Asymptotic Analysis

Solution: Peak Finder

Output of 2 different Algorithms on 1000 elements sorted in Ascending Order.

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 1
999 peak Value
743000 ns Execution Time
```

Algorithm A

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 2
999 peak Value
543000 ns Execution Time
```

Algorithm B

Solution: Peak Finder

Output of 2 different Algorithms on One Hundred Million elements sorted in Ascending Order.

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 1
99999999 peak Value
238955000 ns Execution Time
```

Algorithm A

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 2
99999999 peak Value
1268000 ns Execution Time
```

Algorithm B

Solution: Peak Finder

Output of 2 different Algorithms on 1000 elements sorted in Descending order.

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data (1000 - 0]
4. Laod large data (100000000 - 0]
5. Exit
Your Option: 1
999 peak Value
0 ns Execution Time
```

Algorithm A

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data (1000 - 0]
4. Laod large data (100000000 - 0]
5. Exit
Your Option: 2
999 peak Value
984000 ns Execution Time
```

Algorithm B

Solution: Peak Finder

Output of 2 different Algorithms on One Hundred Million elements sorted in Descending order.

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data (1000 - 0]
4. Laod large data (100000000 - 0]
5. Exit
Your Option: 1
99999999 peak Value
0 ns Execution Time
```

Algorithm A

```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data (1000 - 0]
4. Laod large data (100000000 - 0]
5. Exit
Your Option: 2
99999999 peak Value
1014000 ns Execution Time
```

Algorithm B

Problems with Execution time Comparison

1. It might be possible that for some inputs Algorithm A is better and for some other inputs Algorithm B is better.
2. It might be possible that algorithms are implemented in different languages and their execution time is different.
3. It might be possible that one Algorithm is implemented on a faster computer than the other.
4. It might be possible that one algorithm is tested on Earth and the other is tested on Mars.

Problems with Execution time Comparison

Therefore, measuring the Execution time for determining the Performance or comparison of Algorithms is not practical approach.

Asymptotic Analysis

In order to avoid these problems, Computer Scientists use the asymptotic analysis method for Analyzing the Performance of the Algorithms.

Asymptotic Analysis

The execution time generally depends on the size of the input.

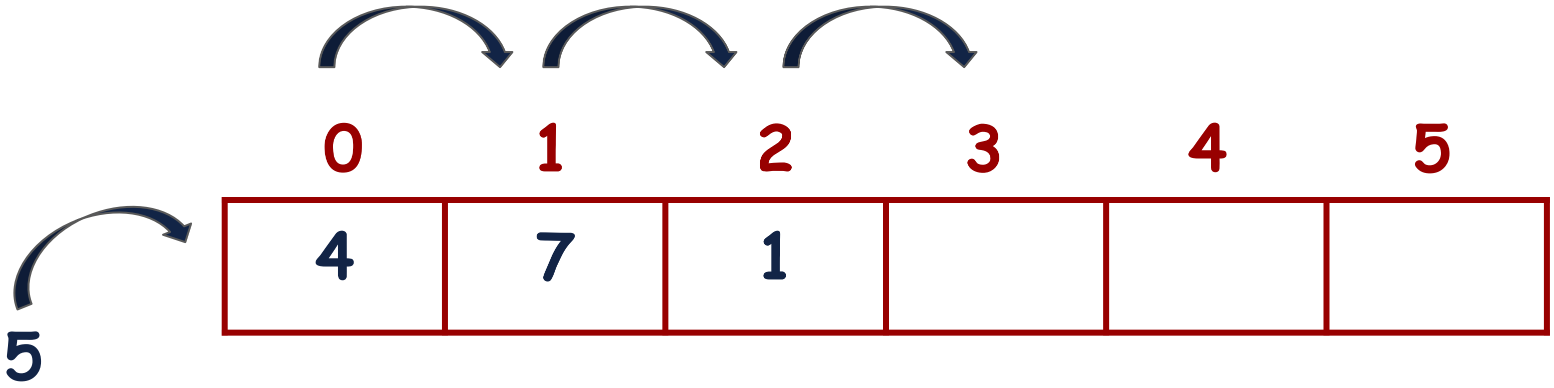
Asymptotic Analysis: Depends on Input Size

Suppose, we have an Array and we want to insert a new element at the start of the array.

0	1	2	3	4	5
4	7	1			

Asymptotic Analysis: Depends on Input Size

We will shift each element towards the right and then add the new element at the 0th index.



Asymptotic Analysis: Depends on Input Size

We will shift each element towards the right and then add the new element at the 0th index.
Suppose it takes 1 unit of time to perform each shift then how much time will it take to insert 5 at the beginning?

0	1	2	3	4	5
5	4	7	1		

Asymptotic Analysis: Depends on Input Size

Now suppose that we have 1000 elements in the array, then how much time will it take to insert the new element?

0	1	2 …	999	1000	1001
5	4	7	1		

Asymptotic Analysis

This clearly shows that the execution time depends on the size of the input.

Asymptotic Analysis

Therefore, We evaluate the performance of an Algorithm in terms of input size instead of the actual execution time.

Asymptotic Analysis

There are three asymptotic notations that are used to represent the time complexity of an algorithm.

Asymptotic Analysis: Peak Finder

There are three asymptotic notations that are used to represent the time complexity of an algorithm.

0	1	2	3	4
5	4	3	2	1

Best Case

Ω (Omega) Notation

Asymptotic Analysis: Peak Finder

There are three asymptotic notations that are used to represent the time complexity of an algorithm.

	0	1	2	3	4	
Best Case	5	4	3	2	1	Ω (Omega) Notation
Average Case	1	2	3	2	1	Θ (Theta) Notation

Asymptotic Analysis: Peak Finder

There are three asymptotic notations that are used to represent the time complexity of an algorithm.

	0	1	2	3	4	
Best Case	5	4	3	2	1	Ω (Omega) Notation
Average Case	1	2	3	2	1	Θ (Theta) Notation
Worst Case	1	2	3	4	5	O (Big O) Notation

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
int squareSum(int a, int b, int c) {
  int sa = a * a;
  int sb = b * b;
  int sc = c * c;
  int sum = sa + sb + sc;
  return sum;
}
```

