Complexity Analysis Problem Set Solutions

Find the Time Complexities of the following snippets of code:

(A)

```
/* Assume that rand() takes constant amount of time */
int a = 0, b = 0;
for (int i = 0; i < N; i++) { // Loop-1
    a = a + rand();
}
for (int j = 0; j < M; ++j) { // Loop-2
    b = b + rand();
}
```

Solution: `Loop-1` runs `N` times (as `'i'` goes from `0` to `N-1`), and `Loop-2` runs `M` times (as `'j'` goes from `0` to `M-1`). Now, both `Loop-1` and `Loop-2` run independently, on their own. So, we just simply add the number of times each loop (`Loop-1` & `Loop-2`) runs.

$\therefore$ Time Complexity is $O(N + M)$ - Linear in Nature.

(B)

```
int a = 0, b = 0;
for (int i = 0; i < N; i++) { // Loop-1
    for (int j = 0; j < N; j++) { // Loop-1.1
        a = a + j;
    }
}
for (int k = 0; k < N; k++) { // Loop-2
    b = b + k;
}
```

Solution: `Loop-1` is independent in nature and runs `N` times, and `Loop-1.1` runs `N` times too. But, for each loop of `Loop-1`, `Loop-1.1` runs `N` times. $\therefore$ `Loop-1` and `Loop-1.1` combinedly run for `N × N` times, *i.e.,* `Loop-1` and `Loop-1.1` run for N^2 times. And now, from `Loop-2` we can see that it individually runs for `N` times. So, `Loop-2` runs for `N` times.

$\therefore$ Time Complexity is $(N^2 + N)$, which is $O(N^2 + N)$ - Quadratic in Nature.

(C)

```
int a = 0;
for (int i = 0; i < N; i++) { // Loop-1
    for (int j = N; j > i; --j) { // Loop-1.1
        a = a + i + j;
    }
}
```

Solution: **`Loop-1`** is an independent loop which is linear in nature, *i.e.,* it runs N times (as **`'i'`** goes from $0 \rightarrow (N-1)$). **`Loop-1.1`** runs $[N \rightarrow (i+1)]$ times, *i.e.,* **`Loop-1.1`** is dependent on **`Loop-1`**. Therefore we analyse this code with the help of the following table and counting total number of iterations::

When 'i' is	'j' takes the values	#iterations
0	N, N-1, N-2, ..., 1	N
1	N, N-1, N-2, ..., 2	N-1
2	N, N-1, N-2, ..., 3	N-2
3	N, N-1, N-2, ..., 4	N-3
...	...	...
N-1	N	1

If we sum up the values in the last column, we have total iterations as:

`T(N)` **`= N + (N-1) + (N-2) + (N-3) + ... + 1`**
`= 1 + 2 + 3 + 4 + ... + (N-2) + (N-1) + N`
This is the the Sum of first 'N' Natural Numbers.

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

$\therefore$ Time Complexity is $O(N^2)$ or, it is Quadratic in Nature.

(D)

```
int a = 0, i = N;
while (i > 0) {   // Loop-1
    a += i;
    i /= 2;
}
```

Solution: In **Loop-1**, note that **a += i;** statement inside the loop doesn't affect **'i'**. Therefore, we don't care whatever happens to **'a'**, as **'a'** is not the variable through which **Loop-1** is controlled.

We only concentrate on the values of **'i'**, as the value of **'i'** determines when the loop **Loop-1** ends.

When 'i' is	#iterations	#total iterations
N	1	1
N/2	1	2
N/4	1	3
N/8	1	4
…	…	
$N/2^K$	1	K+1

We know that, **Loop-1** ends when,

i == 0 [as soon as **i** becomes **0**, **Loop-1** breaks]

∴ In the last loop **i**'s value is **1**. This means that **Loop-1** runs till:

i = **1**

or, $N/2^K = 1$ [∵ in the last loop, $i = N/2^K$]

or, $N = 2^K$

or, $\log_2(N) = \log_2(2^K)$ [Applying $\log_2$ on B.S]

or, $\log_2(N) = K$ [∵ $\log_a(a^K) = K$]

We got, $K = \log_2(N)$.

∴ Time Complexity is $K+1=\log_2(N)+1$, which is $O(log_2(N))$ - Logarithmic in Nature.

