Laboratory Manual

For

Algorithm Analysis and Design

(MF 101)

M.Tech(IT)

SEM I

June 2015

Faculty of Technology
Dharmsinh Desai University
Nadiad.
www.ddu.ac.in

Table of Contents

EXPERIMENT-9

EXPERIMENT-10

LABWORK BEYOND CURRICULA

EXPERIMENT-11

Experiment-12

Sample Experiment

1 AIM: Implement Merge Sort .

2 TOOLS/APPARATUS: Turbo C or gcc / gprof /javac compiler in linux.

3 STANDARD PROCEDURES:

3.1 Analyzing the Problem:

Using divide and Conquer approach in merge sort method, we have to sort n number of data.

For Example: If we have input data like

38 27 43 3 9 82 10

Stepwise solution will be:

38 27 43 3 | 9 82 10
38 27 | 43 3 | 9 82 | 10
38 | 27 | 43 |3 |9 |82 | 10
27 38 | 3 43 | 9 82 | 10
3 27 38 43 | 9 10 82
3 9 10 27 38 43 82

3.2 Designing the Solution:

Algorithm of Merge Sort:

```
Algorithm Mergesort(low,high)
//a[low:  high] is a global array to be sorted.
//Small(P) is true if there is only one element to sort.In this case the list is already
sorted.
{
        If(low<high) then //if there are more then one element
        {
                //Divide P into subproblems.
                        //Find where to split the set.
                        Mid:=(low+high)/2;
                //Solve the subproblems.
                Mergesort(low,mid);
                Mergesort(mid+1,high);
                //Combine the solution.
                Merge(low,mid,high);
        }
}
Algorithm Merge(low,mid,high)
//a[low:high] is a global array containing two sorted subsets in a[low:mid]
// and in a[mid+1:high].The goal is to merge these two serts into a single 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];
}
```

3.3 Implementing the Solution

3.3.1 Writing Source Code:

```
#include<iostream.h>
#include<conio.h>
void merge(int *,int,int,int);//shows how data will be
divided,getting sorted and merged.
void ms(int *,int ,int );//Divide data into subparts and merge them
void main()
{    clrscr();
       int n, a[10], i;//a[]will store input.
       cout<<"Enter the number of elemets =";
       cin>>n;
       cout<<"Enter the elements of arry for sorting = ";
       for(i=0;i<n;i++)
       {
              cin>>a[i];
       }
       ms(a,0,n-1);
       cout<<"Sorted elements : ";
       for(i=0;i<n;i++)
       cout<<a[i]<<' ';
    getch();
 }
 void ms(int *a,int low,int high)
 {
```

```
        //low indicates index of first data and high indicates index of
    last data.
        if(low<high)
        {
            int mid=(low+high)/2;
            ms(a,low,mid);
            ms(a,mid+1,high);
            merge(a,low,mid,high);
        }
    }
    void merge(int *a, int low, int mid,int high)
    {
        int temp[10];
        int h=low,i=low,j=mid+1;
        while(i<=mid&&j<=high)
        {
            if(a[i]<=a[j])
            {
                temp[h++]=a[i++];
}
  else
                temp[h++]=a[j++];

        if(i>mid)
        {
            for(int k=j;k<=high;k++)
            temp[h++]=a[k];
        }

    else
        {       for(int k=i;k<=mid;k++)
            temp[h++]=a[k];

        }
```

```
                    for(int k=low;k<=high;k++)
                    {
                            a[k]=temp[k];
                    }
            }
```

3.3.2 Compilation /Running and Debugging program

In linux,

```
gcc mergesort.c
./a.out
Enter the number of elemets =5
Enter the elements of arry for sorting = 2 3 1 5 4
Sorted elements :1 2 3 4 5
```

3.4 Testing :

Test setup and results:-

Test data characteristics		Actual time taken by algorithm to solve problem
Type	Size	
Almost sorted	10000	0.125
Unsorted	10000	0.220
Almost sorted	15000	0.203
Unsorted	15000	0.315
Almost sorted	20000	0.281
Unsorted	20000	0.381
Almost sorted	25000	0.343
Unsorted	25000	0.451
Almost sorted	35000	0.781
Unsorted	35000	0.841

4 Conclusions

Time Complexity Of Merge Sort in best case is(when all data is already in sorted form):O(n) Time Complexity Of Merge Sort in worst case is: O(n logn) i.e when the data is unsorted .

