Worst, Average and Best Case Time Complexities

It is important to analyze an algorithm after writing it to find it's efficiency in terms of time and space in order to improve it if possible.

When it comes to analyzing algorithms, the asymptotic analysis seems to be the best way possible to do so. This is because asymptotic analysis analyzes algorithms in terms of the input size. It checks how are the time and space growing in terms of the input size.

We can have three cases to analyze an algorithm:

1. **Worst Case**
2. **Average Case**
3. **Best Case**

Below is the algorithm for performing linear search:

```
// Linearly search x in arr[].
// If x is present then return the index,
// otherwise return -1int search(int arr[], int n, int x)
{
    int i;
    for (i=0;i<n;i++){
        if(arr[i]==x){
            return i;
        }
    }
    return -1;
 }

 //Driver program to test above functions
 int main(){
    int arr[]={2,8,12,9};
    int x=12;
    int n=sizeof(arr)/sizeof(arr[0]);
    printf("%d is present in %d index",x,search(arr,n,x));
    getchar();
    return 0;
}
```

1. **Worst Case Analysis (Usually Done):** In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes the maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search( ) functions compares it with all the elements of arr[ ] one by one. Therefore, the worst case time complexity of linear search would be O(N), where N is the number of elements in the array.
2. **Average Case Analysis (Sometimes done):** In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (N+1). Following is the value of average case time complexity.

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{\theta((n+1)*(n+2)/2)}{(n+1)}$$

$$= \theta(n)$$

3. **Best Case Analysis (Bogus):** In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on N). So time complexity in the best case would be O(1).

Example-

```
#include <bits/stdc++.h>
using namespace std;
```

```
// Linearly search x in arr[].
// If x is present then return the index,
// otherwise return -1
int search(int arr[], int n, int x)
{
   int i;
   for (i = 0; i < n; i++) {
       if (arr[i] == x)
           return i;
   }
   return -1;
}

// Driver's Code
int main()
{
   int arr[] = { 1, 10, 30, 15 };
   int x = 30;
   int n = sizeof(arr) / sizeof(arr[0]);

   // Function call
    cout << x << " is present at index "
        << search(arr, n, x);

   return 0;
}
```

Time Complexity Analysis: (In Big-O notation)

- **Best Case:** `O(1)`, This will take place if the element to be searched is on the first index of the given list. So, the number of comparisons, in this case, is 1.
- **Average Case:** `O(n),` This will take place if the element to be searched is on the middle index of the given list.
- **Worst Case:** `O(n),` This will take place if:
  - The element to be searched is on the last index
  - The element to be searched is not present on the list

Important Points:

- Most of the times, we do the worst case analysis to analyze algorithms. In the worst analysis, we guarantee an upper bound on the running time of an algorithm which is a good piece of information.
- The average case analysis is not easy to do in most of the practical cases and it is rarely done. In the average case analysis, we must know (or predict) the mathematical distribution of all possible inputs.
- The Best Case analysis is bogus. Guaranteeing a lower bound on an algorithm doesn't provide any information as in the worst case, an algorithm may take years to run.