191-15-12949

1. Short notes on Optimization

Optimization is a program transformation technique, which tries to improve the code by making it consume less resources and deliver high speed.

In optimization high level general programming constructs are replaced by very efficient low-level programming codes. Three rules of optimization -

* The output code must not, in any way change the meaning of the program.
* Optimization should increase the speed of a program and if possible, the program should demand a smaller number of resources.
* Optimization should itself be fast and should not delay the overall compiling process.

2. Different algorithms that I know.

Some algorithms -

Bubble sort - Sorting with the first element (Index = 0) compare the current element with the next element of the array. If the current element is greater than next element of the array, swap them and repeat 1 & 2.

Best case :- When the array is already sorted. The inner loop won't execute.

$$T(n) = O(n)$$

Worst, $T(n) = (N^2)$

Selection Sort :-

Selecting the lowest element requires scanning all n elements (n-1) (comparisons) and then swap it into the first position. And find the next lowest element requires and so on.

Best case :- $O(n)^2$

Worst case also same time complexity.

Insertion Sort :-

Suppose, an array ascending order and you want to sort it in descending order. that time's worst case complexity occurs.

Best case :- O(n)

Average case :- O(n)^2

3. Why I am learning so many algorithm -

Algorithms are clearly specified means to solve problems.

I think one of the most important things with algorithms is to realize that many of them are related.

An algorithm is a procedure or formula for solving a problem based on conducting a sequence of specified actions.

A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem. Algorithms are widely used throughout all areas of IT. That's why I ~~learned~~ study algorithms. We want to know that our code is based on ideas that solve the problem.

4. Show analysis of a recursive algorithm

Factorial

$n! = 1 \cdot 2 \cdot 3 \cdots n$ and $0! = 1$ (called initial case)

So the recursive definition $n! = n * (n-1)!$

```
Algorithm F(n)
if n=0 then return 1
       else F(n-1) * n // recursive call
```

Multiplication during the recursive call

Formula - $M(n) = M(n-1) + 1$

We need the initial case which corresponds to the base case -

$$M(0) = 0$$

There are no multiplication.

Solve by the method of backward situation-

$$M(n) = M(n-1) + 1$$

$$= [M(n-2)+1]+1 = M(n-2)+2 \text{ substituted}$$

$M(n-2)$ for $M(n-1)$

$$= [M(n-3)+1]+2 = M(n-3)+3$$

$M(n-3)$ for $M(n-2)$ . . . .

$$= M(0) + n$$

$$= n$$

Therefore, $M(n) \in O(n)$

5. Design an iterative and recursive algorithm and prove that my algorithms works -

Recursive :- (Binary search)

```
# include <stdio.h>
int binarySearch (int arr[], int l, int r, int x)
{   if (r >= 1)
    {  int mid = l + (r-l) /2 ;
       if (arr[mid] == x)
          return mid;
       if (arr[mid] > x)
       return binarySearch (arr, l, mid-1, x);

       return binarySearch (arr, mid+1, r, x);
    }

    return -1 ;
}
int main (void)
{ int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof (arr);
    int x = 10 ;
  int result = binarySearch (arr, 0, n-1, x);
```

```
(result == -1)?
    printf("not present in Array");
: printf("present at %d", result);

    return 0;
}
```

Algorithm prove:-

We know that at each step of algorithm, our search space reduces to half. That means if initially our research space reduces to half. Then iteration it contains $n/2$ then $n/4$ so on...

$n \rightarrow n/2 \rightarrow n/4 \dots > 1$

Suppose after M steps our search space is executed.

$n/2m = 1$

$n = 2m$

$m = \log 2n$

Binary search algorithm is $O(\log 2n)$ which in very efficient. $O(\log 2n)$ for recursive implementation due to call stack.