Lab 2

1. Big O Notation

1. **Let $f(n) = 2n + 6$. Is $f(n) = O(n)$**

We say $f(n) = O(n)$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that:

$$f(n) \leq c \cdot n \quad \text{for all } n \geq n_0$$

Given:

$$f(n) = 2n + 6$$

We want:

$$2n + 6 \leq c \cdot n \Rightarrow 6 \leq (c-2)n \Rightarrow n \geq \frac{6}{c-2}$$

Choose $c = 3$:

$$n \geq \frac{6}{1} = 6$$

So,

$$f(n) = 2n + 6 = O(n)$$

2. **Let $f(n) = 3n^2 + 4n - 8$. Is $f(n) = O(n^2)$**

We want:

$$3n^2 + 4n - 8 \leq c \cdot n^2 \Rightarrow 4n - 8 \leq (c-3)n^2$$

Try $c = 5$:

$$4n - 8 \leq 2n^2 \Rightarrow 0 \leq 2n^2 - 4n + 8$$

This is always true for $n \geq 1$.

So,

$$f(n) = 3n^2 + 4n - 8 = O(n^2)$$

3. Prove or disprove that $log(n!)$ is $O(nlogn)$

We use the fact that:

$$4^4 = 4 \times 4 \times 4 \times 4 = 256$$

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

Prove

$$f(n) \leq c \cdot g(n)$$

$$log(1 \times 2 \times 3 \ldots) \leq log(n \times n \times n \ldots)$$

So:

$$\log(n!) \leq n \log n$$

This shows:

$$\log(n!) = O(n \log n)$$

4. Prove that $2n + 3$ is $O(n^2)$

We want to prove:

$$2n + 3 \leq c \cdot n^2 \quad \text{for large } n$$

Divide both sides:

$$\frac{2n+3}{n^2} \leq c \Rightarrow \frac{2}{n} + \frac{3}{n^2} \leq c$$

As $n \to \infty$, both terms on the left go to 0.

So there exists some c (e.g., $c = 1$) such that the inequality holds for $n \geq 3$.

So,

$$2n + 3 = O(n^2)$$

5. Prove that 2^{n+2} is $O(2^n)$

$$f(n) = 2^{n+2} = 4 \cdot 2^n \Rightarrow f(n) = 4 \cdot 2^n$$

We want:

$$f(n) \leq c \cdot 2^n \Rightarrow 4 \cdot 2^n \leq c \cdot 2^n \Rightarrow 4 \leq c$$

Let $c = 4$, inequality holds for all $n \geq 0$

So,

$$2^{n+2} = O(2^n)$$

2. Write the recurrence relation of the following algorithms

1.

$$T(n) = T(n-1) + 4$$

$$T(0) = 1$$

$$\mathrm{T(n) = T(n - 1) + 4, \quad T(0) = 1}$$

This is a **linear recurrence relation** with a constant added at each step.

$$T(n) = 4n + 1$$

2.

$$T(n) = T(n-1) + 8$$

$$T(1) = 8$$

$$T(n) = T(n-1) + 8, \quad T(1) = 8$$

This is also linear, similar to the first one but with a different base and increment.

Unrolling the recurrence:

$$T(n) = 8(n-1) + 8 = 8n$$

3.

$$T(n) \ = \ 4T(\frac{n}{2}) \ + \ n$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

This is a **divide and conquer** recurrence.

Use the **Master Theorem**:

- $a = 4, b = 2, d \ = \ 1, f(n) = n$
- Compare $f(n) \ = \ n$ to $n^{\log_b a} = n^{\log_2 4} = n^2$ and $n^d = n^1 = n$

So, $n^2 > n^1$

Solution:

$$T(n) = O(n^2)$$

4.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Use Master Theorem:

- $(a = 4), (b = 2), (d = 2), (f(n) = n^2)$
- $(n^{\log_b a} = n^2), and (n^2)$

So, $n^2 = n^2$

Solution:

$$T(n) = O(n^2 log_2 n)$$

5.

$$T(n) = 2T\left(\frac{n}{8}\right) + \sqrt[3]{n}$$

Use Master Theorem:

- $(a = 2), (b = 8), \left(d = \frac{1}{3}\right), (f(n) = n^{1/3})$

- Compare with $(n^{\log_b a} = n^{\log_8 2} = n^{1/3})$

So, $n^{1/3} = n^{1/3}$

Solution:

$$T(n) = O\left(n^{\frac{1}{3}} log_8 n\right) = O\left(\sqrt[3]{n} \cdot log_8 n\right)$$

6.

```
int factorial(int n){
    if (n==1)
        return 1;
    return n * factorial(n-1);
}
```

Find time complexity of the following code

1.

```
void foo(int n){
     int i = 1; int s = 1;
     while (s <= n){
         i++;
         s = s + i;
         System.out.print("*");
     }
}
```

Analysis:

This loop continues while:

$$s = 1 + 2 + 3 + \cdots + i \leq n \Rightarrow \frac{i(i+1)}{2} \leq n \Rightarrow i = O(\sqrt{n})$$

Time Complexity:

$$\boxed{O(\sqrt{n})}$$

2.

```
void foo(int n){
     int count = 0;
    for (int i = n / 2; i <= n ; i++) {
        for(int j = 1; j + (n/2) <= n; j++){
            for(int k = 1; k <= n; k*=2){
                count++;
            }
        }
    }
}
```

Analysis:

- Outer loop: runs from $n/2$ to n → $O(n)$
- Middle loop: $j + n/2 \leq n \Rightarrow j \leq n/2$ → $O(n)$
- Inner loop: $k *= 2$ → $O(\log n)$

Total:

$$O(n) \cdot O(n) \cdot O(\log n) = \boxed{O(n^2 \log n)}$$

3.

```
void foo(int n){
     int count = 0;
    for (int i = n / 2; i <= n ; i++) {
        for(int j = 1; j <= n; j*=2){
            for(int k = 1; k <= n; k*=2){
                count++;
            }
        }
    }
}
```

Analysis:

- Outer loop: runs from $n/2$ to $n \rightarrow O(n)$
- Middle loop: $O(\log n)$
- Inner loop: $O(\log n)$

Total:

$$O(n \log n \log n) = \boxed{O(n \log^2 n)}$$

4.

```
void foo(int n){
    if(n == 1) return;
    for (int i = 1; i <= n ; i++) {
        for(int j = 1; j <= n; j++){
            System.out.print("*");
            break;
        }
    }
}
```

Analysis:

- If condition: constant time $O(1)$
- Outer loop: $O(n)$
- Inner loop breaks after 1st iteration → constant

Total:

$$O(n) \cdot O(1) = \boxed{O(n)}$$

5.

```
void foo(int n){
    int a = 0; int i = n;
    while (i > 0){
        a += i;
        i /= 2;
    }
}
```

Analysis:

Loop halves i each time: $n, n/2, n/4, \ldots$

Number of iterations: $O(\log n)$

Time Complexity:

$$\boxed{O(\log n)}$$

6.

```
void foo(int n){
     int count = 0;
    for(int i = n; i > 0; i /= 2){
        for(int j = 0; j < i; j++){
            count++;
        }
    }
}
```

Analysis:

Outer loop: $i = n, n/2, n/4, \ldots \rightarrow O(\log n)$

Inner loop:

- First: n
- Second: $n/2$
- Third: $n/4$
- …

Total work: geometric sum

$$n + n/2 + n/4 + \cdots = \mathrm{n}(1 + 1/2 + 1/4 + \cdots) = O(n)$$

Time Complexity:

$$\boxed{O(n)}$$