Design & Analysis of Algorithms

Soharab Hossain Shaikh

BML Munjal University

Reference: Books

1. Introduction to Algorithms by T.H Cormen & Others, Pub:- PHI
2. Computer Algorithms, Introduction to Design & Analysis by Sara Basse & A.V.Gelder, Pub:- Pearson
3. Algorithm Design, J. Kleinberg & E. Tardos, Pub:- Pearson

What is an Algorithm?

An Algorithm is a finite set of instructions that, if followed, accomplishes a particular task.

Characteristics

1> Input
2> Output
3> Finiteness
4> Definiteness
5> Effectiveness

Algorithm Characteristics

Input: An algorithm has zero or more inputs from a set of inputs.

Output: An algorithm has at least one output.

Finiteness: Every algorithm must terminate after finite number of steps.

Definiteness: Each and every statement of an algorithm must be precisely and unambiguously specified.

Effectiveness: The statements of an algorithm must be sufficiently basic so that it can be carried out by a person in finite time by using paper and pencil.

Example: Euclid’s Algorithm

```
Step 1: [Take Input] > Input m and n, two positive integers. m>n

Step 2: [Find remainder] Divide m by n and let r be the remainder.
                        r=mod(m,n)

Step 3: [Is the remainder zero?]
            if r=0 then
             Print: Result =  n
             Terminate the algorithm.

Step 4: [Exchange Values of m and n]
             m=n
             n=r

Step 5: [Repeat]
             Go to Step 2.
```

Characteristics of Euclid's Algorithm

Input: There are two inputs **m** and **n** which are taken from the set of positive integers.

Output: Here, G.C.D of the two given numbers is the output.

Finiteness: In Step2 **r** is the remainder when **m** is divided by **n**. In Step4 the values of **m** and **n** are decremented. So, a decreasing sequence of integers will eventually terminate after finite number of steps.

Definiteness: Every statement of the Euclid's algorithm is precise and unambiguous.

Effectiveness: Any person using paper and pencil can carryout the algorithm in finite time.

RAM Model :
Model of Implementation Technology

>> **Uni-processor computing system.**

>> **Serial instruction execution. No concurrent processing.**

>> **Constant time instructions**

> **Arithmetic Instructions** *(add, subtract, multiply, divide, floor, ceiling)*

> **Data Movement Instructions** (Load, Store, Copy)

> **Control Instructions** (Conditional and Unconditional branch, Function Call, Return)

>> Data types are integers and floating point numbers.

Amount of work done: How to measure?

>> Actual executing time of the Algorithm?

- >> Highly machine dependent
- >> Dependent on programming language and programming style of the programmer

>> Running time of an algorithm is the number of **primitive/basic operations** done.

A particular operation fundamental to the problem under study. e.g comparing two values in a Searching algorithm, multiplication of two numbers in a Multiplication algorithm etc.

Basic Operations for different problems

Problem	**Operation**
Find **X** in an array of names	Comparison of **X** with an entry in the array
Multiply two matrices with real entries	Multiplication of two real numbers (and addition)
Sort an array of numbers	Comparison of two array entries
Traverse a Binary tree	Traversing an edge
Any non-iterative procedure	Procedure invocation

Basic Operations

>> We are intrinsically interested in the basic operation. It may be very expensive operation compared to others or may have some theoretical importance.

>> Often we are interested in the rate of growth of the time as the inputs of the algorithm get larger. So, as long as the total no. of operations is roughly proportional to the total no. of basic operations, just counting the later gives us pretty good idea of the feasibility of the algorithm on large inputs.

>> Choosing a particular basic operation ensures the study of a particular ***class of algorithm***.

Amount of work done

>> Amount of work done can not be represented by a single number.

>> The number of steps performed is not the same for all inputs.

>> The amount of work done usually depends on the input size.

Measure of size of input for a problem

Problem	**Size of Input**
Find **X** in an array of names	The number of names in the array
Multiply two matrices with real entries	The dimensions of the matrices
Sort an array of numbers	Number of entries in the array
Traverse a Binary tree	Number of nodes in the tree
Solve a system of linear equations	The number of equations or the number of unknowns

Worst Case Complexity

$W(n) = \max \{ t(I) \mid I€D_n \}$

>> **D_n** is the set of inputs of size **n**. **I** is an element of **D_n**.

>> **t(I)** is the number of basic operations performed by the algorithm on input **I**.

>> **W(n)** is the worst case complexity of the algorithm. e.g the maximum number of **basic operations** performed by the algorithm on any input size **n.**

Average Case Complexity

$$A(n) = \sum_{I \in D_n} Pr(I)t(I)$$

>> **Pr(I)** is the probability that input I occurs.

>> **D$_n$** is the set of inputs of size **n**. **I** is an element of **D$_n$**.

>> **t(I)** is the number of basic operations performed by the algorithm on input **I**.

>> **A(n)** is the average case complexity of the algorithm. e.g the number of **basic operations** performed by the algorithm (on an average) on any input size **n**.

Example: Sequential Search

```
int SeqSearch( int[] E, int n, int K )
{
1. int ans, index;
2. ans=-1;
3. for( index=0; index<n; index++)
    {
4.      if ( E[index] == K )
         {
5.          ans=index;
6.          break;
        }
    }
7. return ans;
}
```

Analysis : Sequential Search

>> **Basic operation:** Comparison of K with an array element. [at line 4.]

>> **Worst-Case Analysis: W(n)=n**, it occurs when **K** is not in the array or is the last element of the array. In both the cases **K** is compared with all the **n** elements in the array.

>> We assume that the array elements are distinct. K is equally likely to appear in any position in the array. So, Pr(I)=(1/n)

>> For $\mathbf{0 \le i < n}$, $\mathbf{I_i}$ represents the event that **K** is in the i-th position in the array. Clearly, for $\mathbf{0 \le i < n}$, $\mathbf{t(I_i)=i+1}$.

Analysis : Sequential Search

For Successful search [K is in the array]

$$>> A(n) = \sum_{i=0}^{n-1} Pr(I_i \mid_{succ})\, t(I_i)$$

$$= \sum_{i=0}^{n-1} (1/n)(i+1) = (1/n)(n(n+1)/2) = (n+1)/2$$

For Unsuccessful search (failure) [K is not in the array]

$>> t(I_{fail}) = n$, so, $A_{fail} = n$

Analysis : Sequential Search

Average Complexity

>> A(n) = Pr(succ) A(succ)(n) + Pr(fail) A(fail)(n)

= q(1/2[n+1]) + (1-q)n = n(1- q/2)+ q/2

>> **q** is the probability that **K** is in the array.

>> if **q=1**, i.e **K** is always in the array, then **A(n)= (n+1)/2**

>> if **q=1/2**, i.e 50-50 chance that **K** is not in the array, then **A(n)= 3n/4+1/4,** roughly three-fourth of the entries are examined.

Example: Matrix Multiplication

```
MatMult( A, B, C, m, n, p )
{
 for (i=0; i<m; i++)
   for (j=0; j<p; j++)
     $c_{ij}$=0;
     for (k=0; k<n; k++)
         $c_{ij}=c_{ij}+a_{ik}*b_{kj}$
}
```

>> **Basic operation:** Multiplication of the array entries.

>> **Analysis:** To compute each element of **C**, **n** multiplications are done, **C** has **mp** number of entries, so,

A(m,n,p) = W(m,n,p)=mnp

For the common case when m=n=p,

A(m,n,p) = W(m,n,p)=n^3

To be continued……