**LAB PROGRAMS**

**1) GCD**

**#include <stdio.h>**

**int gcd(int n1, int n2);**

**int main() {**

**int n1, n2;**

**printf("Enter two positive integers: ");**

**scanf("%d %d", &n1, &n2);**

**printf("G.C.D of %d and %d is %d.", n1, n2, gcd(n1, n2));**

**return 0;**

**}**

**int gcd(int n1, int n2) {**

**if (n2 != 0)**

**return gcd(n2, n1 % n2);**

**else**

**return n1;**

**}**

**/\*int gcd(int n1, int n2)**

**{**

**while (n1 != n2) {**

**if (n1 > n2)**

**n1 = n1 - n2;**

**else**

**n2 = n2 - n1;**

**}**

**return n1;**

**}\*/**

**2) Linear search**

**#include <stdio.h>**

**int RecursiveLS(int arr[], int value, int index, int n)**

**{**

**int pos = 0;**

**if(index >= n)**

**{**

**return 0;**

**}**

**else if (arr[index] == value)**

**{**

**pos = index + 1;**

**return pos;**

**}**

**else**

**{**

**return RecursiveLS(arr, value, index+1, n);**

**}**

**return pos;**

**}**

**int main()**

**{**

**int n, value, pos, m = 0, arr[100];**

**printf("Enter the total elements in the array ");**

**scanf("%d", &n);**

**printf("Enter the array elements\n");**

**for (int i = 0; i < n; i++)**

**{**

**scanf("%d", &arr[i]);**

**}**

**printf("Enter the element to search ");**

**scanf("%d", &value);**

**pos = RecursiveLS(arr, value, 0, n);**

**if (pos != 0)**

**{**

**printf("Element found at pos %d ", pos);**

**}**

**else**

**{**

**printf("Element not found");**

**}**

**return 0;**

**}**

**3) Binary search**

**#include <stdio.h>**

**void binary\_search(int [], int, int, int);**

**int main()**

**{**

**int key, size, i;**

**int list[25];**

**printf("Enter size of a list: ");**

**scanf("%d", &size);**

**printf("Enter elements\n");**

**for(i = 0; i < size; i++)**

**{**

**scanf("%d",&list[i]);**

**}**

**printf("Enter key to search\n");**

**scanf("%d", &key);**

**binary\_search(list, 0, size, key);**

**}**

**void binary\_search(int list[], int lo, int hi, int key)**

**{**

**int mid;**

**if (lo > hi)**

**{**

**printf("Key not found\n");**

**return;**

**}**

**mid = (lo + hi) / 2;**

**if (list[mid] == key)**

**{**

**printf("Key found\n");**

**}**

**else if (list[mid] > key)**

**{**

**binary\_search(list, lo, mid - 1, key);**

**}**

**else if (list[mid] < key)**

**{**

**binary\_search(list, mid + 1, hi, key);**

**}**

**}**

**4) Fibonacci**

**#include<stdio.h>**

**int Fibonacci(int);**

**int main()**

**{**

**int n, i = 0, c;**

**printf("enter the number of terms");**

**scanf("%d",&n);**

**printf("Fibonacci series\n");**

**for ( c = 1 ; c <= n ; c++ )**

**{**

**printf("%d\n", Fibonacci(i));**

**i++;**

**}**

**return 0;**

**}**

**int Fibonacci(int n)**

**{**

**if ( n == 0 )**

**return 0;**

**else if ( n == 1 )**

**return 1;**

**else**

**return ( Fibonacci(n-1) + Fibonacci(n-2) );**

**}**

**5) Insertion sort**

**#include<stdio.h>**

**int main(){**

**int i, j, n, temp, number[25];**

**printf("Enter the count");**

**scanf("%d",&n);**

**printf("Enter %d elements: ", n);**

**for(i=0;i<n;i++)**

**scanf("%d",&number[i]);**

**for(i=1;i<n;i++){**

**temp=number[i];**

**j=i-1;**

**while((temp<number[j])&&(j>=0)){**

**number[j+1]=number[j];**

**j=j-1;**

**}**

**number[j+1]=temp;**

**}**

**printf("Order of Sorted elements: ");**

**for(i=0;i<n;i++)**

**printf(" %d",number[i]);**

**return 0;**

**}**

**6) Selection sort**

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h>**

