Jawaharlal Nehru Engineering College

Laboratory Manual

Design and Analysis of Algorithms

For

Third year Students
(Information Technology)

30, July 2019 – ISO 9001:2015, 14001:2015 Tech Document
Prof. Dr. S. C. Tamane, (Prof. & HoD, IT)

FORWARD

It is my great pleasure to present this laboratory manual for third year engineering students for the subject of Design and Analysis of Algorithms keeping in view the vast coverage required to design algorithms.

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9001:2015 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9001:2015 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

Prof. Dr H H Shinde
Principal

LABORATORY MANUAL CONTENTS

This manual is intended for the Third year students of Information Technology in the subject of Design and Analysis of Algorithms. This manual typically contains practical/Lab Sessions related to Design of Algorithms covering various aspects related the subject to enhanced understanding.

Although, as per the syllabus, study of designing algorithms is prescribed, we have made the efforts to cover various methods of designing algorithms covering different aspects of methods to develop algorithms.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

Vision of the department

To develop expertise of budding technocrats by imparting technical knowledge and human value based education.

Mission of the department

1. Equipping the students with technical skills, soft skills and professional attitude.
2. Providing the state of art facilities to the students to excel as competent professionals, entrepreneurs and researchers.

Program Specific Outcomes

PSO1. An ability to design, develop and implement computer programs in the areas related to Algorithms, Multimedia, Website Design, System Software, DBMS and Networking.

PSO2. Develop software systems that would perform tasks related to Research, Education and Training and/or E governance.

PSO3. Design, develop, test and maintain application software that would perform tasks related to information management and mobiles by utilizing new technologies to an individual or organizations.

Program Educational Objectives

PEO1. The graduates will utilize their **expertise** in IT industry and solve industry technological problems.

PEO2. Graduates should excel in **engineering positions** in industry and other organizations that emphasize design & implementation of IT applications.

PEO3. Graduates will be **innovators & professionals** in technology development, deployment & system implementation.

PEO4. Graduates will be pioneers in engineering, engineering management, research and **higher education**.

PEO5. Graduates will be good citizens & cultured human being with full appreciation of importance of IT **professional ethical & social** responsibilities.

Design and Analysis of Algorithms

SUBJECT INDEX

Lab Outcomes:

Students will able to:

1. Students will be able to calculate complexity of an algorithm.
2. Students will be able to select appropriate design techniques to solve real world problems.
3. Students will able to apply the dynamic programming technique to solve the problems.
4. Students will be able to apply the greedy programming technique to solve the problems.
5. Students will be able to select a proper pattern matching algorithm for given problem

DOs and DON'Ts in Laboratory:

1. Do not handle any equipment before reading the instructions/Instruction manuals

2. Read carefully the power ratings of the equipment before it is switched on whether ratings 230 V/50 Hz or 115V/60 Hz. For Indian equipments, the power ratings are normally 230V/50Hz. If you have equipment with 115/60 Hz ratings, do not insert power plug, as our normal supply is 230V/50 Hz, which will damage the equipment.

3. Observe type of sockets of equipment power to avoid mechanical damage

4. Do not forcefully place connectors to avoid the damage

5. Strictly observe the instructions given by the teacher/Lab Instructor

Instruction for Laboratory Teachers::

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

WARMUP EXCERCISES:

1. What is an algorithm?
2. Define Ω-notation, O-notation
3. What do you mean by best case efficiency?
4. What is meant by worst case?
5. Define average case efficiency.
6. Why space complexity of a program is necessary?
7. What is an algorithm design technique?
8. Define little-oh notation.
9. Compare the order of growth n! and 2!

1. Lab Exercises:

Exercise No 1: (2 Hours) – 1 Practical

WAP to implement Towers of Hanoi using recursion.

