Time Complexity

In the time complexity analysis we define the time as a function of the problem size and try to estimate the growth of the execution time with the growth in problem size.

Memory Space

- The space require by the program to save input data and results in memory (RAM).

History of Big-O-Notation

- Big O notation (with a capital letter O, not a zero), also called Landau's symbol, is a symbolism used in complexity theory, computer science, and mathematics to describe the basic behavior of functions. Basically, it tells you how fast a function grows or declines.
- Landau's symbol comes from the name of the German mathematician Edmund Landau who invented the notation.
- The letter O is used because the rate of growth of a function is also called its order.

What is Big O notation???

It describes

- Efficiency of an Algorithm
- Time Factor of an Algorithm
- Space Complexity of an Algorithm

It is represented by O(n) where n is the number of operations.

O(1)

```
void print(int index,int values[]){
cout<<“the value at index”<<index<< “ is ”<<value[index];
}
```

You can directly access the index element of the array, so no matter how big the array gets, the time it takes to access the element won’t change.

O(n)

```
void print(int index, int values[]){
for(int x=0;x<index;x++)cout<<index<<“ .”<<value[index]<<endl;
}
```

As you increase the size of the array, the time for this algorithm will increase proportionally. An array of 50 will take twice the time as an array of 25 and half the time of an array of 100.

O(n^2)

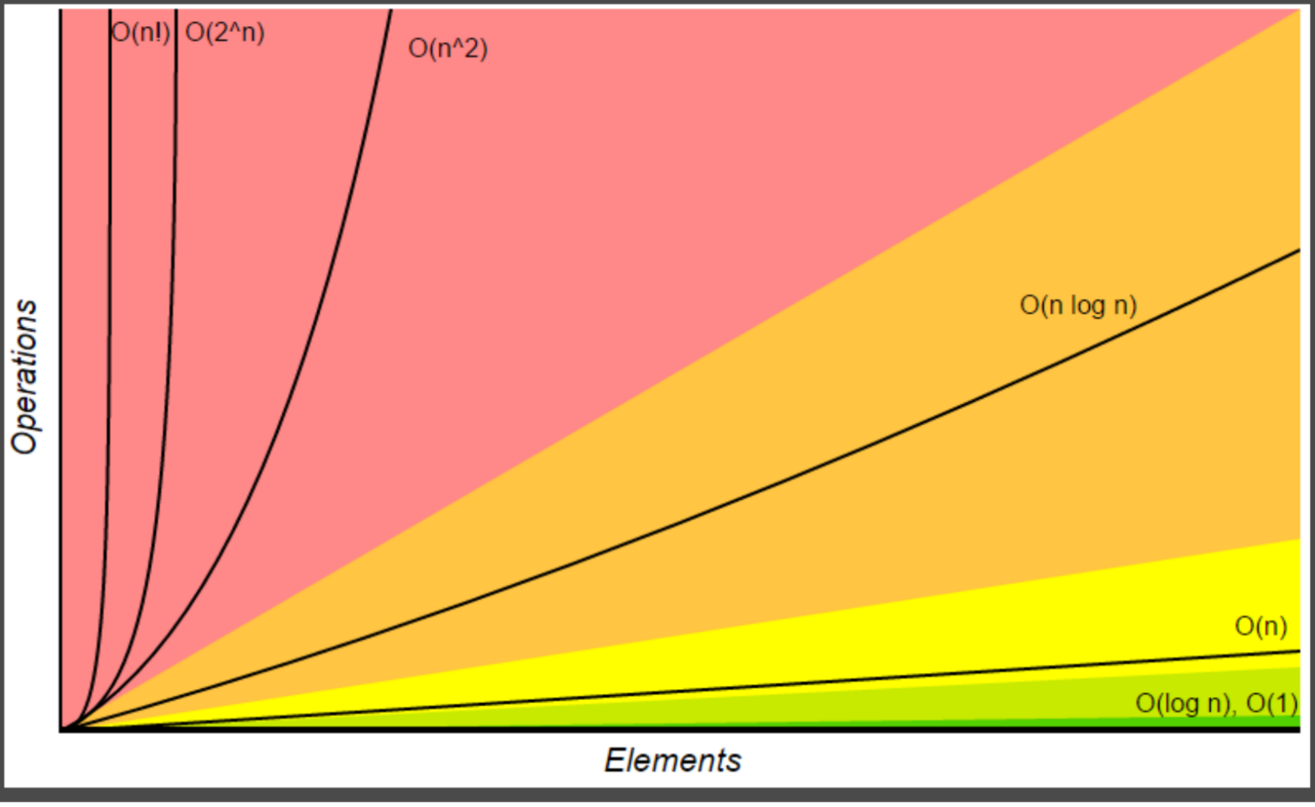
```
• int CompareToAllNumbers (int array1[ ] ,int index1,int array2[ ],int index2){
• for (int x = 0; x < index1; x++)
•            {bool isMin;
•             for (int y = 0; y < index2; y++)
•            { if( array[x] > array[y])
•            isMin = false;
•            }if(isMin)
•            break; }
•            return array[x];}
```

If you add one element to array, you need to go through the entire rest of the array one additional time. As the input grows, the time required to run the algorithm will grow quite quickly.

O(n!)

- Travelling salesman problem

in this problem as the number of cities increases the runtime also increases accordingly by a factor of cities factorial.


O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

We are having two different algorithms to find the smallest element in the array

```
int CompareSmallestNumber (int array[ ])
{     int x, curMin;
      curMin = array[0];
       for (x = 1; x < 10; x++)
       {    if( array[x] < curMin)
      curMin = array[x];}
return curMin;}
```

```
int CompareToAllNumbers (int array[ ])
{bool is Min;
int x, y;
for (int x = 0; x < 10; x++)
      { isMin = true;
       for (int y = 0; y < 10; y++)
      { if( array[x] > array[y])
      isMin = false;
      }if(isMin)
      break; }
      return array[x];}
```

Best ☺

- Uses array to store elements
- Compares a temporary variable in the array
- The Big O notation for it is $O(n)$

This algorithm is efficient and effective as its
O is small as compare to the 2nd algorithm
And hence it gives the best result in less time.

Worst ☹

- Uses array to store elements
- Compares the array with array itself
- The big O notation for it is $O(n^2)$

This algorithm is less effective than the other
Beacause this algo has a high Big O and hence
It is more time taking and less effective.

SCENARIOS

BEST
AVERAGE
WORST

Best case:

When the algorithm takes less
run time w.r.t the given inputs/data.
Best case also defines as Big Ω.

Average case:

When the algorithm takes
average run time w.r.t to the input /data.
This is also known as BigΘ.

Worst case:

When the algorithm takes higher runtime
as compared to the given inputs/data.
Worst case is also called as Big O.

The End
of presentation