**void selection(int \*a,int n)**

**{**

**delay(1000);**

**int i,j,temp;**

**for(i=0;i<n-1;i++)**

**{**

**min=i;**

**for(j=i+1;j<n;j++)**

**{**

**if(a[j]<min)**

**{**

**min=j;**

**}**

**}**

**temp=a[i];a[i]=a[j];a[j]=temp;**

**}**

**}**

**void main()**

**{**

**FILE \*fp;**

**clock\_t c1,c2;**

**int i,datasize=1;**

**long int n=1000;**

**int \*a;**

**fp=fopen("values.dat","w+");**

**if(fp==NULL)**

**{**

**printf("cannot open file");**

**exit(1);**

**}**

**while(datasize<=10)**

**{**

**a=(int \*)calloc(n,sizeof(int));**

**if(a==NULL)**

**{**

**printf("INSUFFICIENT MEMORY\N");**

**exit(1);**

**}**

**randomize();**

**for(i=0;i<=n-1;i++) a[i]=rand() % MAXINT;**

**c1=clock();**

**selection(a,n);**

**c2=clock();**

**free(a);**

**if((c2 -c1) != 0)**

**{**

**fprintf(fp,"%d\t%f\n",(c2-c1)/CLK\_TCK);**

**datasize++;**

**}**

**n=n+1000;**

**}**

**fclose(fp);**

**}**

**7) Tower of Hanoi**

**#include <stdio.h>**

**void tower(int, char, char, char);**

**int main()**

**{**

**int num;**

**printf("Enter the number of disks : ");**

**scanf("%d", &num);**

**printf("The smoves are\n");**

**tower(num, 'A', 'C', 'B');**

**return 0;**

**}**

**void tower(int num, char peg1, char peg3, char peg2)**

**{**

**if (num == 1)**

**{**

**printf("\n Move disk 1 from peg %c to peg %c", peg1, peg3);**

**return;**

**}**

**tower(num - 1, peg1, peg2, peg3);**

**printf("\n Move disk %d from peg %c to peg %c", num, peg1, peg3);**

**tower(num - 1, peg2, peg3, peg1);**

**}**

**8) ADS**

**#include<stdio.h>**

**int a[1][10];**

**void dfs(int n, int cost[10][10], int u, int s[])**

**{**

**int v;**

**s[u]=1;**

**printf("order of dfs is - %d \n",u+1);**

**for(v=0;v<n;v++)**

**{**

**if((cost[u][v]==1) && (s[v]==0))**

**dfs(n,cost,v,s);**

**}**

**}**

**void main()**

**{**

**int n,i,j,cost[10][10],s[10],con,flag;**

**printf("Enter the number of nodes\n");**

**scanf("%d",&n);**

**printf("Enter the adjacency matrix\n");**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<n;j++)**

**scanf("%d", &cost[i][j]);**

**}**

**con=0;**

**for(j=0;j<n;j++)**

**{**

**for(i=0;i<n;i++)**

**s[i]=0;**

**dfs(n,cost,j,s);**

**flag=0;**

**for(i=0;i<n;i++)**

**{**

**if(s[i]==0)**

**flag=1;**

**}**

**if(flag==0)**

**con=1;**

**}**

**if(con==1)**

**printf("Graph is connnected\n");**

**else**

**printf("Graph is not connected\n");**

**getchar();**

**}**

**9) BFS**

**#include<stdio.h>**

**void insertq(int q[],int node, int \*f, int \*r)**

**{**

**if((\*f==-1) && (\*r==-1))**

**{**

**(\*f)++, (\*r)++, q[\*f]=node;**

**}**

**else{**

**(\*r)++, q[\*r]=node;**

**}**

**}**

**int deleteq(int q[],int \*f,int \*r)**

**{**

**int temp;**

**temp=q[\*f];**

**if(\*f == \*r) \*f=\*r=-1;**

**else (\*f)++;**

**return temp;**

**}**

**void bfs(int n, int adj[][10],int src, int visited[])**

**{**

**int q[20], f=-1,r=-1,v,i;**

**insertq(q,src,&f,&r);**

**while((f <=r ) && (f != -1))**

**{**

**v=deleteq(q,&f,&r);**

**if(visited[v]!=1)**

**{**

**visited[v]=1;**

**printf("%d",v);**

**}**

**for(i=1;i<=n;i++)**

**if((adj[v][i]==1) && (visited[i] !=1))**

**insertq(q,i,&f,&r);**

**}**

**}**

**void main()**

**{**

**int n,i,j,adj[10][10],src,visited[10];**

**printf("enter number of vertices\n");**

**scanf("%d",&n);**

**printf("Enter adjacency matrix\n");**

**for(i=1;i<=n;i++)**

**{**

**visited[i]=0;**

**for(j=1;j<=n;j++)**

**scanf("%d",&adj[i][j]);**

**}**

**printf("enter starting vertex\n");**

**scanf("%d",&src);**

**printf("The nodes reachable from src are\n");**

**bfs(n,adj,src,visited);**

**getchar();**

**}**