**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT**

**on**

**ANALYSIS AND DESIGN OF ALGORITHMS**

***Submitted by***

**VENKATA SATWIK POTULA (1BM20CS183)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “Analysis And Design Of Algorithms” carried out by **VENKATA SATWIK POTULA (1BM20CS183),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a**Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-Incharge               **Dr. Jyothi S Nayak**

Designation Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | **Tower of Hanoi and GCD** |  |
| **2** | **Binary Search and Linear Search** |  |
| **3** | **Selection Sort** |  |
| **4** | **Insertion Sort** |  |
| **5** | **DFS, BFS** |  |
| **6** | **Merge Sort** |  |
| **7** | **Quick Sort** |  |
| **8** | **Topological Sort** |  |
| **9** | **Johnson Trotter** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Course Outcome**

|  |  |  |
| --- | --- | --- |
| CO1 | Ability to **analyse** the time complexity of Recursive and Non-recursive algorithms using asymptotic notations | |
| CO2 | | Ability to **design** efficient algorithms using various design techniques |
| CO3 | | Ability to **apply** the knowledge of complexity classes P, NP and NP-Complete and prove certain problems are NP-Complete |
| CO4 | | Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find the time efficiency |

## 1. Tower of hanoi and gcd

#include <stdio.h>

int hcf(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, hcf(n1,n2));

return 0;

}

int hcf(int n1, int n2)

{

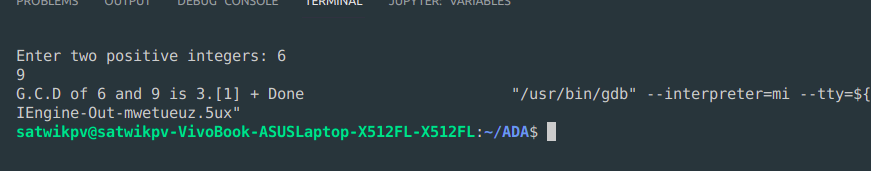
if (n2 != 0)

return hcf(n2, n1%n2);

else

return n1;

}



#include<stdio.h> // place this '<' & '>' instead of '(' & ')' before stdio.h

#include<math.h>

void hanoi(int x, char from, char to, char aux)

{

if(x==1)

printf("Move Disk From %c to %c\n",from,to);

else

{

hanoi(x-1,from,aux,to);

printf("Move Disk From %c to %c\n",from,to);

hanoi(x-1,aux,to,from);

}

}

void main( )

{

int disk;

int moves;

printf("Enter the number of disks you want to play with:");

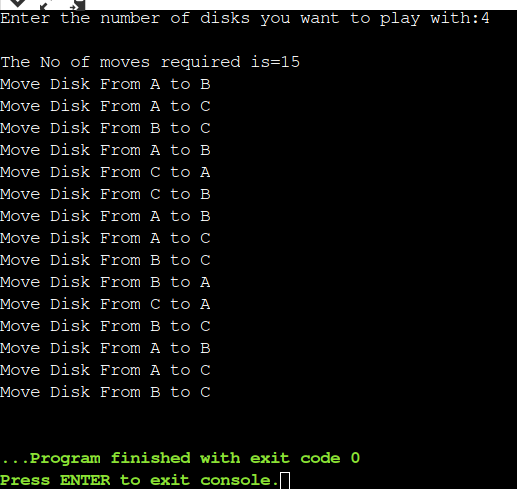
scanf("%d",&disk);

moves=pow(2,disk)-1;

printf("\nThe No of moves required is=%d \n",moves);

hanoi(disk,'A','C','B')

}



## 2. Binary Search and Linear Search

#include<stdio.h>

#include<time.h>

#include<stdlib.h> /\* To recognise exit function when compiling with gcc\*/

int bin\_srch(int [],int,int,int);

int lin\_srch(int [],int,int,int);

void bub\_sort(int[],int);

int n,a[10000];

int main()

{

int ch,key,search\_status,temp;

clock\_t end,start;

unsigned long int i, j;

while(1)

