Course Code : BCS-042
Course Title : Introduction to Algorithm Design

july 2016 - january 2017

1. How can big O-notation be used to estimate the sum of the first n positive integers. ***(5 marks)***
Ans:

```
int sum(int n)
{
int i, sum;
1. sum = 0;                    //add 1 to the time count
2. for(i = 0; i < n; i++)      //add n + 1 to the time count
   {
3.   sum = sum + i * i;       //add n to the time count
   }
4.   return sum;               //add 1 to the time count
```

This function returns the sum from
$i = 1$ *to* n *of* i *squared, i.e.* $sum = 1^2 + 2^2 + \cdots ... + n^2$.

To determine the running time of this program, we have to count the number of statements that are executed in this procedure. The code at line 1 executes 1 time, at line 2 the **for loop** executes $(n + 1) time$, Line 3 executes n *times*, and line 4 executes 1 time. Hence
the sum is $= 1 + (n + 1) + n + 1 = 2n + 3$.

In terms of O-notation this function is $O(n)$.

2. Show how the following matrices should be multiplied using the Strassen's algorithm. X = and Y = (5 marks)
Ans:

$$AB = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 6 & 0 & 3 \\ 4 & 1 & 1 & 2 \\ 0 & 3 & 5 & 0 \end{bmatrix} \begin{bmatrix} 1 & 4 & 2 & 7 \\ 3 & 1 & 3 & 5 \\ 2 & 0 & 1 & 3 \\ 1 & 4 & 5 & 1 \end{bmatrix}$$

We define the following eight n/2 by n/2 matrices:

$$A_{11} = \begin{bmatrix} 1 & 2 \\ 0 & 6 \end{bmatrix} \quad A_{12} = \begin{bmatrix} 3 & 4 \\ 0 & 3 \end{bmatrix} \qquad B_{12} = \begin{bmatrix} 1 & 4 \\ 3 & 1 \end{bmatrix} \quad B_{12} = \begin{bmatrix} 2 & 7 \\ 3 & 5 \end{bmatrix}$$

$A_{21} = \begin{bmatrix} 4 & 1 \\ 0 & 3 \end{bmatrix}$ $A_{22} = \begin{bmatrix} 1 & 2 \\ 5 & 0 \end{bmatrix}$ $B_{21} = \begin{bmatrix} 2 & 0 \\ 1 & 4 \end{bmatrix}$ $B_{22} = \begin{bmatrix} 1 & 3 \\ 5 & 1 \end{bmatrix}$

Strassen showed how the matrix C can be computed using only 7 block multiplications and 18 block additions or subtractions (12 additions and 6 subtractions):

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$P_2 = (A_{21} + A_{22})B_{11}$$
$$P_3 = A_{11}(B_{12} - B_{22})$$
$$P_4 = A_{22}(B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{12})B_{22}$$
$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$C_{11} = P_1 + P_4 - P_5 + P_7$$
$$C_{12} = P_3 + P_5$$
$$C_{21} = P_2 + P_4$$
$$C_{22} = P_1 + P_3 - P_2 + P_6$$

The correctness of the above equations is easily verified by substitution.

$$P_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22}) = \begin{bmatrix} 1 & 2 \\ 0 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 5 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 4 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 5 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 2 & 7 \\ 8 & 2 \end{bmatrix} = \begin{bmatrix} 36 & 22 \\ 58 & 47 \end{bmatrix}$$

$$P_2 = (A_{21} + A_{22}) \times B_{11} = \begin{bmatrix} 14 & 23 \\ 14 & 23 \end{bmatrix}$$

$$P_3 = A_{11} \times (B_{12} - B_{22}) = \begin{bmatrix} -3 & 12 \\ -12 & 24 \end{bmatrix}$$

$$P_4 = A_{22} \times (B_{21} - B_{11}) = \begin{bmatrix} -3 & 2 \\ 5 & -20 \end{bmatrix}$$

