Laboratory Manual

For

Design and Analysis of Algorithm

(IT 509)

B.Tech (IT)

SEM V

July 2015

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

Table of Contents

Sample Experiment

1 AIM: Implement Merge Sort .

2 TOOLS/APPARATUS: Turbo C or gcc / gprof 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 the Solution

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 the Solution

By giving command gprof a.out > abc
vi abc
output will be:

Flat profile:

Each sample counts as 0.01
seconds. no time accumulated
% cumulative self
self total
time seconds seconds calls Ts/call Ts/call name 0.00 0.00
0.00 1 0.00 0.00 frame_dummy
% the percentage of the total running time of the
time program used by this function.
cumulative a running sum of the number of seconds accounted
seconds for by this function and those listed above it.
self the number of seconds accounted for by this seconds function
alone.This is the major sort for this listing.
calls the number of times this function was invoked, if
this function is profiled, else blank.
self the average number of milliseconds spent in this
ms/call function per call, if this function is profiled ,else blank.
total the average number of milliseconds spent in this
ms/call function and its descendents per call, if this
function is profiled, else blank.
name the name of the function. This is the minor sort for this listing.
The index shows the location of the function in the gprof listing. If the
index is
in parenthesis it shows where it would appear in the gprof listing if it were
to be printed.

Call graph (explanation follows)
granularity: each sample hit covers 4 byte(s) no time propagated
index % time self children called name
0.00 0.00 1/1 __do_global_dtors_aux [11]
[1] 0.0 0.00 0.00 1 frame_dummy [1]

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.
Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:
index A unique number given to each element of the table.

Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function in the table.

% time This is the percentage of the `total' time that was spent
in this function and its children. Note that due to
different viewpoints, functions excluded by options, etc,
these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called.
If the function called itself recursively, the number
only includes non-recursive calls, and is followed by
a `+' and the number of recursive calls.

name The name of the current function. The index number is
printed after it. If the function is a member of a
cycle, the cycle number is printed between the
function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function
into this parent.

children This is the amount of time that was propagated from the function's
children into this parent.

called This is the number of times this parent called the
function `/' the total number of times the function
was called. Recursive calls to the function are not
included in the number after the `/'.

name This is the name of the parent. The parent's index
number is printed after it. If the parent is a
member of a cycle, the cycle number is printed between
the name and the index number.

If the parents of the function cannot be determined, the word
`<spontaneous>' is printed in the `name' field, and all the other
fields are blank. For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly
from the child into the function.

children This is the amount of time that was propagated from the
child's children to the function.

called This is the number of times the function called
this child `/' the total number of times the child
was called. Recursive calls by the child are not
listed in the number after the `/'.

name This is the name of the child. The child's index
number is printed after it. If the child is a
member of a cycle, the cycle number is printed
between the name and the index number.

If there are any cycles (circles) in the call graph, there is an

entry for the cycle-as-a-whole. This entry shows who called the cycle
(as parents) and the members of the cycle (as children.)
The `+' recursive calls entry shows the number of function calls that
were internal to the cycle, and the calls entry for each member shows,
for that member, how many times it was called from other members of
the cycle.Index by function name
[1] frame_dummy

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)
Time Complexity Of Merge Sort in average case is: O(n logn)

Required Software/ Software Tool

-Linux Operating System and/ or Windows Operating System -C/C++

COMMON PROCEDURE:

Step 1**:** For the given problem statement design Flowchart/Algorithm/Logic

Step 2: Define variables and functions which will show the flow of program Step 3: Write C code in the file with .c extension

Step 4: Compile code using gcc compiler for Linux ,which will create a.out executable file.

In case of using Windows as Operating System, compile and run .c file.

Step 5: Test the program using sample input and write down output.

EXPERIMENT - 1

Aim: Introduction to gnu profiler tool.

Procedure:

Give introduction to gnu profiler tool.
Write a program in .c file.
Compile a program using gcc filename-pg option.
Execute output file using ./a.out option.
Execute gprof<filename to see profiling.

EXPERIMENT - 2

Aim: a) Write a program to implement Tower of Hanoii problem. Procedure:

Move n number of disks from tower a to tower b.
Use tower c as intermediate tower.

b) Write a program that implements Fibonacii Series.