{

printf("\n1: Binary search\t 2: Linear search\t 3: Exit\n");

printf("\nEnter your choice:\t");

scanf("%d",&ch);

switch(ch)

{

case 1:

n=1000;

while(n<=5000)

{

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

{

//a[i]=random(1000);

a[i]=i; //Insering numbers in Ascending order

}

key=a[n-1]; //Last element of the aray

start=clock();

//bub\_sort(a,n); //Sorting numbers in Ascending order using Bubble sort

search\_status=bin\_srch(a,0,n-1,key);

if(search\_status==-1)

printf("\nKey Not Found");

else

printf("\n Key found at position %d",search\_status);

//Dummy loop to create delay

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 2:

n=1000;

while(n<=5000)

{

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

{

//a[i]=random(10000);

a[i]=i;

}

key=a[n-1]; //Last element of the aray

start=clock();

search\_status=lin\_srch(a,0,n-1,key);

if(search\_status==-1)

printf("\nKey Not Found");

else

printf("\n Key found at position %d",search\_status);

//Dummy loop to create delay

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

default:

exit(0);

}

getchar();

}

}

void bub\_sort(int a[],int n)

{

int i,j,temp;

for(i=0;i<=n-2;i++)

{

for(j=0;j<=n-2-i;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

int bin\_srch(int a[],int low,int high,int key)

{

int mid;

if(low>high)

{

return -1;

}

mid=(low+high)/2;

if(key==a[mid])

{

return mid;

}

if(key<a[mid])

{

return bin\_srch(a,low,mid-1,key);

}

else

{

return bin\_srch(a,mid+1,high,key);

}

}

int lin\_srch(int a[],int i,int high,int key)

{

if(i>high)

{

return -1;

}

if(key==a[i])

{

return i;

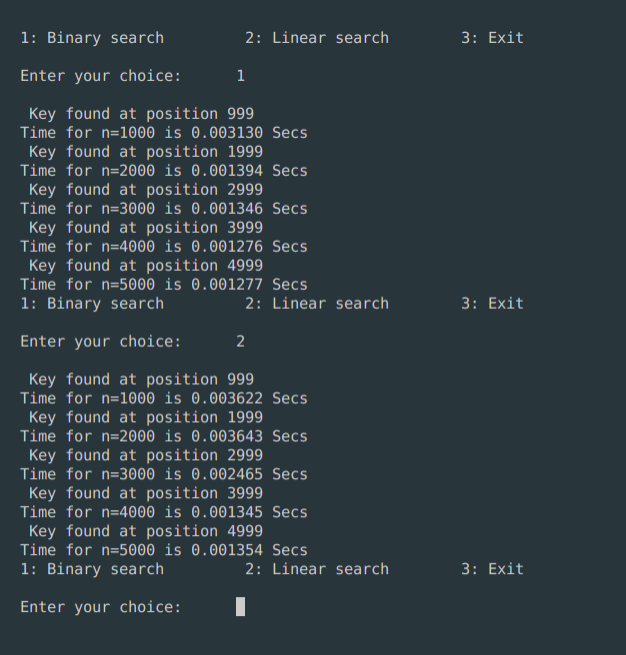
}

else

{ return lin\_srch(a,i+1,high,key);

}

}



## 3. Selection Sort

#include<stdio.h>

#include<time.h>

#include<stdlib.h> /\* To recognise exit function when compiling with gcc\*/

void selsort(int n,int a[]);

void main()

{

int a[15000],n,i,j,ch,temp;

clock\_t start,end;

while(1)

{

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

{

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

}

start=clock();

selsort(n,a);

end=clock();

printf("\nSorted array is: ");

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

printf("%d\t",a[i]);

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2:

n=500;

while(n<=14500) {

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

{

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

selsort(n,a);

//Dummy loop to create delay

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3: exit(0);

}

getchar();

}

}

void selsort(int n,int a[])

{

int i,j,t,small,pos;

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

{

pos=i;

small=a[i];

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

{

if(a[j]<small)

{

small=a[j];

pos=j;

