AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH (AIUB)
FACULTY OF SCIENCE & TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

LAB MANUAL 02

CSC2211 Algorithms

Summer 2019-2020

Courtesy

Part of the content of the lab manual has been taken from previous lab manuals made by different faculty members of CSE. So special thanks to all of them.

TITLE

An Introduction to Recursion and Recursive Algorithm

PREREQUISITE

- Have a clear understanding of Stack
- Know how to analyze the runtime of algorithms
- Know how to analyze the completeness of algorithms
- Have a clear understanding of return statement

OBJECTIVE

- To know about Recursion and Recursive Definition
- To know about Recursive Algorithm
- To be able to analyze Algorithms
- To be able to solve the exercises and lab work at the end of this manual

THEORY

Recursion

Recursion is the process of repeating items in a similar way. In computer science recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem. In case of computer programming, a recursion can be defined as a program that is recursive.

Recursive Definition of Function

Some functions can also be defined recursively.

Condition: The domain of the function you wish to define recursively must be a set defined recursively.

How to define function recursively: First the values of the function for the basic elements of the domain are specified. Then the value of the function at an element, say x, of the domain is defined using its value at the parent(s) of the element x.

Examples of Recursive Definition of Function

Two examples of Recursive Definition are given below. They are all on functions from integer to integer.

Example 1: The function f(n) = n! for natural numbers n can be defined recursively as follows:
The function f is the function that satisfies the following two clauses:
Basis Clause: f(0) = 0! = 1

Inductive Clause: For all-natural number n, f(n+1) = (n+1) f(n).

Note that here Extremal Clause is not necessary, because the set of natural numbers can be defined recursively and that has the extremal clause in it. So, there is no chance of other elements to come into the function being defined.

Using this definition, 3! can be found as follows:
Since 0! = 1, 1! = 1 * 0! = 1 * 1 = 1
Hence 2! = 2 * 1! = 2 * 1 = 2.
Hence 3! = 3 * 2! = 3 * 2 * 1 = 6.

Recursive Algorithm

A recursive algorithm is an algorithm which calls itself with "smaller (or simpler)" input values, and which obtains the result for the current input by applying simple operations to the returned value for the smaller (or simpler) input. More generally if a problem can be solved utilizing solutions to smaller versions of the same problem, and the smaller versions reduce to easily solvable cases, then one can use a recursive algorithm to solve that problem. For example, the elements of a recursively defined set, or the value of a recursively defined function can be obtained by a recursive algorithm.

If a set or a function is defined recursively, then a recursive algorithm to compute its members or values mirrors the definition. Initial steps of the recursive algorithm correspond to the basis clause of the recursive definition and they identify the basis elements. They are then followed by steps corresponding to the inductive clause, which reduce the computation for an element of one generation to that of elements of the immediately preceding generation.

In general, recursive computer programs require more memory and computation compared with iterative algorithms, but they are simpler and for many cases a natural way of thinking about the problem.

LAB WORK

- **Problem Set 1**
  I. Implement the Binary Search Algorithm using C++.

  II. Analyze the Binary Search Algorithm and find the time complexity.

```
binary_search (A, low, high, target):
        if (high >= low):
                mid = (low+high)/2
                if(A[mid] == target)
                        return mid
                if(A[mid] > target)
                        return binary_search (A, low, mid-1, target)
                return binary_search (A, mid+1, high, target)
```

- **Problem Set 2**
  I. Write the recurrence relation for the code below.

```
mystery (n):
        if n ≤ 1
                return 0
        else
                k=n
                for (i=1 to n)
                        do k=k+5
                return k*mystery(n/2) + 8* mystery(n/4)
```

  II. Implement in code and find mystery(10)

- **Problem Set 3**
    I. Write the recurrence relation for the code below.

```
long power (long x, long n)
        if (n == 0)
                return 1;
        if ((n % 2) == 0)
                return power (x, n/2) * power (x, n/2);
        else
                return x * power (x, n/2) * power (x, n/2);
```

    II. Implement in code and find **power** (3,5)