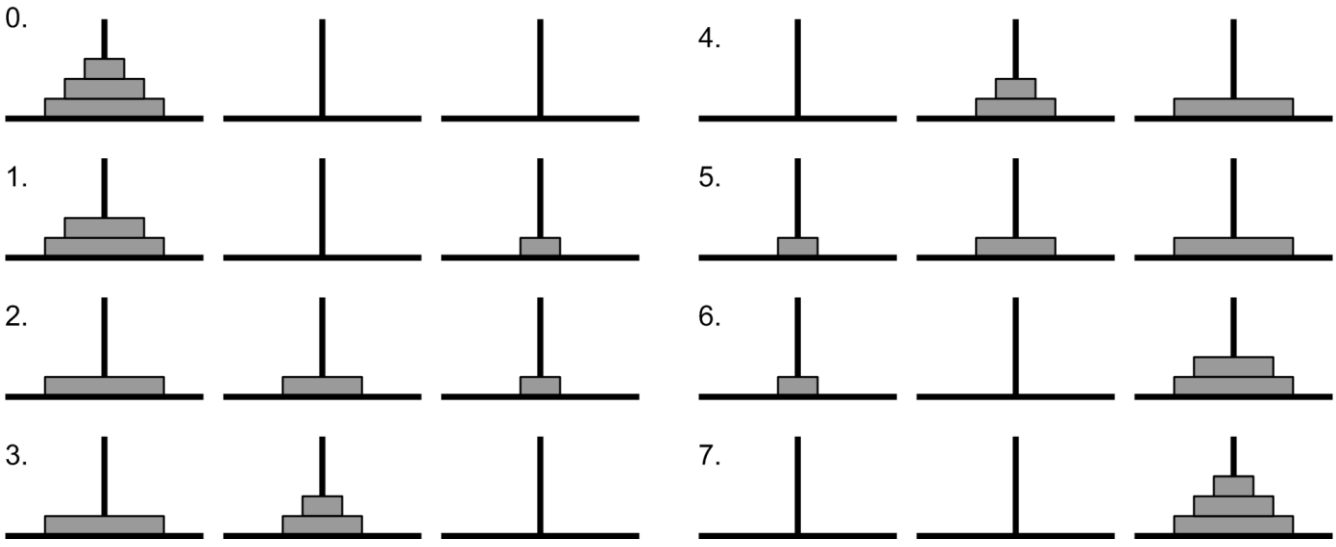
Problem: There are 3 Towers 'X', 'Y' and 'Z'. Some disks of different sizes are given which can slide onto any Tower. Initially all of those are in 'X' Tower in order of size with largest disk at the bottom and smallest disk at the top. We have to move all the disks from 'X' Tower to 'Z' Tower. At the end, 'Z' Tower will have disks in the same order of size. There are some rules:

1) Only one disk can be moved from one Tower to another Tower at a time.
2) A disk can be placed only on top of a larger one.
3) A disk can be moved from top only.

Algorithm: Towers of Hanoi

```
Algorithm TOH(n, x, y, z)
{       If (n >= 1) then
        TOH(n-1, x, z, y);
        Write("move top disk from tower", x , "to top of tower", y);
        TOH(n-1,z, y, x);
}
```

For n=3, total 7 steps are required to transfer the disks from tower 1 to 3.



Towers of Hanoi using Recursion

2. Lab Exercises:

Exercise No2: (2 Hours) – 1 Study Practical

WAP to implement permutation generator algorithm using recursion.

A permutation, also called an "arrangement number" or "order," is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has n! Permutation.
For ex. permutations of string ABC are ABC ACB BAC BCA CBA CAB

Algorithm: Permutation Generator

```
Algorithm perm(a,k,n)
{
        if (k=n) then write (a[1:n]);
        else
                for i=k to n do
                {
                        t=a[k]; a[k]=a[i]; a[i]=t;
                        Perm(a,k+1,n);
                        t=a[k]; a[k]=a[i]; a[i]=t;
                }
}
```