}

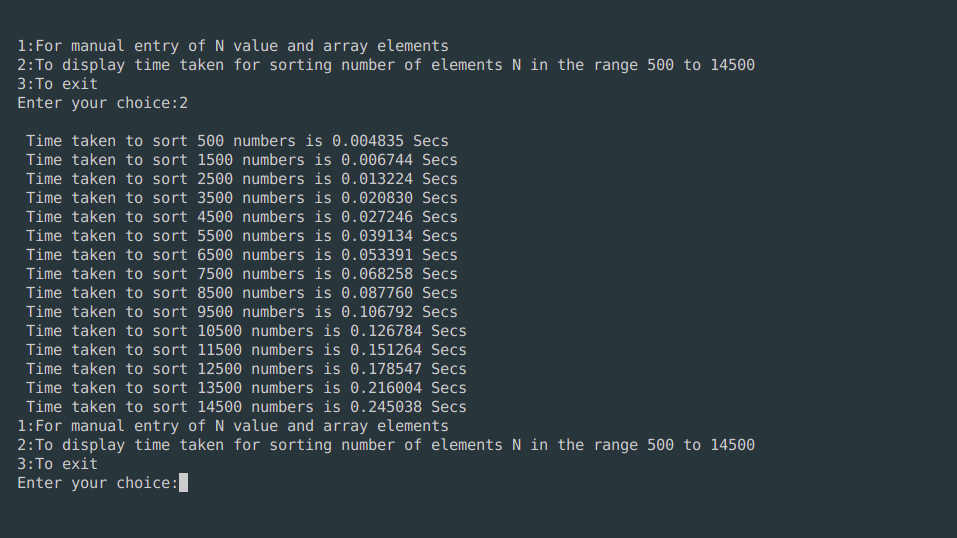
}

t=a[i];

a[i]=a[pos];

a[pos]=t;

} }



## 4. Insertion Sort

#include <stdio.h>

#include <time.h>

void insSort(int arr[], int n)

{

int i, key, j;

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

{

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1],

that are greater than key,

to one position ahead of

their current position \*/

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

void main()

{

int a[15000],n, i,j,ch, temp;

clock\_t start,end;

while(1)

{

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

{

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

}

start=clock();

insSort(a,n);

end=clock();

printf("\nSorted array is: ");

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

printf("%d\t",a[i]);

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2:

n=500;

while(n<=14500) {

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

{

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

insSort(a,n);

//Dummy loop to create delay

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

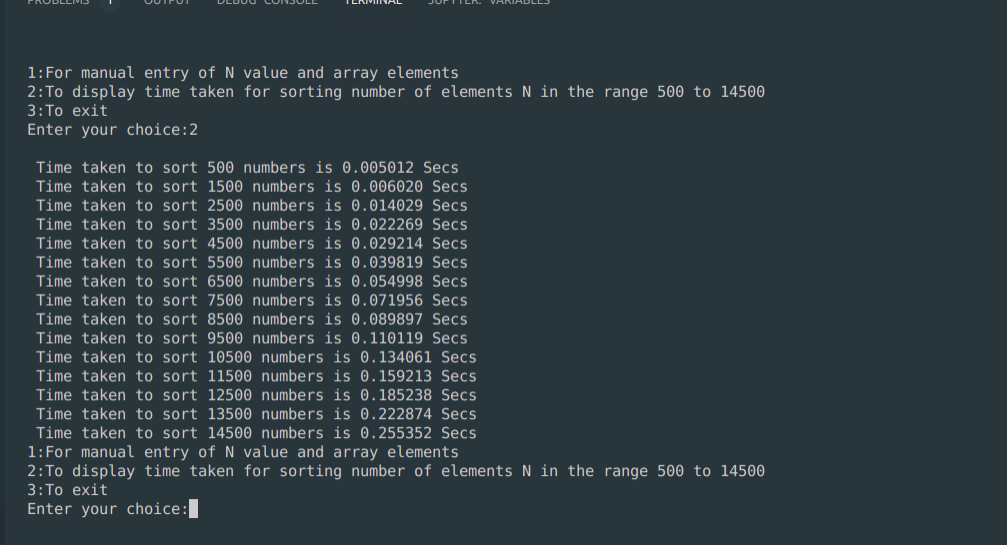
break;

case 3: exit(0);

