DAY 5

Page No. 28
Date 07/09/2022

TIME COMPLEXITY

Time and Space complexities talk about how efficiently we can convert an input into output.

Time Complexity :- Order of time taken by an algorithm to run as a function of the size of input.

Q Check if a given ~~prime~~ input N is prime

A no. with only 2 factors → 1 & N

-1 has exactly 2 factors. But it is not a prime number.

Reason → Prime numbers are always +ve

Brute force

```
factors = 0;
for (int i=1; i<=N; i++) {
    if (N % i == 0)
        factors++;
}
return (factors == 2);
```

For N = 10 (or any value) it takes N steps. Let's try & optimize this.

Optimized approach :- Only look until the factors < N run out.

```
factors = 0;
for (int i=1; i*i<=N; i++) {
    if (N % i == 0) {
        factors++;
    }
}
return (factors == 1);
```

More elegant way :-

Consider N=10

a	b
1	10
2	5

$\Rightarrow$ Factors of 10 = {1, 2, 5, 10}

Here, $a * b = N \Rightarrow b = \frac{N}{a}$

Also,

$a <= b$ always

$\Rightarrow a <= \frac{N}{a}$

$\Rightarrow a^2 <= N \Rightarrow a <= \sqrt{N}$

So,

```
for(int i=2; i<=√N; i++){
    if (N%i == 0)
        return false;
}
return true;
```

We essentially reduced the TC from O(N) to $O(\sqrt{N})$

SC stayed the same O(1).

- Consider two Algos A1 & A2

	A1 (2^N)	A2 (N^3)
N=2	4	8
N=5	32	125
N=10	1024	1000
N=1000	10^{30}	10^6

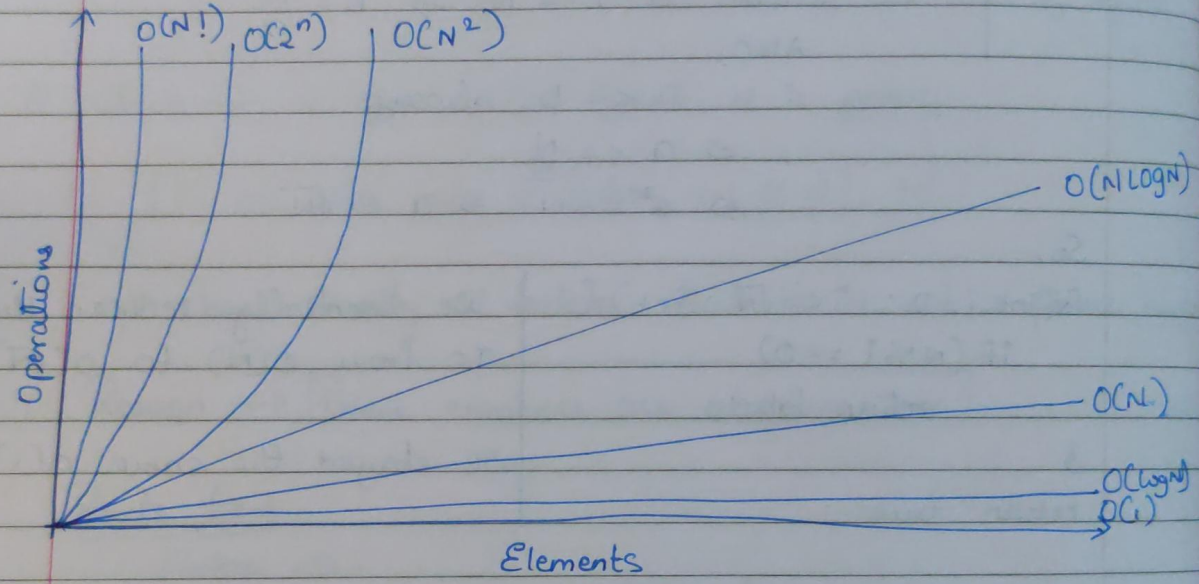
- As N grows, the rate at which the time taken grows is lesser with A2. So we choose it over A1.

Examples for Big-Oh

$f(N) = 2N^2 + N - 2 \Rightarrow O(N^2)$

$f(N) = 2 - 3N + \sqrt{N} \Rightarrow O(N)$

$f(N) = N^{1.1} + 2 + 3N - \sqrt{N} \Rightarrow O(N^{1.1})$



Quiz

1)
```
foo(int i=0; i<=N ; i+=2){
    print(i);
}
```

$O(N)$

2)
```
foo(int i=0; i<N ; i++){
    for(int j=i+1; j<N; j++){
        ans += i*j;
    }
}
```

$i=0 \Rightarrow j = 1 \text{ —— } N$

$i=1 \Rightarrow j = 2 \text{ —— } N$

$i=2 \Rightarrow j = 3 \text{ —— } N$

$= N + (N-1) + (N-2) + (N-3) + \cdots 1$

$= \Sigma N = \frac{N(N+1)}{2} = \frac{N^2+N}{2}$

$\Rightarrow O(N^2)$

3)
```
j = 0;
for (i = 0; i < N; i++) {
    while (j < N && a[i] <= a[j]) {
        j++;
    }
    sum += a[j];
}
```
O(N)

4)
```
i = N; a = 0;
while (i > 0) {
    a += i;
    i /= 2;
}
```
log(N)

$N \to N/2 \to N/2^2 \to N/2^3 \to N/2^4 \to N/2^k = 1 \to 0$

After 2^k steps, N becomes 1.