(E)

```
void fun(int N, int K) {
    for (int i = 1; i <= N; i++) { // Loop-1
        /* Assume that pow() takes constant amount of time */
        int P = pow(i, K);
        for (int j = 1; j <= P; j++) { // Loop-1.1
              /* Some constant amount of computation */
        }
    }
}
```

Solution: `Loop-1` depends on `N`, and the `Loop-1.1` depends on `P`, which is i^K.
$\therefore$ we can say that `Loop-1` runs `P` times, and for each iteration of `Loop-1`, `Loop-1.1` runs i^K times, *i.e.,* we get $N \times i^K$. But, the given input variables are `N` & `K`, and our final time complexity should always be expressed in terms of inputs, which is `N` & `K` for this problem. So, we have to eliminate `i` from the expression. To do that, we need to analyse it more.

We have, $P = i^K$, with this value, we cannot get anywhere while analysing the loops. What we can do is, we can assign `K = 1, 2, 3, ...` to get the correct time complexity.

For `K = 1`:

When 'i' is	$P = i^1$	'j' takes the values	#iterations
1	$1^1 = 1$	1	1
2	$2^1 = 2$	1, 2	2
3	$3^1 = 3$	1, 2, 3	3
4	$4^1 = 4$	1, 2, 3, 4	4
...	...	...	...
N-1	$(N-1)^1 = N-1$	1, 2, 3, ..., (N-1)	N-1
N	$N^1 = N$	1, 2, 3, ..., N	N

When we summate the last column, our sum will be:

$T(N) = 1 + 2 + 3 + 4 + \ldots + (N-1) + N$ [**Sum of first 'N' Natural Numbers**]

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

$\therefore$ when `K = 1`, the highest degree term we got is $\frac{N^2}{2}$

For **K** = 2:

When 'i' is	P = i^2	'j' takes the values	#iterations
1	$1^2 = 1$	1	1
2	$2^2 = 4$	1, 2, 3, 4	4
3	$3^2 = 9$	1, 2, 3, …, 9	9
4	$4^2 = 16$	1, 2, 3, …, 16	16
…	…	…	…
N-1	$(N-1)^2 = N^2+2N-1$	1, 2, 3, …, (N^2+2N-1)	N^2+2N-1
N	$N^2 = N^2$	1, 2, 3, …, N^2	N^2

If we summate the last column, we will have the number of times both **Loop-1** and **Loop-1.1** ran, and that is:

$T(N) = 1 + 4 + 9 + 16 + \ldots + (N^2+2N-1) + N^2$

$= 1^2 + 2^2 + 3^2 + 4^2 + \ldots + (N-1)^2 + N^2$

[**Sum of Squares of first 'N' Natural Numbers**]

$= \frac{N(N+1)(2N+1)}{6} = \frac{N}{3}(N+1)(N+\frac{1}{2}) = \frac{N^2+N}{2}(N+\frac{1}{2}) = \frac{N^3}{3} + \frac{N^2}{2} + \frac{N}{6}$

∴ when **K** = 2, the highest degree term we got is $\frac{N^3}{3}$

For **K** = 3:

When 'i' is	P = i^3	'j' takes the values	#times Loop-1 & Loop-1.1 runs
1	$1^3 = 1$	1	1
2	$2^3 = 8$	1, 2, 3, …, 8	8
3	$3^3 = 27$	1, 2, 3, …, 27	27
4	$4^3 = 64$	1, 2, 3, …, 64	64
5	$5^3 = 125$	1, 2, 3, …, 125	125
…	…	…	…
N	$N^3 = N^2$	1, 2, 3, …, N^3	N^3

If we summate the last column, we will have the number of times both **Loop-1** and **Loop-1.1** ran, and that is:

$T(N) = 1 + 8 + 27 + 64 + \ldots + N^3$

$= 1^3 + 2^3 + 3^3 + 4^3 + \ldots + (N-1)^3 + N^3$

[**Sum of Cubes of first 'N' Natural Numbers**]

T(N) $= \frac{(N)^2 (N+1)^2}{(2)^2} = \frac{N^2}{4}(N^2 + 2N + 1) = \frac{N^4}{4} + \frac{N^3}{2} + \frac{N^2}{4}$

$\therefore$ when **K = 3**, the highest degree term we got is $\frac{N^4}{4}$

Now, when:

1. when **K = 1**, the highest degree term we got is $\frac{N^2}{2} = \frac{N^{1+1}}{1+1}$
2. when **K = 2**, the highest degree term we got is $\frac{N^3}{3} = \frac{N^{2+1}}{2+1}$
3. when **K = 3**, the highest degree term we got is $\frac{N^4}{4} = \frac{N^{3+1}}{3+1}$

... .