}

getchar();

} }



## 

## 5. DFS And BFS

## #include<stdio.h>

int a[10][10],n,vis[10];

int dfs(int);

void main()

{

int i,j,src,ans;

// clrscr();

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

{

vis[j]=0;

}

printf("\nenter the no of nodes:\t");

scanf("%d",&n);

printf("\nenter the adjacency matrix:\n");

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

{

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

{

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

}

}

printf("\nenter the source node:\t");

scanf("%d",&src);

ans=dfs(src);

if(ans==1)

{

printf("\ngraph is connected\n");

}

else

{

printf("\ngragh is not connected\n");

}

// getch();

}

int dfs(int src)

{

int j;

vis[src]=1;

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

{

if(a[src][j]==1&&vis[j]!=1)

{

dfs(j);

}

}

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

{

if(vis[j]!=1)

{

return 0;

}

}

return 1;

}

#include<stdio.h>

int a[10][10],n;

void bfs(int);

void main()

{

int i,j,src;

// clrscr();

printf("\nenter the no of nodes:\t");

scanf("%d",&n);

printf("\nenter the adjacency matrix:\n");

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

{

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

{

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

}

}

printf("\nenter the source node:\t");

scanf("%d",&src);

bfs(src);

// getch();

}

void bfs(int src)

{

int q[10],f=0,r=-1,vis[10],i,j;

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

{

vis[j]=0;

}

vis[src]=1;

r=r+1;

q[r]=src;

while(f<=r)

{

i=q[f];

f=f+1;

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

{

if(a[i][j]==1&&vis[j]!=1)

{

vis[j]=1;

r=r+1;

q[r]=j;

}

}

}

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

{

if(vis[j]!=1)

{

printf("\nnode %d is not reachable\n",j);

}

else

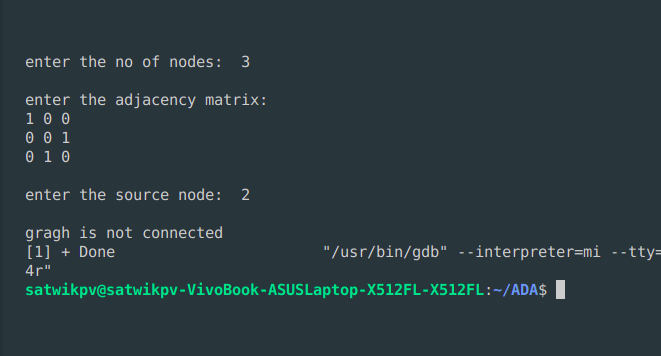
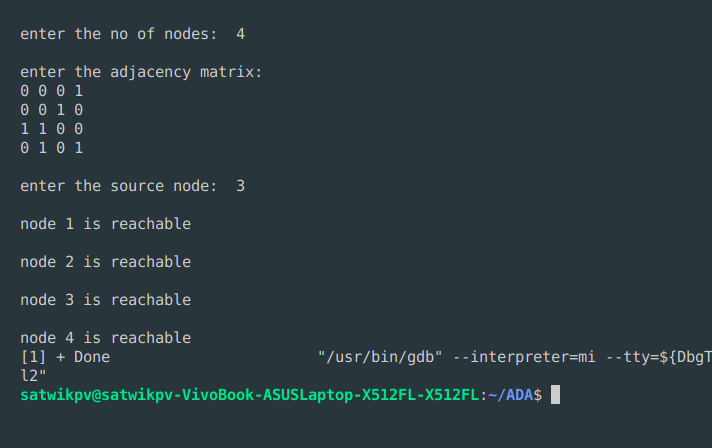
{

printf("\nnode %d is reachable\n",j);

}

}

}



## 6.Topological Sort

#include <stdio.h>

void source\_removal(int n, int a[10][10])