EXPERIMENT – 3

Aim: a) Write a program that implements Insertion Sort
Procedure:

Function insert takes as a parameter an array containing a sequence of length n that is to be sorted.
The input array A contains the sorted output sequence when insert function is finished.

b) Write a program that implements Selection
Sort. Procedure:

Selection Sort selects the kth smallest element in a[1…n] and places it in the kth position of a[].
The remaining elements are rearranged such that a[m]<=a[k] for 1<=m<k and a[m]>=a[k] for k<m<=n.

EXPERIMENT – 4

Aim: a) Write a program that implements Heap Sort.

Procedure:

Heap sort starts by using Build_Max_Heap to build a max heap on the input array A[1…n], where n =length[A].

Maximum element of the array is stored at the root A[1], it can be put its correct final position by exchanging it with A[n].

b) Write a program that implements Quick Sort.

Procedure:

Quick sort sorts the elements a[p]….a[q] which resides in the global array a[1…n] into ascending order.

A[n+1] is considered to be difined and it must be >= all the elements in a[1…n] 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.

c) Write a program that implements Merge Sort.

Procedure:

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 is 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.

EXPERIMENT – 5

Aim: a) Write a program that implements Binary Search.

Procedure:

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.

b) Write a program that implements Prim's Algorithm. In

Prim's algorithm, E is the set of edges in G.

Cost [1:n,1:n] is the cost adjacency matrix of n vertex graph such that cost[I,j] is either a positive real number or infinity if no edge (I,j) exists.

A minimum spanning tree is computed and stored as a set of edges in the array t[1:n-1,1:2].

T[I,1],t[I,2] is an edge in the minimum cost spanning tree.

The final cost is returned as output.

EXPERIMENT – 6

Aim: a) Write a program that implements Kruskal's Algorithm.

Procedure:

In Kruskals algorithm, E is the set of edges in graph G. G has n vertices.

Cost[u,v] is the cost of edges (u,v).

T is the set of edges in the minimum cost spanning tree.

The final cost is returned as output.

b) Write a program that implements String editing Procedure:

In String editing, two strings X-x1,x2…xn and Y=y1,y2,…yn where xi,i<=i<-n and yj, 1<=j<=m are members of a finite set of symbols.

We want to transform X into Y using a minimum sequence of operations. Permissible operations are insert,delete and change.

EXPERIMENT – 7

Aim: a) Write a program that implements Make a change using greedy. 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.

b) Write a program that implements Knapsack using greedy. Procedure:

P[1:n] and w[1:n] contain the profits and weights respectively f the n objects .

N objects are ordered such that p[i]/w[i]>=p[i+1]/w[i+1].

M is the knapsack size and x[1:n] is the solution vector.

EXPERIMENT – 8

Aim: a) Write a program that implements Dijkstra's Algorithm.

Procedure:

Dijkstra's algorithm finds length of the shortest path from source node to each of other nodes of the graph.

C is a candidate set and S is the solution set.

b) Write a program that implements Longest Common Subsequence. Procedure:

Given two strings two strings X=x1,x2…xn and Y=y1,y2,…yn where xi,i<=i<-n and yj, 1<=j<=m are members of a finite set of symbols.

We have to find longest common sequence from the given two strings,

EXPERIMENT – 9

Aim: Write a program that implements N-Queen Problem.

Procedure:

In N queen problem, we have to arrange n number of queens on n*n chessboard such that no two queens can be on same row,same column or same diagonal.

EXPERIMENT – 10

Aim: Write a program that implements Knapsack using backtracking. Procedure:

P[1:n] and w[1:n] contain the profits and weights respectively f the n objects .

N objects are ordered such that p[i]/w[i]>=p[i+1]/w[i+1].

M is the knapsack size and x[1:n] is the solution vector.

EXPERIMENT – 11

Aim**:** Write a program that implements Make a change using dynamic. 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.

EXPERIMENT – 12

Aim: Write a program that implements All pair shortest path
problem. Procedure:

All pair shortest path algorithm finds length of the shortest path between each pair of nodes

C is a candidate set and S is the solution set.

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.