1.Consider the following statement:

```cout << "Hello";```

How many times is the statement executed?

$\Longrightarrow$ 1 Times

2. Consider the code:

```
for (int i = 0; i < n; i++)
cout << i;
```

(a) How many times does the cout statement execute when n = 10?

$\Longrightarrow$ 9 times

(b) How many times when n = 100?

$\Longrightarrow$ 99 times

3. Consider the code:

```
for (int i = 0; i < n; i++) {
int x = i + 1;
int y = x * 2;
}
```

If n = 5, how many assignment operations are performed?

$\Longrightarrow$ init 1 = 0; 1

i < n; 6

i++ 5

int x = i + 1; 5

int y = x * 2; 5

Total 22 Operations

4.Consider the loop:

```
for (int i = 0; i < n; i += 2)
cout << i;
```
```

(a) How many times does the loop run when n = 10?

$\implies$ 9 Times

(b) Approximately how many times does it run for general n?

$\implies$ n-1 Times

5 Consider the loop:

```
for (int i = n; i > 0; i--)
cout << i;
```

Does counting down from n instead of up to n change the total amount of work?

$\implies$ No, the loop will still execute n-1 times in this loop

---

Nested Loops and Growth

6.Consider the code:

```
for (int i = 0; i < n; i++)
cout << i;
```

```
for (int j = 0; j < n; j++)
cout << j;
```

How many total print statements are executed?

$\implies$ $(n-1) \times 2$ Times

7.Consider the nested loop:

```
for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
cout << i << j;
```

(a) How many times does cout execute when n = 3?

$\implies$ $3^2 = 9$ Times

(b) Represent the execution as a table.

```
i  j
0  0
0  1
0  2
1  0
1  1
1  2
2  0
2  1
2  2
```

8.Consider the code:

```
for (int i = 0; i < n; i++)
for (int j = 0; j < 5; j++)
cout << i << j;
```

For large n, which loop contributes more to the total work?

$\Longrightarrow$ `for (int j = 0; j < 5; j++)` Contributes more towards

9.Consider the code:

```
for (int i = 0; i < n; i++)
for (int j = 0; j <= i; j++)
cout << "*";
```

(a) Draw the output pattern when n = 4.

$\Longrightarrow$ `***************`

(b) Is the total work closer to n, n^2, or something in between?

$\Longrightarrow$ Total Work is $\frac{n(n+1)}{2}$

10.Consider the code:

```
for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
for (int k = 0; k < 3; k++)
cout << i << j << k;
```

Does adding a loop with a small constant limit significantly change the growth?

$\implies$ if the loop:

```
for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
```

was growing at rate $j \times i$ it is now growing at $3(j \times i)$ which is 3 times the growth. It may not be significant for small values under 10 but as the n grows the value grows too

---

Input Size and Growth Intuition

11.A program takes 1 second for input size n = 100 and approximately 4 seconds for input size n = 200. Is the growth more likely linear or quadratic? Explain.

$\implies$ when n = 100, t = 1sec
when n = 200, t = 4sec
$100^2 = 10000$
$200^2 = 40000$
if we considered it took 1 sec to process 10000 instructions it falls in line with taking 4 sec to process 40000 instructions which tells us the growth is more likely **Quadratic**

12.Rank the following functions according to their growth rate as n becomes large:

n, n + 100, 2n

$\implies$ For growth rate (Big-O thinking), we ignore constants and lower-order terms.

n+100 ≈n for large n (constant 100 becomes negligible)
2n differs only by a constant factor

So all three are **linear**.

Ranking: n < n + 100 < 2n

13.Rank the following functions according to their growth rate:

n, n^2, n^3

Use values such as n = 10, 100, and 1000 to justify your answer.

$\implies$ For growth rate (Big-O thinking), we ignore constants and lower-order terms. As there are no constants, we should focus on n

for n = 10
$n^2 = 100, n^3 = 1000$

for n = 100
$n^2 = 10000, n^3 = 1000000$

for n = 1000
$n^2 = 1000000, n^3 = 1000000000$

as we can see the growth rate isn't the same for $n, n^2 \ \& \ n^3$
rating: $n < n^2 < n^3$

14.Which function grows faster for large n?
$1000n \ or \ n^2$
Approximately for what value of n does n^2 become larger?

$\implies$ in general n^2 will grow faster in terms of O() is concerned, but to go in details: till n = 1000 both 1000n will grow faster but above 1000, n^2 will grow faster

for $1000n > n^2$
$n > n^2/1000$

for n = 1000
1000n = 1000000
n^2 = 1000000
1000n = n^2

for n = 1001
1000n = 1001000
n^2 = 1002001
$n < n^2$

Introducing Big-O Notation

15.Consider the loop:

```
for (int i = 0; i < 3*n + 5; i++)
     cout << i;
```

For large n, which part of the expression mainly determines the number of itera-tions?

$\implies$ i < 3*n determines the number of iterations

16.Identify the dominant term for large n:

$n^2 + 10n + 50, n + log\ n$

$\implies$ for $n^2 + 10n + 50$: n^2
for $n + log\ n$: n

17.Match each type of code to its Big-O complexity:

• A loop that runs a fixed number of times
• A loop that runs n times
• Two nested loops each running n times
Options: $O(n^2),\ O(1),\ O(n)$.

$\implies$
• A loop that runs a fixed number of times: $O(1)$
• A loop that runs n times: $O(n)$
• Two nested loops each running n times: $O(n^2)$

18.Consider the code:
for (int i = 0; i < n; i++)
if (arr[i] == key)
break;

(a) When does the loop terminate early?
$\implies$ if the arr[i] key is found | i != n & i < n
(b) When does it execute all iterations?

$\Longrightarrow$ if the arr[i] key is at the last index

i.e arr[i] == n