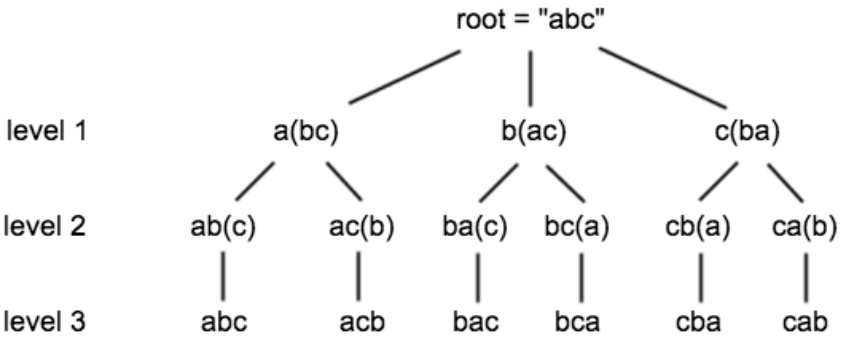
Output:

```
permutations ([])
 []

permutations ([1,])
[1]

permutations ([1,2])
[1, 2]
[2, 1]

permutations ([1,2,3])
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```



3. Lab Exercises:

Exercise No 3: (2 Hours) – 1 Study Practical

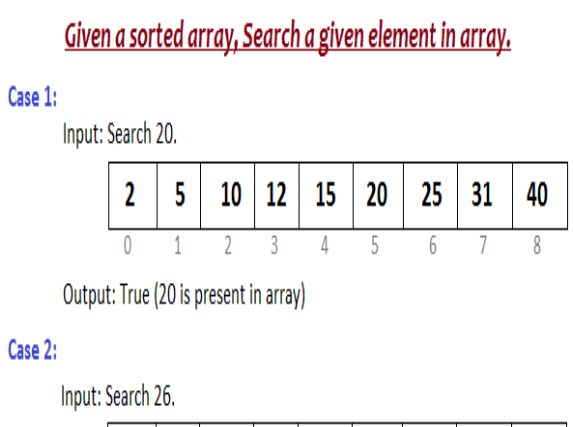
WAP to implement binary search algorithm.

Let a_i, 1 <= i <= n be a list of elements which are sorted in non-decreasing order. Consider the problem of determining whether a given element x is present in the list. In case x is present, we are to determine a value j such that $a_j = x$. If x is not in the list then j is to be set to zero. Divide-and conquer suggests breaking up any instance $I = (n, a_1, \ldots, a_n, x)$ of this search problem into sub instances. One possibility is to pick an index k and obtain three instances: I1 = $(k - 1, a_1, \bullet\bullet\bullet, a_{k-1}, x)$, $I2$ = $(1, a_k, x)$, and I3 = $(n - k, a_{k+1}, \ldots, a_n, x)$. The search problem for two of these three instances is easily solved by comparing x with a_k. If $x = a_k$ then $j = k$ and I1 and I3 need not be solved. If $x < ak$ then for $I2$ and I3, $j = 0$ and only I1 remains to be solved. If $x > ak$ then for I1 and 12, ,j = 0 and only 13 remains to be solved. After a comparison with a_k, the instance remaining to be solved (if any) can be solved by using this divide-and-conquer scheme again. If k is always chosen such that a_k is the middle element (i.e. k = L$(n + 1)/2$J then the resulting search algorithm is known as binary search.

Algorithm has three inputs, A, n and x, and one output, j. The while loop continues processing as long as there are more elements left to check. The case statement permits the selection of the three alternatives. The first two conditions are checked for, and if they do not occur, the "else clause" is automatically executed. At the conclusion of the procedure either j = 0 if x is not present, or $A(j) = x$.

Algorithm: Binary Search

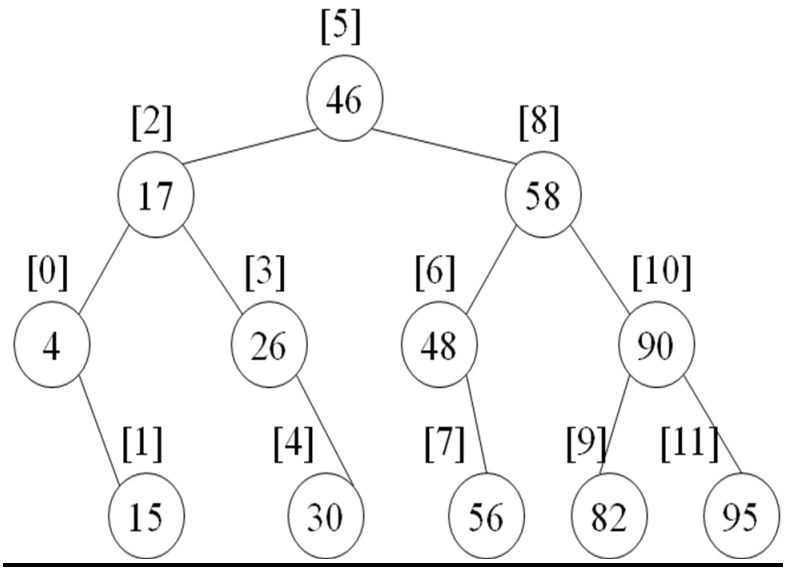
```
Algorithm BINSRCH(a, n, x)
//given an array A(i:l) of elements in non-decreasing order,
//determine if x is present, and if so, set j such that x = A(j)
//else return 0
{
low = 1; high = n
while (low <= high) do
{       mid = low + high)/2;
        If ( x < a[mid] ) then high = mid – 1;
        else if ( x > a[mid] ) then low = mid + 1;
        else return mid;
}
Return 0;
}
```



Algorithm: Binary search using one comparison per cycle