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
int squareSum(int a, int b, int c) {
  int sa = a * a;
  int sb = b * b;
  int sc = c * c;
  int sum = sa + sb + sc;
  return sum;
}
```

$O(1)$

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
void printData(int n)
{
    for (int idx = 0; idx < n; idx++)
    {
        cout << idx << endl;
    }
}
```

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
void printData(int n)
{
    for (int idx = 0; idx < n; idx++)
    {
        cout << idx << endl;
    }
}
```

O(n)

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
void printData(int n)
{
     for (int idx = 1; idx < n; idx = idx * 2)
     {
          cout << idx << endl;
     }
}
```

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
void printData(int n)
{
    for (int idx = 1; idx < n; idx = idx * 2)
    {
        cout << idx << endl;
    }
}
```

$O(\log(n))$

Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
void printData(int n)
{
    for (int idx = 0; idx < n; idx++)
    {
        for (int idx1 = 0; idx1 < n; idx1 = idx1 + 2)
        {
            cout << idx1;
            cout << endl;
        }
    }
}
```

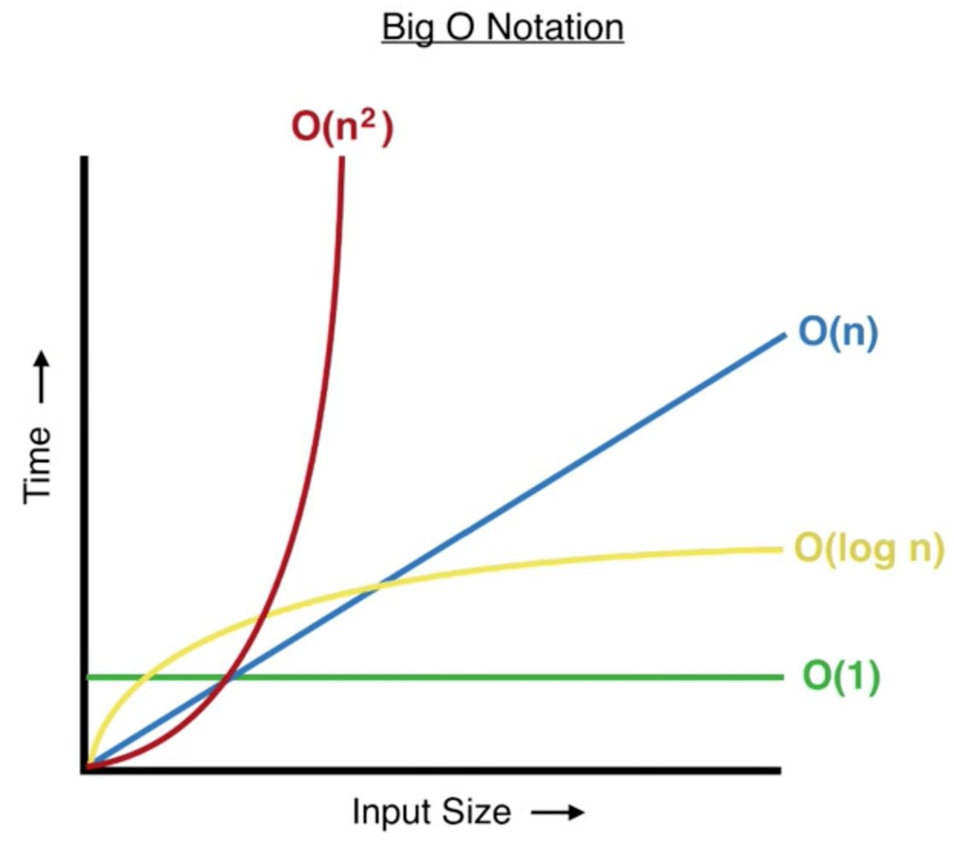
Asymptotic Analysis

Now, lets Calculate the Time Complexity in terms of Big O notation of different functions.

```
void printData(int n)
{
    for (int idx = 0; idx < n; idx++)
    {
        for (int idx1 = 0; idx1 < n; idx1 = idx1 + 2)
        {
            cout << idx1;
            cout << endl;
        }
    }
}
```

$O(n^2)$

Asymptotic Analysis



Learning Objective

Students should be able to **calculate** the time complexities of different algorithms

Self Assessment: Calculate Time Complexity

```
int fn(int [] arr, int target) {
  int low = 0;
  int high = arr.length - 1;
  int mid;
  while ( low <= high ) {
    mid = ( low + high ) / 2;
    if ( target < array[mid] )
      high = mid - 1;
    else if ( target > array[mid] )
      low = mid + 1;
    else
        break;
  }
  return mid;
}
```

Reading Activity

- https://adrianmejia.com/how-to-find-time-complexity-of-an-algorithm-code-big-o-notation/
- https://www.enjoyalgorithms.com/blog/time-complexity-analysis-in-data-structure-and-algorithms