... .

k. when **K = k**, the highest degree term will be $\frac{N^{k+1}}{k+1}$

$\therefore$ Time Complexity is $O(\frac{N^{k+1}}{k+1})$.

(F)

```
int count = 0;
for (int i = N; i > 0; i /= 2) { // Loop-1
    for (int j = 0; j < i; j++) { // Loop-1.1
        count += 1;
    }
}
```

Solution: **Loop-1** is non-linear in nature and is an independent loop. **Loop-1.1** is linear in nature and depends on **Loop-1**. Therefore, we need to analyse both **Loop-1** & **Loop-1.1** iteration by iteration, *i.e.,*

when 'i' is	'j' takes the values	#iterations
N	$0, 1, 2, 3, \ldots, (N-1)$	N
$\frac{N}{2}$	$0, 1, 2, 3, \ldots, (\frac{N}{2}-1)$	$\frac{N}{2}$
$\frac{N}{4}$	$0, 1, 2, 3, \ldots, (\frac{N}{4}-1)$	$\frac{N}{4}$
...	...	...
$\frac{N}{2^k}$	$0, 1, 2, 3, \ldots, (\frac{N}{2^k}-1)$	$\frac{N}{2^k}$

If we summate the last column, we will have the number of times both **Loop-1** and **Loop-1.1** ran, and that is:

T(N) $= N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \ldots + \frac{N}{2^k}$

$= N\left((\frac{1}{2})^0 + (\frac{1}{2})^1 + (\frac{1}{2})^2 + \ldots + (\frac{1}{2})^{k-1} + (\frac{1}{2})^k\right)$ ------- *Eqn (1)*

[This is a **Geometric Progression]**

$\text{Sum}_{GP} = \frac{a\left(1 - r^{T}\right)}{1 - r}$, where: a is first term, r is the common ratio & T is number of terms in the GP.

From *Eqn (1)*, we have, $a = \left(\frac{1}{2}\right)^{0} = 1$

$r = \frac{\left(\frac{1}{2}\right)^{1}}{\left(\frac{1}{2}\right)^{0}} = \frac{\left(\frac{1}{2}\right)^{2}}{\left(\frac{1}{2}\right)^{1}} = \left(\frac{1}{2}\right) \neq 1$. Therefore, the GP formula can be applied.

$T \Rightarrow$ Number of terms in the GP, *i.e.,* between $\left(\frac{1}{2}\right)^{0} \rightarrow \left(\frac{1}{2}\right)^{k}$ there are (k-0+1) terms, *i.e.,* there are k+1 terms.

If we substitute the values of a, r & T in Sum_{GP} formula, we have:

$$\text{Sum}_{GP} = \frac{1\left(1 - \left(\frac{1}{2}\right)^{k+1}\right)}{1 - \frac{1}{2}} = \frac{1\left(1 - \left(\frac{1}{2}\right)^{k+1}\right)}{\frac{1}{2}}$$

$$= 2\left(1 - \left(\frac{1}{2}\right)^{k+1}\right) \text{ ------- } \textit{Eqn (2)}$$

When we substitute the value in *Eqn (2)* in *Eqn (1)*, we have:

$$T(N) = N\left(2\left(1 - \left(\frac{1}{2}\right)^{k+1}\right)\right) \text{ ------- } \textit{Eqn (3)}$$

In Eqn (3), the term k+1 needs to be eliminated to get our answer in terms of our input, N. Now, we know that **`Loop-1`** only runs till $i > 0 \Rightarrow i = 1$ in the final loop. When i has reached the value 1, it means that, **`Loop-1`** is in its $\left(\frac{N}{2^{k}}\right)^{th}$ iteration. We can use this information and equate i to:

	i	=	1	
or,	$\frac{N}{2^{k}}$	=	1	
or,	N	=	2^{k}	[Apply log_2 on both sides]
or,	$log_{2}(N)$	=	k	------- *Eqn (4)*

We can now substitute the value of k from *Eqn (4)* into *Eqn (3)*, *i.e.,*

$$T(N) = N\left(2\left(1 - \left(\frac{1}{2}\right)^{log_{2}(N)+1}\right)\right) \text{ ------- } \textit{Eqn (5)}$$