$$P_5 = (A_{11} \times A_{12}) \times B_{22} = \begin{bmatrix} 34 & 18 \\ 45 & 9 \end{bmatrix}$$

$$P_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12}) = \begin{bmatrix} 3 & 27 \\ -18 & -18 \end{bmatrix}$$

$$P_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22}) = \begin{bmatrix} 18 & 16 \\ 3 & 0 \end{bmatrix}$$

$$C_{11} = P_{21} + P_4 - P_5 + P_7 = \begin{bmatrix} 17 & 22 \\ 21 & 18 \end{bmatrix}$$

$$C_{12} = P_3 + P_5 = \begin{bmatrix} 31 & 30 \\ 33 & 33 \end{bmatrix}$$

$$C_{21} = P_2 + P_4 = \begin{bmatrix} 11 & 25 \\ 19 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 22 & 38 \end{bmatrix}$$

$$C = \begin{bmatrix} 17 & 22 & 31 & 30 \\ 21 & 18 & 33 & 33 \\ 11 & 25 & 22 & 38 \\ 19 & 3 & 14 & 30 \end{bmatrix}$$

3. Using Induction method, show that for all the integers n 12 + 22 + ……..+ n2 = ***(5 marks)***

Ans:

$$P(n): 1^2 + 2^2 + 3^2 + \ldots\ldots + n^2 = \frac{n(n+1)(2n+1)}{6}.$$

Proof: (Base Step):

Consider n=1, then

$$P(1) = 1^2 = \frac{1(1+1)(2+1)}{6} = \frac{1.2.3}{6} = 1$$

Hence for n=1 it is true.

(Induction Step):

Assume P(n) is true for 'n' i.e.

$$P(n) = 1^2 + 2^2 + 3^2 + \ldots\ldots + n^2 = \frac{n(n+1)(2n+1)}{6}.$$

Now

$$P(n+1) = 1^2 + 2^2 + 3^2 + \ldots\ldots + n^2 + (n+1)^2$$

$$= \frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

$$= \frac{n(n+1)(2n+1) + 6(n+1)^2}{6}$$

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

$$= \frac{(n+1)[n^2 + n + 6n + 6]}{6}$$

$$= \frac{(n+1)(n+2)(2n+3)}{6}$$

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

Which is P(n+1).

Hence P(n+1) is also true whenever P(n) is true $\Rightarrow$ P(n) is true for all n.

4. What is a recurrence relation? Where it is used? Write recurrence relation for the followings and explain it.

(i) Binary Search Algorithm
(ii) Merge Sort Algorithm
(iii) Fibonacci Series Algorithm
(iv) Quick Sort Algorithm

(8 marks)

Ans:

A **recurrence relation** is an equation or inequality that describes a function in terms of its value on smaller inputs or as a function of preceding (or lower) terms. Like all recursive functions, a recurrence also consists of two steps:

1. Basic step: Here we have one or more constant values which are used to terminate recurrence. It is also known as **initial conditions** or **base conditions.**

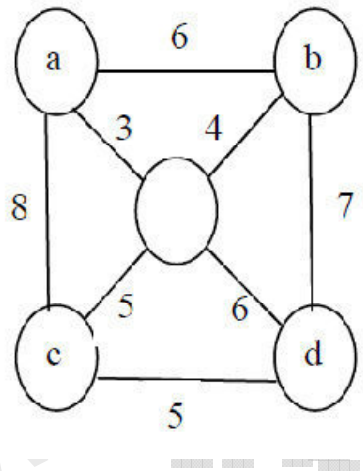
2. Recursive steps: This step is used to find new terms from the existing (preceding) terms. Thus in this step the recurrence compute next sequence from the k preceding values. This formula is called a ***recurrence relation*** (or ***recursive formula***). This formula refers to itself, and the argument of the formula must be on smaller values (close to the base value).

Recurrence relation:-

(i) $T(n) = T(n-1) + n$
(ii) $T(n)=2T(n/2) + c$
(iii) $F_n = F_{n-1} + F_{n-2}$,
(iv) $T(N) = N + T(N-1)$.

