Design and Analysis of Algorithm (KCS503)

Introduction to Algorithms and its analysis mechanism

Lecture - 1

Overview

- Provide an overview of algorithms and analysis.
- Try to touching the distinguishing features and the significance of analysis of algorithm.
- Start using frameworks for describing and analysing algorithms.
- See how to describe algorithms in pseudo code in the context of real world software development.
- Begin using asymptotic notation to express running-time analysis with Examples.

What is an Algorithm ?

- An algorithm is a finite set of rules that gives a sequence of operations for solving a specific problem
- An algorithm is any well defined computational procedure that takes some value or set of values, as input and produces some value or set of values as output.
- We can also view an algorithm as a tool for solving a well specified computational problem.

Characteristics of an Algorithm

- **Input:** provided by the user.
- **Output:** produced by algorithm.
- **Definiteness:** clearly define.
- **Finiteness:** It has finite number of steps.
- **Effectiveness:** An algorithm must be effective so that it's output can be carried out with the help of paper and pen.

Analysis of an Algorithm

- **Loop Invariant** technique was done in three steps:
  - Initialization
  - Maintenance
  - Termination
- It deals with predicting the resources that an algorithm requires to its completion such as memory and CPU time.
- To main measure for the efficiency of an algorithm are Time and space.

Complexity of an Algorithm

- Complexity of an Algorithm is a function, f(n) which gives the running time and storage space requirement of the algorithm in terms of size n of the input data.
- Space Complexity: Amount of memory needed by an algorithm to run its completion.
- Time complexity: Amount of time that it needs to complete itself.

Cases in Complexity Theory

- Best Case: Minimum time
- Worst Case: Maximum amount of time
- Average Case: Expected / Average value of the function f(n).

Example 1

```
#include <stdio.h>

int main()
{
	printf("Hello PSIT");
	return 0;
}
```

Complexity of Example 1

```
#include <stdio.h>

int main()
{
	printf("Hello PSIT");
	return 0;
}
```

$$T(n) = O(1)$$

Example 2

```
#include <stdio.h>
void main()
{
    int i, n = 8;
    for (i = 1; i <= n; i++) {
        printf("Hello PSIT !!!\n");
    }
}
```

Complexity of Example 2

```
#include <stdio.h>
void main()
{
    int i, n = 8;
    for (i = 1; i <= n; i++) {
        printf("Hello PSIT !!!\n");
    }
}
```

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

Example 3

```
#include <stdio.h>
void main()
{
    int i, n = 8;
    for (i = 1; i <= n; i=i*2) {
        printf("Hello PSIT !!!\n");
    }
}
```

Complexity of Example 3

```
#include <stdio.h>
void main()
{
    int i, n = 8;
    for (i = 1; i <= n; i=i*2) {
        printf("Hello PSIT !!!\n");
    }
}
```

$$T(n) = \log_2 n$$

Example 4

```
#include <stdio.h>
#include <math.h>
void main()
{
        int i, n = 8;
        for (i = 2; i <= n; i=pow(i,2)) {
                printf("Hello PSIT !!!\n");
        }
}
```

Complexity of Example 4

```c
#include <stdio.h>
#include <math.h>
void main()
{
	int i, n = 8;
	for (i = 2; i <= n; i=pow(i,2)) {
		printf("Hello PSIT !!!\n");
	}
}
```

$$T(n) = O(\log_2(\log_2 n))$$

Example 5

```
A()
{
  int i;
    for(i=1;i<=n;i++)
       {
                    printf("ABCD");
        }
}
```

Complexity of Example 5

```
A()
{
  int i;
    for(i=1;i<=n;i++)
       {
                 printf(“ABCD”);
        }
}
```

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

Example 6

```
A()
{
    int i=1 ,s=1;
    scanf(“%d”, &n);
    while(s<=n)
       {
              i++;
              s=s+i;
              printf(“abcd”);
          }
}
```

Complexity of Example 6

```
A()
{
    int i=1 ,s=1;
    scanf("%d", &n);
    while(s<=n)
       {
              i++;
              s=s+i;
              printf("abcd");
          }
}
```

$$T(n) = O(\sqrt{n})$$

Example 7

```
A()
{
    int i=1;
        for(i=1; i²<=n; i++)
        {
            printf("abcd");
        }
}
```

Complexity of Example 7

```
A()
{
    int i=1;
        for(i=1; i^2<=n; i++)
        {
            printf("abcd");
        }
}
```

$$T(n) = O(\sqrt{n})$$

Example 8

```
A()
{
   int i=1;
   for (i=1; i≤n; i++)
   {
    for (j=1; j≤ i²; j++)
     {
        for (k=1; k≤n/2; k++)
        {
          printf("abcd");
        }
       }
     }
   }
```

Complexity of Example 8

$T(n) = O(n^4)$

Explanation:

$I\ =\ 1, 2, 3, 4, 5, \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.$

$J\ =\ 1, 4, 9, 16, 25, \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$

$K = \frac{n}{2}, \frac{4n}{2}, \frac{9n}{2}, \frac{16n}{2}, \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$

Complexity of Example 8

$T(n) = O(n^4)$

Explanation:

$I = 1, 2, 3, 4, 5, \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots .$

$J = 1, 4, 9, 16, 25, \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots$

$K = \frac{n}{2}, \frac{4n}{2}, \frac{9n}{2}, \frac{16n}{2}, \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots$

$\boldsymbol{Sum\ of\ square\ of\ Natural\ Number}.$

$= [\boldsymbol{n(n+1)(2n+1)}]/\boldsymbol{6}$