```
Algorithm binsrch1 (a, i, l, x)
{
If ( l=i ) then
{ if (x=a[i]) then return i;
        else return 0;
}
else
{
        mid = (i+l)/2;
        if (x=a[mid]) then return mid;
        else If ( x < a[mid] ) then return binsrch1(a,i,mid-1,x);
        else return binsrch1(a,mid+1,l,x);
}
}
```

4, 15, 17, 26, 30, 46, 48, 56, 58, 82, 90, 95



4. Lab Exercises:

Exercise No 4: (2 Hours) – 1 Practical

WAP to implement maxmin algorithm using DnC method.

The problem is to find the maximum and minimum items in a set of *n* elements. Though this problem may look so simple as to be contrived, it allows us to demonstrate divide-and-conquer in a simple setting. A divide-and-conquer algorithm for this problem would proceed by dividing any instance *I = (n,* A(l), ... , *A(n))* into smaller instances. For example we might divide l into the two instances l1 = (n/2, A(l), .. . ,*A(*n/2)) and *12 = (n - n/2, A (n/2* + 1), ... , *A(n)).* If MAX(I) and MIN(I) are the maximum and minimum of the elements in *I* then MAX(I) = the larger of MAX(I1) and MAX(I2), and MIN(I) == the smaller of MIN(I1) and MIN(I2). If *I* contains only one element then the answer can be computed without any splitting.

Algorithm: Recursively finding the maximum and minimum

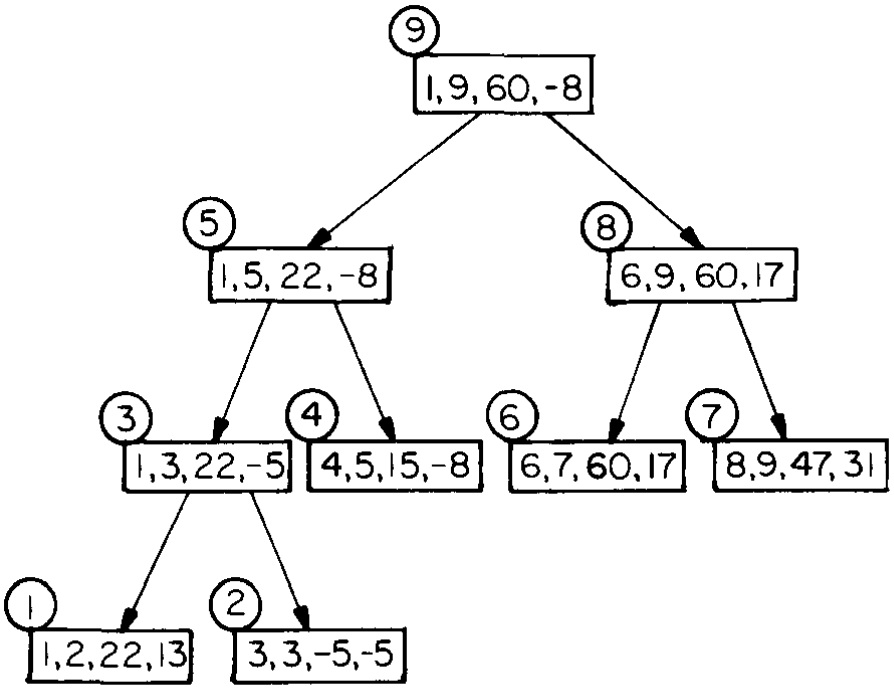
Algorithm MAXMIN(i, j, max, min)
//A is a global array containing n numbers in A(l), ... , A(n)//
//Parameters i, j are integers: 1 <= i<= n. The effect is to assign to max and min the largest and smallest values in A(i:j) //

```
{
If (i=j) then max=min=a[i];
else
if (i = j – 1) then
        { if (A(i) < A(j) then
                { max = A(j); min = A(i);}
          else
                { max = A(i);min = A(j);}
        }
else
{       mid = (i + j)/2;
        MAXMIN(i, mid, max, min) ;
        MAXMIN(mid + 1,j, max1, min1) ;
        If ( max<max1) then  max=max1;
        If ( min > min1)then  min= min1;
}
}
```

Example:

A:	(1)	*(2)*	*(3)*	*(4)*	(5)	(6)	(7)	*(8)*	(9)
	22	13	-5	-8	15	60	17	31	47

Max= 60
Min= -8



5. *Lab Exercises:*

Exercise No 5: (2 Hours) – 1 Study Practical

WAP to implement merge sort algorithm using DnC method.

We shall assume that the elements are to be sorted in non-decreasing order. Given a sequence of *n* elements (also called keys) A(l), … , *A(n)* the general idea is to imagine them split into two sets A(l), … , *A(*n/2) and *A(*n/2 + 1), … , *A(n).* Each set is individually sorted and the resulting sequences are merged to produce a single sorted sequence of *n* elements. Thus we have another ideal example of the divide-and-conquer strategy where the splitting is into two equal size sets and the combining operation is the merging of two sorted sets into one.

Algorithm: Mergesort

```
Algorithm MERGESORT(low, high)
//A(low : high) is a global array //
//small (p) is true if there is only one element to sort.//

if (low < high ) then
{
        mid = (low + high)/2;  //find where to split the set//
        MERGESORT(low, mid);
        MERGESORT(mid + 1, high);
        MERGE(low, mid, high);
}
```

Algorithm: Merging two sorted sets using auxiliary storage