Time Complexity Of Merge Sort in average case is: O(n logn). Practically also it was observed same . Thus it should be preferred if we expect data to be almost sorted.

Required Software/ Software Tool

-Linux Operating System and/ or Windows Operating System

-C /C++ / java / Any high level programming language

Common Procedure:

1) Design algorithm for respective problem given.
2) Select a barometer instruction and find the time complexity in best,worst and average cases. Write this apriori analysis in your file also .

3) Implement algorithm using C/C++/java/ etc language.

4) Do the empirical (posteriori) analysis of the algorithm.
i.e Analyze the actual running time program.

Empirical analysis(posteriori) :-

Type of Data sample	Data size	Data characteristic	Actual time taken
Sorted data ex:- 1,2,3,6,8,14,45	100	-best case	0.005 msec
Reverse sorted data- ex:10,2,4,1	150	worst case	1.008 msec

4) Give your conclusion regarding the algorithm.

When to use and when to NOT use the algorithm. Time complexity and space complexity based on empirical restuls obtained.

EXPERIMENT - 1

Aim: Introduction to gnu profiler tool.
Procedure:
[1]. Write the following programs in "C" –
Search elements in large unsorted list ..
Find factorial of given number.

[2]. Compile the program (using gcc filename-pg option.)

The first step in generating profile information for your program is to compile and link it with profiling enabled.

To compile a source file for profiling, specify the `-pg' option when you run the compiler. (This is in addition to the options you normally use.)

To link the program for profiling, if you use a compiler such as cc to do the linking, simply specify `-pg' in addition to your usual options. The same option, `-pg', alters either compilation or linking to do what is necessary for profiling. Here are examples:

```
cc -g -c myprog.c utils.c -pg
cc -o myprog myprog.o utils.o -pg
```

The `-pg' option also works with a command that both compiles and links:

```
cc -o myprog myprog.c utils.c -g -pg
```

[3]. Execute output file using ./a.out option.

Once the program is compiled for profiling, you must run it in order to generate the information that gprof needs. Simply run the program as usual, using the normal arguments, file names, etc. The program should run normally, producing the same output as usual. It will, however, run somewhat slower than normal because of the time spent collecting and the writing the profile data.

The way you run the program--the arguments and input that you give it--may have a dramatic effect on what the profile information shows. The profile data will describe the parts of the program that were activated for the particular input you use. For example, if the first command you give to your program is to quit, the profile data will show the time used in initialization and in cleanup, but not much else.

Your program will write the profile data into a file called `gmon.out' just before exiting. If there is already a file called `gmon.out', its contents are overwritten. There is currently no way to tell the program to write the profile data under a different name, but you can rename the file afterward if you are concerned that it may be overwritten.

In order to write the `gmon.out' file properly, your program must exit normally: by returning from main or by calling exit. Calling the low-level function _exit does not write the profile data, and neither does abnormal termination due to an unhandled signal.

The `gmon.out' file is written in the program's *current working directory* at the time it exits. This means that if your program calls chdir, the `gmon.out' file will be left in the last directory your program chdir'd to. If you don't have permission to write in this directory, the file is not written, and you will get an error message.

[4]. Execute gprof<filename to see profiling result .

After you have a profile data file `gmon.out', you can run gprof to interpret the information in it. The gprof program prints a flat profile and a call graph on standard output. Typically you would redirect the output of gprof into a file with `>'.