{

int i,j,k,u,v,top,s[10],t[10],indeg[10],sum;

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

{

sum=0;

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

{

sum+=a[j][i];

}

indeg[i]=sum;

}

top=-1;

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

{

if(indeg[i]==0)

{

s[++top]=i;

}

}

k=0;

while(top!=-1)

{

u=s[top--];

t[k++]=u;

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

{

if(a[u][v]==1)

{

indeg[v]=indeg[v]-1;

if(indeg[v]==0)

s[++top]=v;

}

}

}

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

{

printf("%d\n", t[i]);

}

}

void main()

{

int i,j,a[10][10],n;

printf("Enter 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", &a[i][j]);

}

}

source\_removal(n,a);

// getch(); }



## 7. Merge Sort

## 

#include <stdio.h>

#include<time.h>

#include<stdlib.h> /\* To recognise exit function when compiling with gcc\*/

void split(int[],int,int);

void combine(int[],int,int,int);

void main()

{

int a[15000],n, i,j,ch, temp;

clock\_t start,end;

while(1)

{

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

{

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

}

start=clock();

split(a,0,n-1);

end=clock();

printf("\nSorted array is: ");

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

printf("%d\t",a[i]);

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2:

n=500;

while(n<=14500) {

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

{

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

split(a,0,n-1);

//Dummy loop to create delay

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3: exit(0);

}

getchar();

}

}

void split(int a[],int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

split(a,low,mid);

split(a,mid+1,high);

combine(a,low,mid,high);

}

}

void combine(int a[],int low,int mid,int high)

{

int c[15000],i,j,k;

i=k=low;

j=mid+1;

while(i<=mid&&j<=high)

{

if(a[i]<a[j])

{

c[k]=a[i];

++k;

++i;

}

else

{

c[k]=a[j];

++k;

++j;

}

}

if(i>mid)

{

while(j<=high)

{

c[k]=a[j];

++k;

++j;

}

}

if(j>high)

{

while(i<=mid)

{

c[k]=a[i];

++k;

++i;

}

}

for(i=low;i<=high;i++)

{

a[i]=c[i];

}

}

## 

## 

## 8. Quick Sort

#include<stdio.h>

#include<time.h>

#include<stdlib.h> /\* To recognise exit function when compiling with gcc\*/

// Quick sort in C

// function to swap elements

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

// function to find the partition position

int partition(int array[], int low, int high) {

// select the rightmost element as pivot

int pivot = array[high];

// pointer for greater element

int i = (low - 1);

// traverse each element of the array

// compare them with the pivot

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

// if element smaller than pivot is found

// swap it with the greater element pointed by i

i++;

// swap element at i with element at j

swap(&array[i], &array[j]);

}

}

// swap the pivot element with the greater element at i

swap(&array[i + 1], &array[high]);

// return the partition point

return (i + 1);

}

void quick(int array[], int low, int high) {

if (low < high) {

// find the pivot element such that

// elements smaller than pivot are on left of pivot

// elements greater than pivot are on right of pivot

int pi = partition(array, low, high);

// recursive call on the left of pivot

quick(array, low, pi - 1);

// recursive call on the right of pivot

quick(array, pi + 1, high);

}

}

// function to print array elements

void printArray(int array[], int size) {

for (int i = 0; i < size; ++i) {

printf("%d ", array[i]);

}

printf("\n");

}

void main()

{

int a[15000],n, i,j,ch, temp;

clock\_t start,end;

while(1)

{

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

{

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

}

start=clock();

quick(a,0,n-1);

end=clock();

printf("\nSorted array is: ");

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

printf("%d\t",a[i]);

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2:

n=500;

while(n<=14500) {

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

{

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

quick(a,0,n-1);