$\Rightarrow \frac{N}{2^k} = 1 \Rightarrow N = 2^k \Rightarrow \log(N) = \log(2^k)$

$\Rightarrow k = \log_2 N \Rightarrow O(\log N)$

Note :- Why don't we consider bases in logs while writing time complexity ?

$\log_a b = \frac{\log(b)}{\log(a)}$. Consider $\log_2 N \Rightarrow \frac{\log N}{\log 2}$

Constants and multipliers are to be ignored.

5) `for(int i=1; i<N; i=i*2) { print(i); }` | O(log N)

$i = 1 \to 2 \to 4 \to 8 \to \dots \to N$

$\Rightarrow i = 1 \to 1\times2^1 \to 1\times2^2 \to 1\times2^3 \to \dots \to 1\times2^k \Rightarrow N = 2^k \Rightarrow k = \log_2 N$

***
6) k = 0;
for (i=0; k<=N; i++) {
    k = k+i;
}

i	k
0	0
1	1
2	3
3	6
4	10
5	15
6	21

N = 20

Increments of k :-

$0 + 1 + 2 + 3 + \dots + x <= N$

$\Sigma x <= N$

$\frac{x(x+1)}{2} <= N$

$x^2 + x <= 2N$

$x^2 + x - 2N = 0$

$\frac{-1 \pm \sqrt{1+8N}}{2} = 0$ $\left[\frac{-b \pm \sqrt{b^2-4ac}}{2a}\right]$

$\pm \sqrt{1+8N} = 0+1$

$\pm \sqrt{1+8N} = 1$

$\Rightarrow O(\sqrt{N})$

SPACE COMPLEXITY

- Extra space taken by an algo to convert given i/p to the desired o/p as a fn.
- ~~Directly proportional to the size of i/p~~
- Fn of i/p of the size.

Q For a given i/p array, find the product of freq of every element in the array. $|A| = N <= 10^5$
$A[i] = x <= 10^5$

Eg:
A = [3 2 1 1 3 5 5 5]

A[i]	→ freq
3	2
2	1
1	2
5	3

$2 \times 1 \times 2 \times 3 = 12$

Sol Create a frequency based array. $|F| = x$

```
∀ i  F[i] = 0
for (i=0; i<N; i++) {
        F[A[i]]++;
}
ans = 1;
for (i=0; i<x; i++) {
    if (F[i] == 0) continue;
    ans *= F[i];
}
return ans;
```

Space Complexity (SC)

O(x) → biggest element in the array.

Q Given an i/p array, return the frequency of the elements.

Sol We can use the same method in the prev question but the SC will become O(1).
Since we do not consider i/p & o/p memory as EXTRA space. X is passed as an i/p as well.

Data Types and Sizes in Java

DataType	Size
Boolean	1 bit
char	2 byte
short	2 byte
int	4 byte
long	8 byte
float	4 byte
double	8 byte

Note:- Arrays always go for continous memory allocation. The sizes of arrays are always fixed. However, these are extremely quick. We can access elements in const time.

Location = BaseAddressOfTheArray + (index * SizeOfDataType)

Assignment

Q6. Arrange the following complexities in the increasing order

a) 2^n b) $n^{3/2}$ c) $n\log n$ d) $n^{\log n}$

Sol

$$n\log n < n^{3/2} < n^{\log n} < 2^n$$

Homework

Q2. What is the time complexity of the following code?

```
int i, j, k = 0;
for (i = n/2 ; i<=n; i++) {
    for(j = 2; j<=n ; j++) {
        k = k+n/2;
    }
}
```

Sol The first loop is running $n/2$ times $\longrightarrow n$
The second inner loop goes for $\log n$ times.
So the answer is $O(n \log n)$

Q5.

```
count = 0
i = N
while i > 0:
    for j in range(i):
        count += 1;
    i = i //2
```

Sol

i	# steps j takes
N	N
N/2	N/2
N/4	N/4
N/8	N/8
⋮	1
⋮	1
⋮	⋮
1	1

→ Here, the TC = # steps done. ie.,
Σ (# steps j takes)
$= N + N/2 + N/4 + N/8 + \cdots + \frac{N}{2^k}$

→ If k is the total no of steps it takes for i to reach 1,
then $\frac{N}{2^k} = 1 \Rightarrow 2^k = N \Rightarrow k = \log_2 N$

→ Now, for the summation,
$\Sigma(\#) = N\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^k}\right)$

→ This is G.P. $\Sigma(GP) = \frac{a(1-r^n)}{(1-r)}$

$$\rightarrow E = N\left(\frac{1(1-(1/2)^{\log_2 N}}{1-1/2}\right) = N\left(\frac{1-(\frac{1}{2})^{\log_2 N}}{1/2}\right)$$

$$= N\left[2 - 2\left(\frac{1}{2}\right)^{\log_2 N}\right] = 2N\left[1 - \frac{1}{2^{\log_2 N}}\right]$$

$$= 2N\left[1 - \frac{1}{2^{\log_2 N}}\right]$$

$$= 2N\left[1 - \frac{1}{N}\right] \qquad \left[\because 2^{\log_2 N} = N\right]$$

$$= 2N - 2 \approx O(2N) \approx O(N)$$