You run gprof like this:

gprof *options* [*executable-file* [*profile-data-files*...]] [> *outfile*]

Here square-brackets indicate optional arguments.

If you omit the executable file name, the file `a.out' is used. If you give no profile data file name, the file `gmon.out' is used. If any file is not in the proper format, or if the profile data file does not appear to belong to the executable file, an error message is printed.

You can give more than one profile data file by entering all their names after the executable file name; then the statistics in all the data files are summed together.

The order of these options does not matter.

EXPERIMENT - 2

Aim: Implement following programs using divide and conquer technique –

(1) Binary search

Input:- sorted array of elements example: [a,b,c,d,e,f……] , value to be searched.

Procedure :- use appropriate divide and conquer algorithm .

- In binary search, an array a[1:n] of elements in non decreasing order ,n>0
- Binary search determines whether element x is present and if so return j such that x=a[j]
- Else return 0.

Output:- location of element if found else error message.

(1) Quick sort /Merge sort

Input:- Unsorted array of elements example: [b,a,x,e,f……] , and order needed [ascending/descending]

Output:- sorted data according to ascending order example :- [a,b,c,d……]

Procedure: - use appropriate divide and conquer algorithm.

For merge sort:

- In Merge Sort a[low:high] is a global array containing two sorted subsets in a[low:mid] and in a[mid+1:high].
- The goal is to merge these two subsets into a single set residing in a[low:high].
- Take b as auxilary array.
- Mid value can be found by using partition function.
- In Partition function, a[m],a[m+1],….a[p-1] the elements are rearranged in such a manner that if initially t=a[m],then after completion a[q] = t for some q between m and p-1,a[k]<=t for m<=k<q and a[k]>=t for q<k<p.

For quick sort:

Suppose we are sorting a subarray array[p..r] .

- **Divide** by choosing any element in the subarray array[p..r]. Call this element the **pivot**. Rearrange the elements in array[p..r] so that all other elements in array[p..r] that are less than or equal to the pivot are to its left and all elements in array[p..r] are to the pivot's right. We call this procedure **partitioning**. So, for example, if the subarray consists of [9, 7, 5, 11, 12, 2, 14, 3, 10, 6], and we choose 6 as the pivot. After partitioning, the subarray might look like [5, 2, 3, 6, 12, 7, 14, 9, 10, 11]. Let q be the index of where the pivot ends up.

- Conquer by recursively sorting the subarrays array[p..q-1] (all elements to the left of the pivot, which must be less than or equal to the pivot) and array[q+1..r] (all elements to the right of the pivot, which must be greater than the pivot).

EXPERIMENT – 3

Aim: Implement following programs using greedy technique :-

i) make change

Problem Make a change of a given amount using the smallest possible number of coins. Start with nothing.

Procedure:

- Make change for n units using the least possible number of coins.
- C is candidate sets which contains different amount of coins.
- S is the solution set which contains total number of coins.
- At every stage without passing the given amount.
- Add the largest to the coins already chosen.

Example Make a change for 2.89 (289 cents) here n = 2.89 and the solution contains 2 dollars, 3 quarters, 1 dime and 4 pennies.

ii) Knapsack (fractional) -

Procedure:

- Assume knapsack holds weight W and items have value v_i and weight w_i

- Rank items by value/weight ratio: v_i / w_i
  - Thus: $v_i / w_i \geq v_j / w_j$, for all $i \leq j$
- Consider items in order of decreasing ratio
- Take as much of each item as possible

Example :- Knapsack Capacity W = 30 and

Item	A	B	C	D
Value	50	140	60	60
Size	5	20	10	12
Ratio	10	7	6	5

Solution:

- All of A, all of B, and ((30-25)/10) of C (and none of D)
- Size: 5 + 20 + 10*(5/10) = 30
- Value: 50 + 140 + 60*(5/10) = 190 + 30 = 220

EXPERIMENT – 4

Aim: Implement following programs using Dynamic programming(atleast 2)

General procedure for these type of technique:-

1. Create a table (of the right dimensions to describe our problem.
2. Fill the table, re-using solutions to previous sub-problems.