//Dummy loop to create delay

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

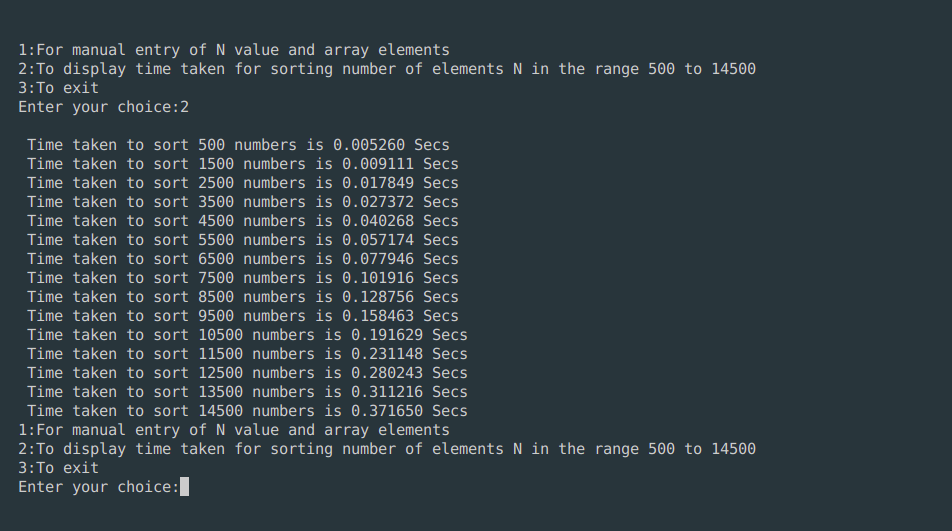
case 3: exit(0);

}

getchar();

}

}



## 9. Johnson Trotter

#include <stdio.h>

#include <stdlib.h>

int flag = 0;

int swap(int \*a,int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int search(int arr[],int num,int mobile)

{

int g;

for(g=0;g<num;g++) {

if(arr[g] == mobile)

return g+1;

else

flag++;

}

return -1;

}

int find\_Moblie(int arr[],int d[],int num)

{

int mobile = 0;

int mobile\_p = 0;

int i;

for(i=0;i<num;i++)

{

if((d[arr[i]-1] == 0) && i != 0)

{

if(arr[i]>arr[i-1] && arr[i]>mobile\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

flag++;

}

else if((d[arr[i]-1] == 1) & i != num-1)

{

if(arr[i]>arr[i+1] && arr[i]>mobile\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

flag++;

}

else

flag++;

}

if((mobile\_p == 0) && (mobile == 0))

return 0;

else

return mobile;

}

void permutations(int arr[],int d[],int num)

{

int i;

int mobile = find\_Moblie(arr,d,num);

int pos = search(arr,num,mobile);

if(d[arr[pos-1]-1]==0)

swap(&arr[pos-1],&arr[pos-2]);

else

swap(&arr[pos-1],&arr[pos]);

for(int i=0;i<num;i++)

{

if(arr[i] > mobile)

{

if(d[arr[i]-1]==0)

d[arr[i]-1] = 1;

else

d[arr[i]-1] = 0;

}

}

for(i=0;i<num;i++)

{

printf(" %d ",arr[i]);

} }

int factorial(int k)

{

int f = 1;

int i = 0;

for(i=1;i<k+1;i++)

f = f\*i;

return f;

}

int main()

{

int num = 0;

int i;

int j;

int z = 0;

printf("Johnson trotter algorithm to find all permutations of given numbers \n");

printf("Enter the number\n");

scanf("%d",&num);

int arr[num],d[num];

z = factorial(num);

printf("total permutations = %d",z);

printf("\nAll possible permutations are: \n");

for(i=0;i<num;i++)

{

d[i] = 0;

arr[i] = i+1;

printf(" %d ",arr[i]);

}

printf("\n");

for(j=1;j<z;j++) {

permutations(arr,d,num);

