Asymptotic Analysis

Analysis of Algorithms

- **Analysis of Algorithms** is the determination of the amount of **time, storage** and/or other **resources** necessary to execute them.

- Analyzing algorithms is called **Asymptotic Analysis.**

- **Asymptotic Analysis** evaluate the performance of an algorithm.

Time complexity

- **Time complexity** of an algorithm quantifies the amount of **time** taken by an algorithm.

- We can have three cases to analyze an algorithm:
  - Worst Case.
  - Average Case.
  - Best Case.

Time complexity

- Assume the below algorithm using C++ code:

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Worst Case Analysis

- In the worst case analysis, we calculate upper bound on running time of an algorithm.

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Worst Case Analysis

- The case that causes maximum number of operations to be executed.

- For Linear Search, the worst case happens when the element to be searched is not present in array (example: search for number 8).

2	3	5	4	1	7	6

Time complexity
Worst Case Analysis

- When x is not present, the search() functions compares it with the elements of **arr** one by one.

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Worst Case Analysis

- Time complexity of linear search would be O(n).

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Average Case Analysis

- We take all possible inputs and calculate computing time for all of the inputs.

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Best Case Analysis

- Calculate lower bound on running time of an algorithm.

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Best Case Analysis

- Time complexity in the best case of linear search would be O(1).

```
int search(const std::vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); ++i)
    {
        if (arr[i] == x)
        {
            return i + 1;
        }
    }
    return -1;
}
```

Time complexity
Best Case Analysis

- The case that causes minimum number of operations to be executed.

- For Linear Search, the best case occurs when x is present at the first location. (example: search for number 2).

- So time complexity in the best case would be $\Omega(1)$

2	3	5	4	1	7	6

Time complexity

- Most of the times, we do **worst case** analysis to analyze algorithms.
- The **average case** analysis is not easy to do in most of the practical cases and it is rarely done.
- The **best case** analysis is bogus. Guaranteeing a lower bound on an algorithm doesn’t provide any information.

Asymptotic Notations

- **Big-O Notation:** is an Asymptotic Notation for the upper bound.
- **Ω Notation** (omega notation): is an Asymptotic Notation for the lower bound.
- **Θ Notation** (theta notation): is an Asymptotic Notation for both the lower and the upper bounds.

Big-O Notation
O(1)

- Time complexity of a function (or set of statements) is considered as O(1) if it doesn't contain loop, recursion and call to any other non-constant time function. For example **swap()** function has O(1) time complexity.
- A loop or recursion that runs a constant number of times is also considered as O(1). For example the following loop is O(1).

```
int c = 4;
for (int i = 0; i < c; ++i)
{
    std::cout << i;
}
```

```
void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Big-O Notation
O(n)

- Time Complexity of a loop is considered as O(n) if the loop variables is incremented / decremented by a constant amount.
- For example the following loop statements have O(n) time complexity.

```
for (int i = 0; i < n; ++i)
{
    std::cout << i;
}
```

Big-O Notation
$O(n^c)$

- Time complexity of nested loops is equal to the number of times the innermost statement is executed.
- For example the following loop statements have $O(n^2)$ time complexity.

```
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        std::cout << i << j << std::endl;
    }
}
```

Big-O Notation
$O(log(n))$

- Time complexity of a loop is considered as O(log(n)) if the loop variables are divided / multiplied by a constant amount.