5. What are the applications of spanning tree. Write Prim's algorithm and apply it to the following graph to find out total cost of a minimum spanning tree. Show all the intermediate steps. Also, discuss time complexity of Prim's algorithm.

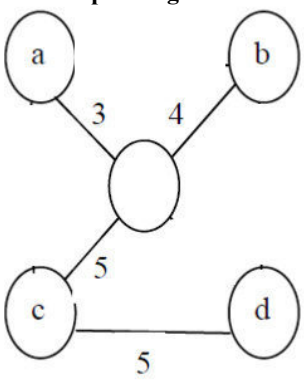
(10 marks)



Ans:

Applications of spanning tree. –
Salesman travelling problem, finding minimum Cost.

Minimum Spanning Tree



PRIM's algorithm works as follows:

1) Initially the set A of nodes contains a single arbitrary node (i.e. starting vertex) and the set T of edges are empty.
2) At each step PRIM's algorithm looks for the shortest possible edge (u, v) such that
   $u \in V\text{-}A$ *and* $v \in A$
3) In this way the edges in T form at any instance a minimal spanning tree for the nodes in A. We repeat this process until $A \neq V$.

Time complexity of PRIM's algorithm

Running time of PRIM's algorithm can be calculated as follows:

- While loop at line-3 is repeated $|V| - 1 = (n - 1)$ times.
- For each iteration of while loop, the inside statements will require $O(n)$ time.
- So the overall time complexity is $O(n^2)$.

6. Define what is an optimization problem? What are the tasks performed in Greedy method to solve optimization problem. Design a Greedy algorithm for Knapsack problem. We are given n objects and a Knapsack or a bag object. It has a weight Wi and the Knapsack has capacity m. If a fraction Xi , 0 ≤ Xi ≤ 1, of object *i* is placed into the Knapsack, profit of PiXi is earned. The objective is to fill the Knapsack in such a way as to maximize the total profit earning. ***(10 marks)***

Ans:

1) In order to solve optimization problem using greedy technique, we need the following data structures and functions: 1) A candidate set from which a solution is created. It may be set of nodes, edges in a graph etc. call this set as: **C: Set of given values or set of candidates**

2) A solution set S (where S , in which we build up a solution. This structure contains those candidate values, which are considered and chosen by the greedy technique to reach a solution. Call this set as: **S: Set of selected candidates (or input) which is used to give optimal solution.**

3) A function (say ***solution***) to test whether a given set of candidates give a solution (not necessarily optimal). 4) A selection function (say ***select***) which chooses the best candidate form C to be added to the solution set S, 5) A function (say ***feasible***) to test if a set S can be **extended** to a solution (not necessarily optimal) and 6) An objective function (say **ObjF**) which assigns a ***value*** to a solution, or a partial solution. To better understanding of all above mentioned data structure and functions, consider the minimum number of notes problem of example1. In that problem: 1) C={1, 2, 5, 10,50,100,500,1000}, which is a list of available notes (in rupees). Here the set C is a multi-set, rather than set, where the values are repeated. 2) Suppose we want to collect an amount of Rs. 283 (with minimum no. of notes). If we allow a multi-set rather than set in the sense that values may be repeated, then S={100,100,50,20,10,2,1}

3) A function **solution** checks whether a solution is reached or not. However this function does not check for the optimality of the obtained solution. In case of minimum number of notes problem, the function ***solution*** finds the sum of all values in the multi-set S and compares with the fixed amount, say Rs. 283. If at any stage S={100,100, 50}, then sum of the values in the S is 250, which does not equal to the 283, then the function ***solution*** returns "solution not reached". However, at the later stage, when S={100,100,50,20,10,2,1}, then the sum of values in S equals to the required amount, hence the function ***solution*** returns the message of the form "solution reached".