Problem description :-

(i) Matrix chain multiplication (or Matrix Chain Ordering Problem, MCOP) is an optimization problem. Given a sequence of matrices, the goal is to find the most efficient way to multiply these matrices.

Procedure:-

Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

Example:-
Input: p[] = {40, 20, 30, 10, 30}
Output: 26000
There are 4 matrices of dimensions 40x20, 20x30, 30x10 and 10x30.
Let the input 4 matrices be A, B, C and D. The minimum number of multiplications are obtained by putting parenthesis in following way
(A(BC))D → 20*30*10 + 40*20*10 + 40*10*30

(ii) String editing

Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert 'str1′ into 'str2′.

a) Insert b) Remove c) Replace

All of the above operations are of equal cost.

Example:

Input: str1 = "geek", str2 = "gesek"

Output: 1

We can convert str1 into str2 by inserting a 's'.

iii) single source shortest path problem. -Given a graph and a source vertex *src* in graph, find shortest paths from *src* to all vertices in the given graph.

Example :

```
Graph =                {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                        {4, 0, 8, 0, 0, 0, 0, 11, 0},
                        {0, 8, 0, 7, 0, 4, 0, 0, 2},
                        {0, 0, 7, 0, 9, 14, 0, 0, 0},
                        {0, 0, 0, 9, 0, 10, 0, 0, 0},
                        {0, 0, 4, 0, 10, 0, 2, 0, 0},
                        {0, 0, 0, 14, 0, 2, 0, 1, 6},
                        {8, 11, 0, 0, 0, 0, 1, 0, 7},
                        {0, 0, 2, 0, 0, 0, 6, 7, 0}
                       };
```

Output:-

```
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14
```

EXPERIMENT – 5

Aim: Implement following programs for Exploration of graphs :

i) Depth first search
Procedure :-
- starts at the root (selecting some arbitrary node as the root in the case of a graph)
- explore as far as possible along each branch before backtracking.

ii) Breadth first search-
Procedure:-
- Starts at the tree root (or some arbitrary node of a graph)
- Explore the neighbor nodes first, before moving to the next level neighbors.

OR

Aim : Implement application of Depth first search :

i) Finding articulation point of given graph.

Problem Articulation Points or Cut Vertices in a Graph :- A vertex in an undirected connected graph is anarticulation point (or cut vertex) iff removing it (and edges through it) disconnects the graph. It can be thought of as a single point of failure.

EXPERIMENT – 6

Aim: Implement following program using Backtracking technique :

1) n-queens

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

Example : solution for 4 Queen problem.

The output is a binary matrix which has 1s for the blocks where queens are placed.

```
{ 0,  1,  0,  0}
{ 0,  0,  0,  1}
{ 1,  0,  0,  0}
{ 0,  0,  1,  0}
```

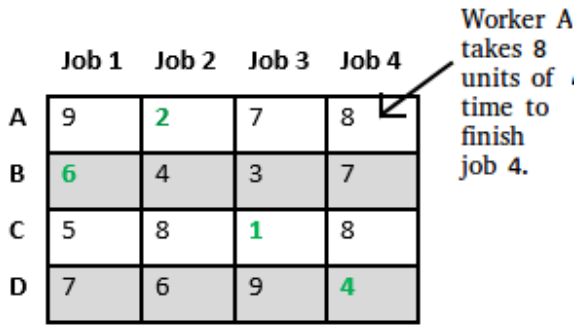
EXPERIMENT – 7

Aim: Implement following program(s) using Branch and bound technique: (alteast one)

(i) Assignment problem

Given n tasks and n agents. Each agent has a cost to complete each task.
• Assign each agent a task to minimize cost

Example:-



An example job assignment problem. Green values show optimal job assignment that is A-Job4, B-Job1 C-Job3 and D-Job4