Now, if we take values for N, that will approximate T(N), *i.e.,*

1. $N = 2^{30} \Rightarrow T(N) = N\left(2\left(1 - (0.5)^{log_{2}(2^{30})+1}\right)\right)$

$$= N\left(2\left(1 - (0.5)^{31}\right)\right)$$

$$= N\left(2\left(1 - (4.65 * 10^{-10})\right)\right)$$

$$\Rightarrow T(N) \approx 2N$$

2. $N = 2^{60} \Rightarrow T(N) = N\,(\,2\,(1-(0.5)^{\log_2(2^{60})+1})\,)$

$= N\,(\,2\,(1-(0.5)^{61})\,)$

$= N\,(\,2\,(1-(4.33 * 10^{-19}))\,)$

$\Rightarrow T(N) \approx 2N$

In both 1 & 2, we can see that as we increase the value of N, the value reaches closer and closer to $2N$ *i.e.,*

$$\lim_{N \to \infty} T(N) = 2N$$

$\therefore$ Time Complexity is $O(N)$ - Linear in Nature

(G)

```
int k = 0;
for(int i = N/2; i <= N; ++i) { // Loop-1
    for (int j = 2; j <= N; j = j * 2) { // Loop-1.1
        k = k + N/2;
   }
}
```

Solution: **`Loop-1`** is linear in nature and it runs for values $\frac{N}{2} \to N$, *i.e.,* **`Loop-1`** runs for $\frac{N}{2} + 1$ times. **`Loop-1.1`** is non-linear in nature and it is not dependent on **`Loop-1`**. Therefore, we can deduce the time complexity of **`Loop-1.1`** separately, *i.e.,*

'j' takes the value	2	4	8	16	...	2^k

Therefore, in the final loop, **`'j'`** takes the value 2^k. **`Loop-1.1`** ends when **`'j'`** reaches $N+1$. We can use this information, and get,

$$\begin{aligned} j &= N \\ 2^k &= N \quad \text{(read the paragraph above)} \\ k &= \log_2(N) \quad \text{(Applying } \log_2 \text{ on both sides)} \\ \Rightarrow \quad T(N) &= \log_2(N) \end{aligned}$$

`Loop-1` runs $\frac{N}{2} + 1$ times and for each loop in **`Loop-1`**, **`Loop-1.1`** runs $\log_2(N)$ times.

$\therefore$ Time Complexity is $O(N * \log_2(N))$ - Quasilinear in Nature.

(H)

```
int j = 0;
for(int i = 0; i < N; ++i) { // Loop-1
    while(j < N && arr[i] <= arr[j]) { // Loop-1.1
        j++;
    }
}
```

Solution: **`Loop-1`** is linear in nature and it runs for N times (as **`i`** takes the values $0 \rightarrow (N-1)$). **`Loop-1.1`** has a conditional statement **`arr[i] <= arr[j]`** which, if we assume to be **`true`**, then, when **`i`** is 0 in **`Loop-1`**, **`j`** reaches N only once in **`Loop-1.1`**, after that, when **`i`** goes from $1 \rightarrow (N-1)$, **`Loop-1.1`**'s conditional statement "**`j < N`**" is always **`false`** and **`Loop-1.1`** doesn't iterate even once.

$\therefore$ Time Complexity is $O(N)$ - Linear in Nature.

Sone Important Points to Note

1. When a loop is ***linear in nature***, it means that the loop variable increases/decreases by only a **single unit** in every iteration of the loop.
   Example-1 : **`for(int i = 0; i < N; ++i)`** This loop is linear in nature as **`i`** increases from $0 \rightarrow (N-1)$ in steps of 1 each.

   Example-2: **`for(int i = N; i > 0; --i)`** This loop is linear in nature as **`i`** decreases from $N \rightarrow 1$ in steps of 1 each.

   Example-3: **`for(int i = N; i > 0; i /= 2)`** This loop is non-linear in nature as **`i`** decreases in divisions of 2 after every iteration.

Practice Problem: Match the following Time Complexities

(1) `Linear`	**(A)** N^{K+G}	**(4)**
(2) `Logarithmic`	**(B)** $5^{N \times 2}$	**(3)**
(3) `Exponential`	**(C)** $\frac{N}{4}\log_2(\frac{N}{4000})$	**(5)**
(4) `Polynomial`	**(D)** $3^{20}N + 10^5$	**(1)**
(5) `Linear Logarithmic`	**(E)** $10N + 9\frac{N}{100} + 340N^2$	**(6)**
(6) `Quadratic`	**(F)** $10^3\log_2(N+3N)$	**(2)**