexity for large / infinite data
3) 

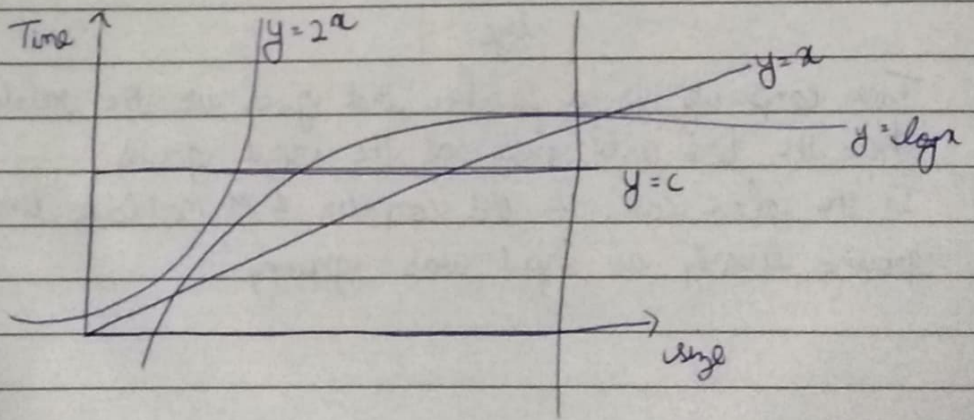
all are taking linear time (n).

Even the value of actual time different they are all growing linearly.
We don't care about actual time as it will be different for different machines.
This is why we ignore all constants

4) $O(N^3 + \log(N))$
from pt ② ⇒ N = 1mil
⇒ $((1mil)^3 + \log(1mil))$
⇒ $((1mil)^3 + 6\ secs)$ ↗ does this 6 secs has any significance in comparison with $(mil)^3$ secs.

Always ignore less dominating terms

e.g $O(3N^3 + 4N^2 + 5N + 6)$
$= O(N^3 + N^2 + N) = \underline{O(N^3)}$ (from pt ③ & ④)



$O(1) < O(\log(N)) < O(N) < O(2^N)$
↳ constant

⇒ Big O Notation

If you are said that time complexity is $O(N^3)$, then it means that the upper bound of your algorithm is N^3 you algorithm might take N^2, $N \log N$ or any other time but it will never exceed N^3 time.

Mathematically

$$f(N) = O(g(N))$$

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$$ means it should be a finite answer

eg $O(N^3) = O(6N^3 + 3N + 5)$

$g(N)$ $f(N)$

$$\lim_{N \to \infty} \frac{6N^3 + 3N + 5}{N^3}$$

$$\lim_{N \to \infty} \frac{6 + \frac{3}{N^2} + \frac{5}{N^3}}{} = 6 + \frac{3}{\infty} + \frac{5}{\infty}$$

$= 6 < \infty$ finite value

⇒ Big Omega Notation

If you are said that time complexity is $\Omega(N^3)$, then it means the lower bound of your algorithm is N^3, it will atleast take N^3 time complexity it can take N^4, N^5 or any other time complexity but it never be less than N^3.

Mathematically

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} > 0$$

⇒ Theta Notation

It simply means combining both upper and lower bound.

Mathematically

$$0 < \lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$$

⇒ Little O Notations

This is also giving upper bound but loose upper bound not strict

Big O	little o
	(stronger statement)
$f = O(g)$	$f = o(g)$
$f \leq g$	$f < g$ (strictly)

Mathematically

$$\lim_{n \to \infty} \frac{f(N)}{g(N)} = 0$$

eg $f = N^2 \quad g = N^3$

$$\lim_{N \to \infty} \frac{N^2}{N^3} = \lim_{N \to \infty} \frac{1}{N} = 0$$

⇒ Little omega Notation

Loose lower bound

Big Ω	little ω
$f = \Omega(g)$	$f = \omega(g)$
$f \geq g$	$f > g$

Mathematically

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} = \infty$$

eg $$\lim_{N \to \infty} \frac{N^3}{N^2} = \lim_{N \to \infty} N = \infty$$

Space Complexity

Space Complexity = Input Space + Auxillary Space

Auxillary space is the extra space or temporary space used by an algorithm.

Q)
```
for (i=1; i<=N; ) {
    for (j=1; j<=K; j++) {
        // some operation that take time t
    }
    i = i+K
}
```
find time complexity

(Sol^n) Inner loop is running K times and performing t work in each iteration.

$O(Kt)$ time for Inner loop

Time Complexity of Outer loop = $O(xt$ * times outer loop is running$)$

$i = 1, 1+K, 1+2K, 1+3K, 1+4K \ldots 1+xK$

$1+xK \leq N$

$$x = \frac{N-1}{K}$$

$x \rightarrow$ No of times outer loop is running

$$TC = O\left(Kt * \frac{N-1}{K}\right) = O(t*(N-1))$$

$\therefore TC = O(t*N)$ (ignoring the constant)

Q) Some common loops

1)
```
for (int i=0; i<n; i=i+c) {
    // some worst work
}
```

$n = 10, c = 2$ $\quad TC = O(n/c)$

$0, 2, 4, 6, 8$ $\quad = O(n)$

$\frac{10}{2} = 5$

2) for(i=n, i>0; i=i-c){
// some const work
}

n=10 c=2	n=11, c=2
i=10, 8, 6, 4, 2	i=11, 9, 7, 5, 3, 1

∴ TC = O(n/c + 1)
TC = O(n)

3) for(i=1; i<n; i=i*c){
O(1) work
}

n=32, c=2	n=33, c=2
i=1, 2, 4, 8, 16	i=1, 2, 4, 8, 16, 32

$1, c, c^2, c^3 \dots c^{k-1}$

$c^{k-1} < n$

$k-1 < \log n$

$k < \log n + 1$

TC = O(log n)

4) for(i=n, i>1, i=i/c){
O(1) work
}

n=32, c=2	TC = O(log(n))
32, 16, 8, 4, 2	

5) for(i=2; i<n, i=Math.pow(i,c)){
O(1) work
}

$2, 2^c, (2^c)^c, \dots c^{k-1}$

$2, 2^c, 2^{c^2}, \dots 2^{c^{k-1}}$

$2^{c^{k-1}} < n$

$c^{k-1} < \log_2 n \Rightarrow k-1 < \log_c \log_2 n$

$k < \log_c \log_2 n + 1$

∴ TC = O(log log(n)).
2

5)
```
for (i=0; i<n; i++) {
    for(j=1; j<n; j=j*2){
        O(1) work
    }
}
```
TC = O(n * log n)
TC = O(n log n)

Space Complexity

1)
```
int arrSum (int arr[], int n){
    int sum = 0;
    int ~~a~~
    for (int i=0; i<n; i++) {
    sum += arr[i];
    }
}
```
Auxillary space = O(1)
I/p space = O(n)
Space complexity = O(n) + O(1) = O(n)

2)
```
int fib (int n){
    int f[n+1];
    f[0] = 0
    f[1] = 1
    for (int i=2; i<=n; i++)
        f[i] = f[i-1] + f[i-2];
return f[n];
}
```
Auxillary space = O(n)
I/p space = O(1)
SC = O(n) + O(1) = O(n)

3)
```
int fib(int n){
  if (n==0 || n==1)
        return n;
  int a=0, b=1;
  for (int i=2, i<=n; i++) {
      c=a+b;
      a=b
      b=c
  }
return c;}
```
Auxillary space = O(1)
I/p space = O(1)
SC = O(1)