```
Algorithm MERGE(low, mid, high)
// A(low:high) is a global array containing two sorted subsets //
//in A(low:mid) and in A(mid + l:high). //
//The objective is to merge these sorted sets into a single sorted set //
// residing in A(low:high). B is an auxiliary global array //
{

h = low; i = low; j= mid+ 1;
while (( h <= mid) and (j <= high) do
{
        If (A(h) <= A(j)) then
        {
                B(i) = A(h);
```

```
                H= h + 1 ;
        }
        else
        {
                B(i) = A(j);
                J= j + 1 ;
        }

i - i + 1 ;
}
If (h > mid) then
        For k= j to high do
{
B[i]=a[k];
I=i+1;
}
Else
For k=h to mid do
{
B[i]=a[k];
I=i+1;
 }

For k=low to high do a[k]=b[k];
}
```

Example:
A = (310, 285, 179, 652, 351, 423, 861, 254, 450, 520)
Pictorially the file can now be viewed as

(310 | 285 | 179 | 652, 351 | 423, 861, 254, 450, 520)

with the vertical bars indicating the boundaries of subfiles. A(l) and A(2) are merged to yield

(285, 310 | l79 | 652, 351 | 423, 861, 254, 450, 520)

Then A(3) is merged with A(1:2) producing

(179, 285, 310 | 652, 351 | 423, 861, 254, 450, 520)

Next, elements A(4) and A(5) are merged

(179, 285, 310 | 351, 652 | 423, 861, 254,450, 520)

followed by the merging of A(1:3) and A(4:5) to give

(179, 285, 310, 351, 652 | 423, 861, 254, 450, 520)

At this point the algorithm has returned to the first invocation of MERGESORT and it is about to process the second recursive call. Repeated recursive calls are invoked producing the following subfiles:

(179, 285, 310, 351, 652 | 423 |86l | 254 | 450, 520)

A(6) and A(7) are merged and then A(8) is merged with A(6:7) giving

(179, 285, 310, 351, 652 | 254, 423, 861 | 450, 520)

Next A(9) and A(10) are merged followed by A(6:8) and A(9:10)

(179, 285, 310, 351, 652 | 254, 423, 450, 520, 861)

At this point there are two sorted subfiles and the final merge produces the fully sorted result

(179, 254, 285, 310, 351, 423, 450, 520, 652, 861)

6. Lab Exercises:

Exercise No 6: (2 Hours) – 1 Study Practical

WAP to implement quick sort algorithm using DnC method.

The divide-and-conquer approach may be used to arrive at an efficient sorting method different from mergesort. In mergesort, the file A(l:n) was divided at its midpoint into subfiles which were independently sorted and later merged. In quicksort, the division into two subfiles is made such that the sorted subfiles do not need to be later merged. This is accomplished by rearranging the elements in A(l :n) such that *A(i)<= A(j)* for all *i* between 1 and *m* and all j between *m* + 1 and *n* for some *m,* 1<= *m* <= *n.* Thus, the elements in A(l:m) and A(m + l:n) may be independently sorted. No merge is needed. The rearrangement of the elements is accomplished by picking some element of *A,* say *t = A(s),* and then reordering the other elements so that all elements appearing before *t* in A(l:n) are less than or equal to *t* and all elements appearing after *t* are greater than or equal to *t.* This rearranging is referred to as partitioning.

Algorithm QuickSort

```
procedure PARTITION(m, p)
{
integer m, p, i; global A(m: p)
v = A(m); i= m //A(m) is the partition element//
loop
loop i= i + 1 until A(i)>= v repeat     // i moves left to right//
loop p= p - 1 until A(p)<= v repeat    //p moves right to left//
if  i < p
        then call INTERCHANGE(A(i),A(p)) //exchangeA(i) andA(p)//
   else exit
end if
repeat
A(m)= A(p);A(p)= v
end PARTITION        //the partition element belongs at position p//
```

```
procedure QUICKSORT(p, q)
// sorts the elements A (p ), ... , A (q) which reside in the global array A (1 :n) into
ascending order;//
// A (n + 1) is considered to be defined and must be >= all elements in A (p :q ); //
integer p, q; global n, A (l:n)
if p < q   then j= q + 1
call PARTITION(p, j)
call QUICKSORT(p, j - 1)    //j is the position of the partitioning element //
call QUICKSORT(j + 1, q)
endif
end QUICKSORT
```

Example:

```
(1)  (2)  (3)  (4)  (5)  (6)  (7)  (8)  (9)  (10)  i  p
65   70   75   80   85   60   55   50   45   +∞    2  9
     I·····................................................ I
65   45   75   80   85   60   55   50   70   +∞    3  8
          I ·..................................................I
65   45   50    80  85   60   55   75   70   +∞    4  7
               I ·.............................I
65   45   50   55   85   60   80   75   70   +∞    5  6
                         I····· I
65   45   50   55   60   85   80   75   70   +∞    6  5
I·..... I
60   45   50   55   65   85   80   75   70   +∞
```

7. Lab Exercises:

Exercise No 7: (2 Hours) –1 Study Practical

WAP to implement Knapsack algorithm using greedy method.

We are given *n* objects and a knapsack. Object *i* has a weight wi and the knapsack has a capacity *M.* If a fraction *xi,* 0 <=*xi* <= 1, of object *i* is placed into the knapsack then a profit of *PiXi* is earned. The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Since the knapsack capacity is *M,* we require the total weight of all chosen objects to be at most *M.*

Algorithm Knapsack:

procedure *GREEDY_KNAPSACK(P, W, M, X,* n)
// P(l :n) and *W(l* :n) contain the profits and weights respectively of the n objects ordered so that *P(i)/ W(i)>= P(i + l)/ W(i* + 1), *M is* the knapsack size and X(l :n) is the solution vector//

real P(l:n), W(l:n), X(l:n), *M, cu;*
integer *i, n* ;
X= 0 //initialize solution to zero//
cu = M // *cu* = remaining knapsack capacity//
for i = 1 *to n* do
if *W(i) > cu* then exit endif
X(i) = 1
cu = cu - W(i)
repeat
if *i<= n* then *X(i)= cu/ W(i)* endif
end *GREEDY _KNAPSACK*

Example:

Consider the following instance of the knapsack problem:
n = 3, *M* = 20, *(p1, p2, p3)* = (24, 15, 25) and (w1, w2, w3) = (15, 10, 18).

(X1, x2, x3) = (1, 1/2, 0)
M = 20, P= 31.5

8. Lab Exercises:

Exercise No 8: (2 Hours) –1 Study Practical

WAP to implement Job sequencing with deadlines using greedy method.

We are given a set of n jobs. Associated with job i is an integer deadline di >= 0 and a profit pi >= 0. For any job i the profit pi is earned iff the job is completed by its deadline. In order to complete a job one has to process the job on a machine for one unit of time. Only one machine is available for processing jobs. A feasible solution for this problem is a subset, J, of jobs such that each job in this subset can be completed by its deadline. The value of a feasible solution J is the sum of the profits of the jobs in J. An optimal solution is a feasible solution with maximum value.

Algorithm Job Sequencing with Deadlines:

procedure *GREEDY _JOB(D, J,* n)
// J is an output variable. It is the set of jobs to be completed by their deadlines //
1 j={1}
2 for *i* = 2 to *n* do
3 if all jobs in *J* U { i} can be completed by their deadlines
Then J = JU {i}
4 end if
5 repeat
6 end *GREEDY _JOB*

Example:
Let n = 4, (p1, p2, p3, p4) = (100, 27, 15, 10) and (d1, d2, d3, d4) = (2, 1, 2, 1).

Processing sequence = (2, 1)

9. *Lab Exercises:*

Exercise No 9: (2 Hours) – 1 Study Practical

WAP to implement graph traversal technique: BFS.

In breadth first search, start at a vertex v and mark it as having been reached (visited). The vertex v will at this time be said to be unexplored. A vertex will be said to have been explored by an algorithm when the algorithm has visited all vertices adjacent from it. All unvisited vertices adjacent from v are visited next. These are new unexplored vertices. Vertex v has now been explored. The newly visited vertices haven't been explored and are put onto the end of a list of unexplored vertices. The first vertex on this list is the next to be explored. Exploration continues until no unexplored vertex is left. The list of unexplored vertices operates as a queue and may be represented using any of the standard queue representations. Procedure BFS describes the details of the search. It makes use of two algorithms DELETEQ(v, Q) which deletes a vertex from the queue Q and returns, in v, the index and the vertex deleted and ADDQ(v, Q) which adds vertex v to the rear of queue Q.

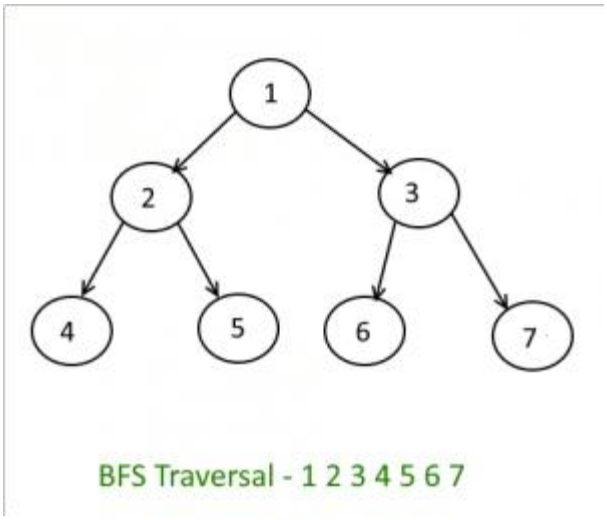
Algorithm BFS:

```
Procedure BFS(v)
// A breadth first search of G is carried out beginning at vertex v. All vertices
visited are marked as VISITED(i) = 1. The graph G and array VISITED are global
and VISITED is initialized to zero.//

 VISITED(v) = 1; u = v;
 initialize Q to be an empty queue     // Q is a queue of unexplored vertices//
 loop
        for all vertices w adjacent from u do
                if VISITED(w) = 0 then call ADDQ(w, Q)        //w is unexplored//
                        VISITED(w) = 1;
        endif
        repeat

        if Q is empty then return endif     //no unexplored vertex //
        call DELETEQ(u, Q)                  //get first unexplored vertex //
 repeat
endBFS
```

```
procedure BFT( G, n)
/ /breadth first traversal of G //
declare VISITED(n)
for i= 1 to n do    // mark all vertices unvisited //
VISITED(i) = 0
repeat
for i = 1 to n do    // repeatedly call BFS //
if VISITED(i) = 0 then call BFS(i) endif
repeat
end BFT
```



10. Lab Exercises:

Exercise No 10: (2 Hours) – 1 Study Practical

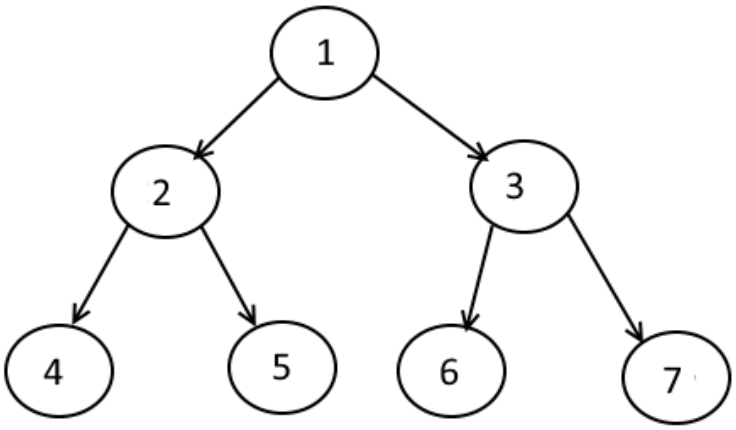
WAP to implement graph traversal technique: DFS.

A depth first search of a graph differs from a breadth first search in that the exploration of a vertex *v* is suspended as soon as a new vertex is reached. At this time the exploration of the new vertex *u* begins. When this new vertex has been explored, we continue to explore *v*. The search terminates when all reached vertices have been fully explored.

Algorithm DFS:

procedure *DFS(v)*
// Given an undirected (directed) graph G = *(V, E)* with *n* ver*tices* and an array VISITED (n) initially set to zero, this algorithm visits all vertices reachable from *v*. G and VISITED are global. //

VISITED(v) = 1
for each vertex *w* adjacent from *v* do
if *VISJTED(w)* = 0 then call *DFS(w)* endif
repeat
endDFS



DFS Traversal - 1 2 4 5 3 6 7

11. Lab Exercises:

Exercise No 11: (2 Hours) – 1 Study Practical

WAP to implement Sum of Subsets Algorithm

(Sum of subsets) Given $n + 1$ positive numbers: wi, 1<=i<=n and *M,* this problem calls for finding all subsets of the wi whose sum is *M.* For example, if $n = 4$, (w1, w2, W3, W4) = (11, 13, 24, 7) and $M = 31$ then the desired subsets are (11, 13, 7) and (24, 7). Rather than represent the solution vector by the wi which sum to *M,* we could represent the solution vector by giving the indices of these wi. Now the two solutions are described by the vectors (1, 2, 4) and (3, 4). In general, all solutions are k-tuples *(x1, x2, … , Xk),* $1 <= k <= n$ and different solutions may have different size tuples. The explicit constraints require $xi \in \{j \mid j$ is an integer and $1 <= j <= n\}$. The implicit constraints require that no two be the same and that the sum of the corresponding wi be *M.* Since we wish to avoid generating multiple instances of the same subset (e.g. (1, 2, 4) and (1, 4, 2) represent the same subset).

Algorithm Sum of Subsets:

procedure *SUMOFSUB(s, k,* r)