4) A function ***select*** finds the "best" candidate value (say x) from C, then this value x is tried to add to the set S. At any stage, value x is added to the set S, if its addition leads to a partial (feasible) solution. Otherwise, x is rejected. For example, In case of minimum number of notes problem, for collecting Rs. 283, at the stage when S={100, 100,50}, then first the function ***select*** try to add the Rs 50 to S. But by using a function **solution**, we can found that the addition of Rs. 50 to S will lead us a infeasible solution, since the total value now becomes 300 which exceeds Rs. 283. So the value 50 is rejected. Next, the function ***select*** attempts the next lower denomination 20. The value 20 is added to the set S, since after adding 20, total sum in S is 270, which is less than Rs. 283. Hence, the value 20 is returned by the function ***select***.

Number of objects; $n = 3$
Capacity of Knapsack; M=20
$(p_1, p_2, p_3) = (25,24,15)$
$(w_1, w_2, w_3) = (18,15,10)$

To solve this problem, Greedy method may apply any one of the following strategies:

- From the remaining objects, select the object with maximum profit that fit into the knapsack.
- From the remaining objects, select the object that has minimum weight and also fits into knapsack.
- From the remaining objects, select the object with maximum p_i/w_i that fits into the knapsack.

Approach	(x_1, x_2, x_3)	$\sum_{i=1}^{3} w_i x_i$	$\sum_{i=1}^{3} p_i x_i$
1	$(1, \frac{2}{15}, 0)$	18+2+0=20	28.2
2	$< 0, \frac{2}{3}, 1 >$	0+10+10=20	31.0
3	$< 0, 1, \frac{1}{2} >$	0+15+5=20	31.5

7. What are the properties of a shortest path. Write Bellman Ford's algorithm and apply it to the following graph. Show all the intermediate steps of the algorithm. ***(10 marks)***

Ans:

Bellman-ford algorithm, allow ***negative weight edges*** in the input graph. This algorithm either finds a shortest path from source vertex to every other vertex or detect a negative weight cycles in G, hence ***no solution***. If there is no negative weight cycles are reachable (or exist) from source vertex s, then we can find a shortest path form source vertex to every other vertex. If there exist a negative weight cycles in the input graph, then the algorithm can detect it, and hence "No solution".

Bellman-Ford Algorithm

Bellman-ford algorithm, allow ***negative weight edges*** in the input graph. This algorithm either finds a shortest path from a source vertex $s \in V$ to every other vertex $v \in V - \{s\}$

```
BELLMAN_FORD(G,w,s)

1. INITIALIZE_SIGLE_SOURCE(G,s)
2. for i ← 1 to |V| − 1
3.   do for each edge (u, v) ∈ E[G]
4.     do RELAX(u, v, w)
5. for each edge (u, v) ∈ E[G]
6.     do if (d[v] > d[u] + w(u, v)
7.       then return FALSE          // we detect a negative weight cycle
exist
7.  return TRUE
```

8.Write Merge Sort algorithm and explain the operation of the algorithm with help of the following example. 70 30 20 80 40 90 25 50 ***(7 marks)***
Ans:

See unit 2 Block-2, page-42

9. Explain the following terms :
(i) Adjacency matrix
(ii) Convected graph
(iii) Asymptotic
(iv) Time complexity
(v) Branch and broad

(10 marks)

Ans: See Block

10. Write a Pseudocode to perform a Linear Search Algorithm. Calculate the total number of comparison-operations, assignment-operations and how many times the loop will execute for finding the smallest number in an arrary ?

Ans:

```
#include<stdio.h>
#define max 10
int lsearch(int arr[],int n)
{
int i;
for(i=0;i<=max-1;i++)
{
if(arr[i]==n)
return i;
}
return -1;
}

void main()
{
int arr[max]={5,6,7,3,2,4,5,16,17,9};
int i, n,find;
clrscr();
printf("enter the number for searching");
scanf("%d",&n);
find=lsearch(arr,n);
if (find==-1)
printf("not found");
else
printf("found at the position=%d",find);

getch();
}
```