(ii) knapsack problem

The difference between this problem and the fractional one is that you can't take a fraction of an item. You either take the whole thing or none of it. So here, is the problem formally described:
Your goal is to maximize the value of a knapsack that can hold at most W units worth of goods from a list of items I_0, I_1, ... I_{n-1}. Each item has two attributes:
1) Value - let this be v_i for item I_i.
2) Weight - let this be w_i for item I_i.
Now, instead of being able to take a certain weight of an item, you can only either take the item or not take the item.
Example:-

Item	Weight	Value
I_0	3	10
I_1	8	4
I_2	9	9
I_3	8	11

The maximum weight the knapsack can hold is 20.
optimal solution- {I_0, I_2, I_3}

EXPERIMENT – 8

Aim: Implement following programs using NP complete problems

1) Traveling salesman problem ……...

Given a set of cities and distance between every pair of cities, the **problem** is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Example:-

Consider the six-city problem with cost matrix given as input as follows:

0 8 5 3 1 2
8 0 4 9 2 8
5 4 0 9 6 7
3 9 9 0 1 1
1 2 6 1 0 9
2 8 7 1 9 0

Summary Results(output)
Cost of optimum tour: 15
Optimum tour: 1 3 2 5 4 6 1
Total of nodes generated: 31
Total number of nodes pruned: 13

EXPERIMENT – 9

Aim: Implement following programs using probabilistic design technique using Las vegas method

1) eight queens problem

One possible Algorithm

```
repeat
  set valid to true
  shuffle the current permutation vector
  set the two diagonal vectors to all false
  mark the row 0 queen in the two diagonal vectors
  for row = 1 to n-1
    set test = row+1
    while this queen is on an in-use diagonal
      if test = n
        set valid to false
        break out of the while loop
      else
        swap positions row and test
        increment test
      end if/else
    end while
    if not valid
      break out of the for loop
    mark this row's queen in the diagonal vectors
  end for
loop until valid
```

note :- The board is represented by a permutation vector giving the positions of the *n* queens in the *n* rows. This means that there are necessarily no attacks either horizontally or vertically; one only needs to check for the two diagonal attacks. One vector flags the diagonals of constant (row–col); the other flags the diagonals of constant (row+col).

EXPERIMENT – 10

Aim: Implement following programs using Understand Heuristic and approximate technique
1) determine whether a graph can be painted with just two colors……..

Example :- input

```
Graph g1(5);
    g1.addEdge(0, 1);
    g1.addEdge(0, 2);
    g1.addEdge(1, 2);
    g1.addEdge(1, 3);
    g1.addEdge(2, 3);
    g1.addEdge(3, 4);
```

```
Output:- Coloring of graph 1
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 1
```

EXPERIMENT – 11

Aim: Write a program that solves the Chinese postman problem .

The Chinese postman problem is a mathematical problem of graph theory. It is also known as route inspection problem. Suppose there is a mailman who needs to deliver mail to a certain neighborhood. The mailman is unwilling to walk far, so he wants to find the shortest route through the neighborhood, that meets the following criteria:

- It is a closed circuit (it ends at the same point it starts).
- He needs to go through every street at least once.

If the graph travelled has an Eulerian path, this circuit is the ideal solution.

EXPERIMENT – 12

Aim: Write a program that solve All pair shortest path problem with negative edges .

Problem :- find **shortest paths** in a weighted graph with positive or **negative** edge **weights** (but with no **negative** cycles).

References

Reference books:

- Fundamentals of Computer Algorithms by Horowitz, Sahni, Galgotia Pub. 2001 ed.
- Fundamentals of Algorithms by Brassard & Brately, PHI.
- Introduction to Algorithms by Coreman, Tata McGraw Hill.
- Design & Analysis of Computer Algorithms, Aho, Ullman, Addision Wesley
- The art of Computer Programming Vol.I & III, Kunth, Addision Wesley.