BML MUNJAL UNIVERSITY™

Data Structures & Algorithms

What is an Algorithm?

An algorithm is a finite sequence of clearly specified, step-by-step instructions for carrying out a series of computations to solve any instance of a particular problem.

It takes some value or set of values, as input, and produces some value or set of values, as output effectively.

For each problem or class of problems, there may be many different algorithms.

For each algorithm, there may be many different implementations (programs).

Describing Algorithms

Methods of Algorithm Description

Algorithms are usually described using some form of pseudo-code (preferred notation).

Pseudo-code is a high-level description of an algorithm.

The pseudo-code syntax can be based on any structured language (Algol, Pascal, C, Ada, ...).

More structured than English prose.

Less detailed than a program.

Hides program design issues.

Pseudo-code description of finding maximum value in the array

```
/* This algorithm compute the maximum element in the array A of length n */
 Findmax(A[], n)
  {
  Pick first element in the array as 'max'   (or current max)
 // Assume the first element is the current maximum number
    For each remaining element in the array
        if it's bigger than 'max'
            replace the value of 'max' by the larger value
    // When we're all done, 'max' is the largest number we found in the array
  }
```

Convert the above pseudo-code in terms of C programming constructs

Pseudo-code description of finding maximum value in the array

/* This algorithm compute the maximum element in the array A of length n */

```
Findmax(A[], n)
{
   max = A[0] ;
    for (i = 1; i<n; i++)
          if (A[i] > max)
             max = A[i];
    return max;
}
```

The number of times the algorithm executes : n - 1 (maximum).

Complexity Analysis of Algorithms

What is algorithm analysis?

Why should we analyze an algorithm?

A problem can have many solutions.

Which solution to choose?

Algorithm analysis is the theoretical estimation, concerned with comparing algorithms based upon the amount of computing resources(mainly, time and space) that each algorithm required to execute to solve a given problem.

We want to be able to consider two algorithms and say that one is better than the other because it is more efficient in its use of computer's resources or perhaps because it simply uses fewer.

Analysis of algorithms are platform-independent.

Complexity Analysis of Algorithms

Notation for complexity analysis

Big O notation

Definition : $O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$

We write $f(n) = O(g(n))$ to indicate that a function $f(n)$ is a member of the set $O(g(n))$.

Time Complexity of an Algorithm

An algorithm can have different complexity on input instances of the same input size.

Best-case = minimum number of operations to be executed by the algorithm on input of size n.

Worst-case(bad input) = maximum number of operations to be executed by the algorithm on input of size n.

We focus on the worst-case running time.

Average case = average number of operations to be executed by the algorithm on input of size n.

Analysis Example 1

What’s the running time of the following algorithm?

/* This algorithm compute the maximum element in the array A of length n*/

```
Findmax(A[], n)
{
    max = A[0] ;
    for (i = 1; i<n; i++)
        if (A[i] > max)
          max = A[i];
    return max;
}
```

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

Analysis Example 2

For the following program fragment, give the running time

```
count = 0;
for(i = 1;  i <= n;  i=i++)
   for(j = 1;  j <= n;  j++)
      count++;
```

$T(n) = n^2 = O(n^2)$

For you to try

> Determine the running time complexity of the algorithms for adding and multiplying two 2D matrices.

> Determine the running time complexity of the algorithm for transposing a matrix.

> Given two sorted arrays A[] and B[] of size m and n respectively. Write an algorithm to merge the elements of them into another sorted array C[] of size m+n. Assume that A[] and B[] are sorted in ascending order and so C[] is also be sorted in ascending order. Determine the running time complexity of your algorithm.