1 global integer *M, n;* global real W(l:n); global Boolean X(l:n)
2 real *r, s;* integer *k, j*
// generate left child. Note that s + *W(k)* <= *M* because B_{k-1} = true //
3 *X(k) =1*
4 if s + *W(k)* = *M* //subset found//
5 then print (X(j), *j* - 1 to k)
// there is no recursive call here as W(j) > 0, 1 <= *j* <= *n* //

6 else
7 *if* s+ *W(k)* + *W(k* + 1) <= M then // *Bk*= true//
8 call *SUMOFSUB(s* + *W(k), k* + 1, *r* - *W(k))*
9 endif
10 endif
//generate right child and evaluate *Bk*//
11 *If s* + *r* - *W(k)* >= *M* and *s* + *W(k* + 1) <=*M* // *Bk* = true //
12 then X(k) = 0
13 *call SUMOFSUB(s, k* + 1, *r* - *W(k))*
14 endif
15 end *SUMOFSUB*

12. Quiz on the subject:

Quiz should be conducted on tips in the laboratory, recent trends and knowledge of the subject. The quiz questions should be formulated such that questions are normally from the scope outside of the books. However twisted questions and self formulated questions by the faculty can be asked but correctness of it is necessarily to be thoroughly checked before the conduction of the quiz.

13. Conduction of Viva-Voce Examinations:

Teacher should conduct oral exams of the students with full preparation. Normally, the objective questions with guess are to be avoided. To make it meaningful, the questions should be such that depth of the students in the subject is tested. Oral examinations are to be conducted in co-cordial environment amongst the teachers taking the examination. Teachers taking such examinations should not have ill thoughts about each other and courtesies should be offered to each other in case of difference of opinion, which should be critically suppressed in front of the students.

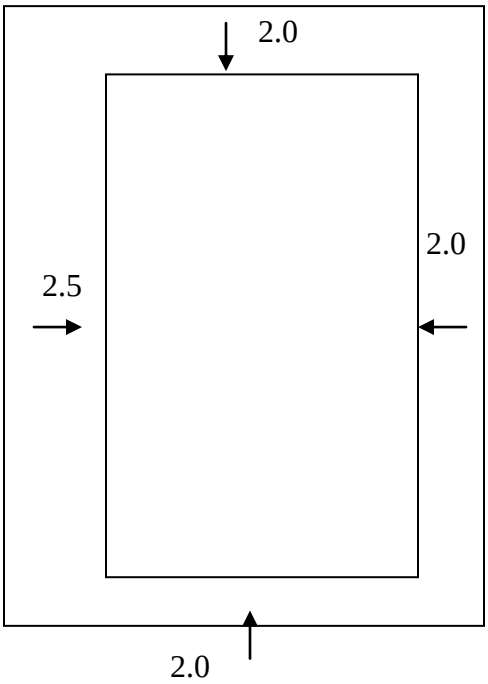
14. Submission:

Document Standard:

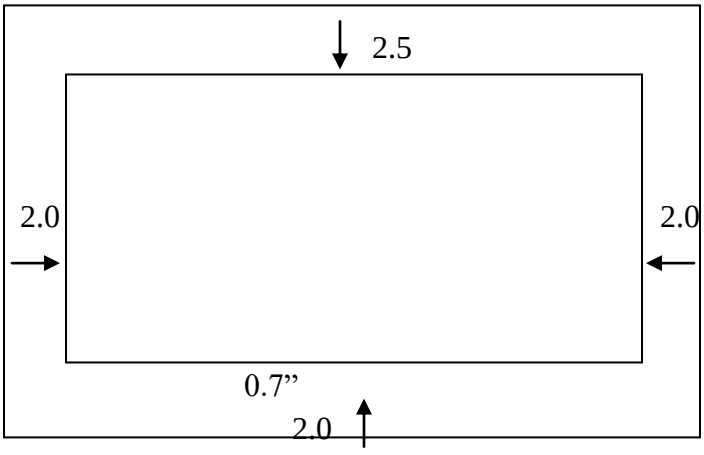
A] Page Size	A4 Size
B] Running text	Justified text
C] Spacing	1 Line

D] Page Layout and Margins (Dimensions in Cm)

Normal Page



Horizontal



Description	Font	Size	Boldness	Italics	Underline	Capitalize
College Name	Arial	24	-----	------	Yes	----------
Document Title	Tahoma	22	-----	------	---------	----------
Document Subject	Century Gothic	14	-----	------	---------	Capital
Class	Bookman old Style	12	-----	------	---------	----------
Document No	Bookman old Style	10	-----	------	---------	----------
Copy write info	Bookman old Style	9	-----	------	---------	----------
Forward heading	Bookman old Style	12	-----	------	Yes	Capital
Forward matter	Bookman old Style	12	-----	------	---------	----------
Lab man Contents title	Bookman old Style	12	-----	------	Yes	Capital
Index title	Bookman old Style	12	Yes	------	Yes	Capital
Index contents	Bookman old Style	12	-----	------	---------	----------
Heading	Tahoma	14	Yes	Yes	Yes	----------
Running Matter	Comic Sans MS	10	-----	------	---------	----------

15. Evaluation and marking system:

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that students are faced with unjust circumstances.