DAA Lab-8

Name: Kshitij Kumar Sharma Roll No.: 1905514 Date: 10/09/2021

**Q-8.1)** Write a program to implement the activity selection problem stated as follows.

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. For n = {10, 50, 100}, generate start time s[i] randomly in the range [1-50], [1-100], and [1-150], respectively. Then, the finish time is f[i] = s[i] + x[i], where x[i] is a random number generated in the range [1,s[i]].

Report maximum number of compatible activities and run time.

**Program:**

/\*

Written by: Kshitij Kumar Sharma Roll No.: 1905514

Idea of the solution:

I have taken a activity structure which keeps id, start and end of the activity. And using quick sort algorithm I have sorted the activity structure array on the basis of the end time of the activity. And then by recursive and iterative approaches I have found the maximum no of activities that can be taken and analysed the time for both the algorithms and then displayed it.

\*/

#include<bits/stdc++.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

using namespace std;

int n=10,a[10],f=0; //size and global flag.

struct activity //activity structure

{

    int id,start,end;

}s[10],t;

int partition(int p,int r) //for quick sort

    {

        int j,i=p-1;

        int pivot=s[r].end;

        for(j=p;j<=r-1;j++)

            {

                if(s[j].end<=pivot)

                    {

                        i++;

                        t=s[i];

                        s[i]=s[j];

                        s[j]=t;

                    }

            }

            t=s[i+1];

            s[i+1]=s[r];

            s[r]=t;

            return i+1;

    }

int random\_partition(int p,int r) //for quick sort

    {

        srand(time(0));

        int i=(rand()%(r-p+1))+p;

        t=s[r];

        s[r]=s[i];

        s[i]=t;

        return partition(p,r);

    }

void random\_quick\_sort(int p,int r) //for quick sort

    {

        int q;

        if(p<r)

            {

                q=random\_partition(p,r);

                random\_quick\_sort(p,q-1);

                random\_quick\_sort(q+1,r);

            }

    }

int pos(int a) //this returns the position of activity id

    {

        int i;

        for(i=0;i<n;i++)

            {

            if(a==s[i].id)

                return i;

            }

    }

void print() //to display activities

    {

        int i;

        cout<<"<start, end> \n";

        for(i=0;i<n;i++)

            {

                cout<<"<"<<s[i].start<<", "<<s[i].end<<">"<<" ";

            }

        cout<<endl;

    }

void print(int c[],int n)

    {

        int i;

        cout<<"<start, end> \n";

        for(i=0;i<n;i++)

            {

                cout<<"<"<<s[pos(c[i])].start<<", "<<s[pos(c[i])].end<<">"<<" ";

            }

        cout<<endl;

    }

void activity\_selector\_recursive(int k) //activity selection recursive

    {

        int m=k+1; //next activity to be checked

        if(k==0) //keeping first activity

            {

                a[f]=s[k].id; //keeping selected activity in a[]

                f++;

            }

        while(m<n && s[m].start<s[k].end) //skipping un-selectable activity

            m++;

        if(m<n) //condition for selection

            {

                a[f]=s[m].id; //keeping selected activity in a[]

                f++;

                activity\_selector\_recursive(m); //recursive call

            }

        else

            return;

    }

void activity\_selector\_iterative() //activity selector iterative

    {

        int i,b[n],z=0;

        b[z]=s[0].id; //array for keeping selected id’s

        z++;

        int k=0;

        for(i=1;i<n;i++) //iterating throughout all activities

            {

                if(s[i].start >= s[k].end)   //checking selection condition

                    {

                        b[z]=s[i].id; //keeping selected activity in b[]

                        k=i;

                        z++;

                    }

            }

        print(b,z);

    }

int main()

    {

        int i,j;

        clock\_t start,stop; //variables for timing analysis

        double duration;

        srand(time(0));

        for(i=0;i<n;i++) //random activity generator

            {

            s[i].id=i+1;

            s[i].start=rand()%20;

            s[i].end=s[i].start+1+rand()%20;

            }

        print();

        random\_quick\_sort(0,n-1); //sorting the activities

        print();

        cout<<"Recursive \n";

        start=clock();

        activity\_selector\_recursive(0); //call to recursive version

        print(a,f);

        stop=clock();

        duration=((double)(stop-start)/CLOCKS\_PER\_SEC);

        printf("time taken=%f sec\n",duration);

        cout<<"Iterative \n";

        start=clock();

        activity\_selector\_iterative(); //call to iterative version

        stop=clock();

        duration=((double)(stop-start)/CLOCKS\_PER\_SEC);

        printf("time taken=%f sec\n",duration);

    }

**Output:**