```
int sum = 0;
while (n != 0)
{
    sum += n % 10;
    n /= 10;
}
```

Big-O Notation

- How to combine time complexities of consecutive loops?

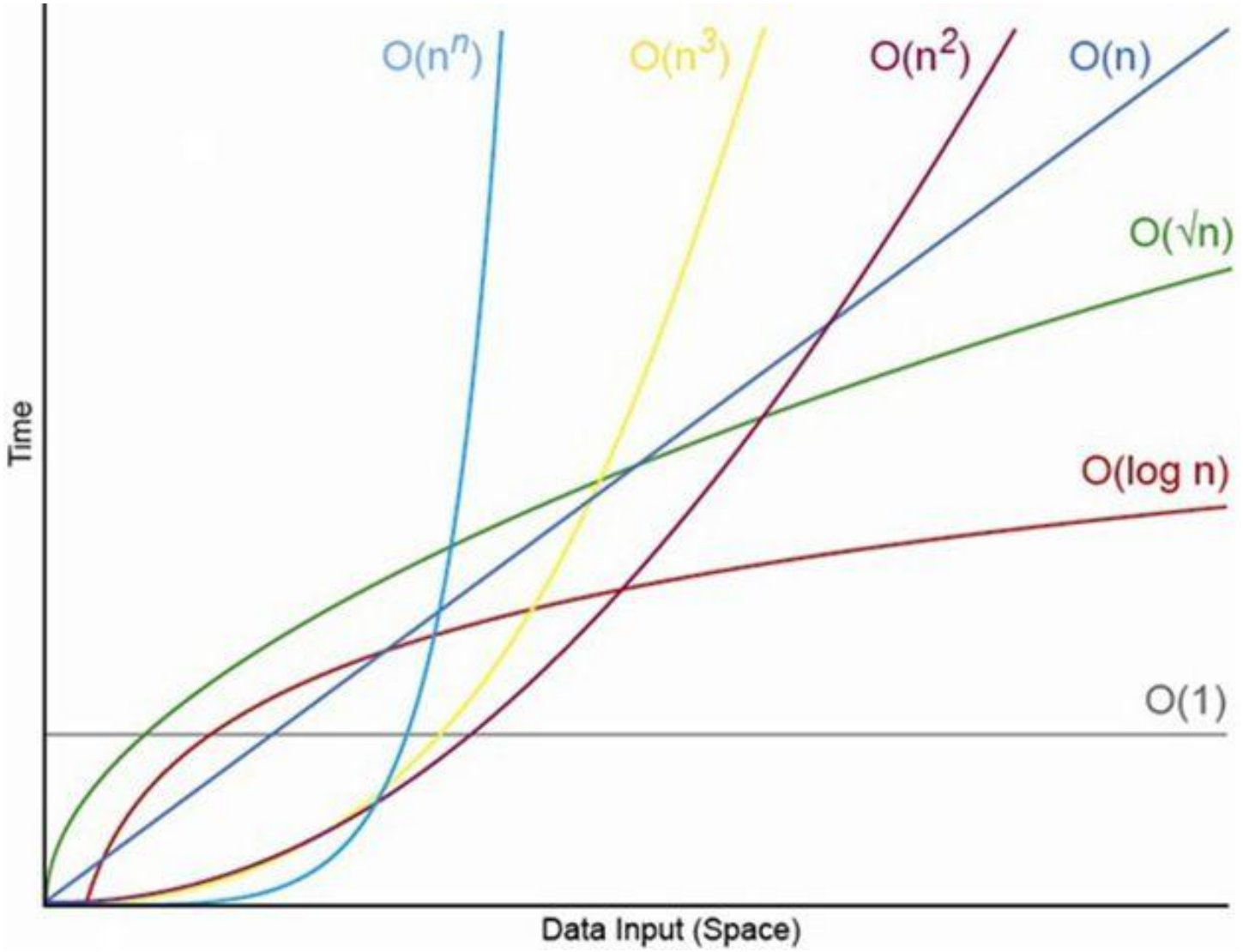
```
for (int i = 0; i < n; ++i)
    std::cout << i;
for (int i = 0; i < m; ++i)
    std::cout << i;
```

- Time complexity of above code is O(n) + O(m) which can also be written as O(n+m) or O(max(n, m)).

Big-O Notation. Growth Orders

n	O(1)	O(log(n))	O(n)		$O(N^2)$	$O(2^n)$	O(n!)
1	1	0	1	1	1	2	1
12	1	4	12	48	144	4096	$4x10^8$
27	1	5	27	135	729	$1.3x10^8$	10^{28}
500	1	9	500	4500	$2.5x10^5$	$3x10^{150}$	10^{1134}
1000	1	10	1000	$10x10^3$	10^6	10^{301}	$4x10^{2567}$
$16x10^3$	1	14	$16x10^3$	$2.2x10^5$	$2.6x10^8$	-	-
10^5	1	17	10^5	$1.7x10^6$	10^{10}	-	-

Big-O Notation. Growth Orders



Big-O Notation. Growth Orders

Big-O Notation

- What is this code complexity?

```
for(int i = 0; i < n; ++i)
    for (int j = 0; j < n * n; ++j)
        for (int k = 0; k < j; ++k)
        {
            std::cout << i << ' ';
            std::cout << j << ' ';
            std::cout << k << '\n';
        }
```

Big-O Notation

- What is this code complexity?

```
for(int i = 0; i < n; ++i)
    for (int j = 0; j < n * n; ++j)
        for (int k = 0; k < j; ++k)
        {
            std::cout << i << ' ';
            std::cout << j << ' ';
            std::cout << k << '\n';
        }
```

- Time complexity of above code is $O(n^5)$.

Big-O Notation

- What is this code complexity?

```
for (int i = 0; i < n; ++i)
    for (int j = 0; j * j <= n; ++j)
        std::cout << i + j << ' ';
```

Big-O Notation

- What is this code complexity?

```
for (int i = 0; i < n; ++i)
    for (int j = 0; j * j <= n; ++j)
        std::cout << i + j << ' ';
```

- Time complexity of above code is $O(n \cdot \sqrt{n})$.