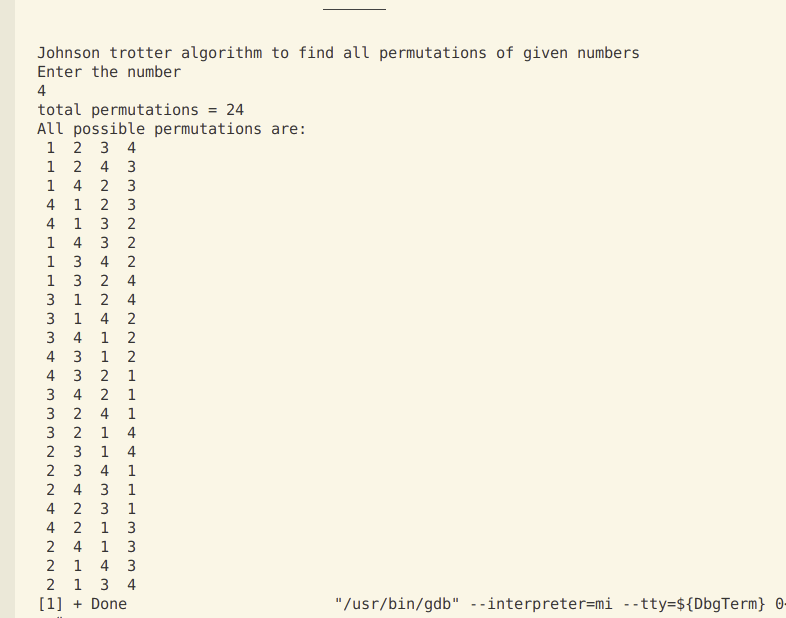
printf("\n");

}

return 0;

}

Output:



## 10. Floyd’s Algorithm

#include<stdio.h>

#include<conio.h>

int a[10][10],n;

void floyds();

int min(int,int);

void main()

{

int i,j;

clrscr();

printf("\nenter the no. of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

{

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

{

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

}

}

floyds();

getch();

}

void floyds()

{

int i,j,k;

for(k=1;k<=n;k++)

{

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

{

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

{

a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

}

}

}

printf("\nall pair shortest path matrix is:\n");

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

{

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

{

printf("%d\t",a[i][j]);

}

printf("\n\n");

}

}

int min(int x,int y)

{

if(x<y)

{

return x;

}

else

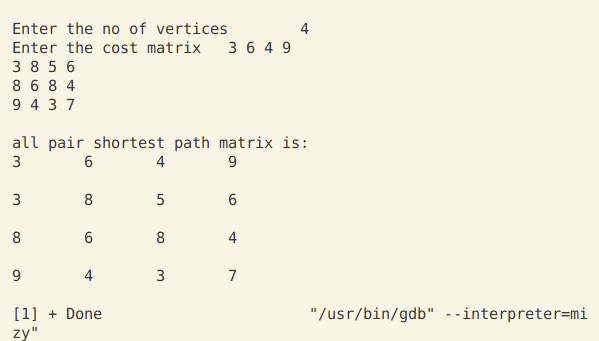
{

return y;

}

}

Output:



## 11. Warshall’s Algorithm

#include <stdio.h>

int a[10][10], n;

void warshall();

void main()

{

int i, j;

printf("Enter the no of vertices\t");

scanf("%d", &n);

printf("Enter the cost matrix\t");

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

{

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

{

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

}

}

warshall();

}

void warshall()

{

int i, j, k;

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

{

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

{

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

{

a[i][j] = (a[i][j] || (a[i][k] && a[k][j]));

}

}

}

printf("\ntransitive closure is:\n");

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

{

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

{

printf("%d\t", a[i][j]);

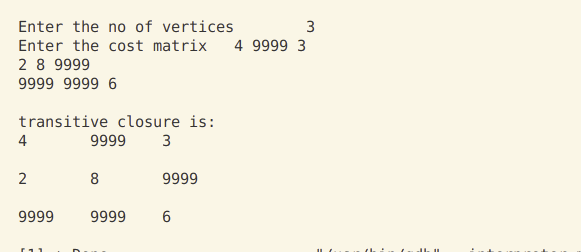
}

printf("\n\n");

}

}

Output:



## 12. Knapsack Problem

#include <stdio.h>

// #include<conio.h>

void knapsack();

int max(int, int);

int i, j, n, m, p[10], w[10], v[10][10];

void main()

{

// clrscr();

printf("\nenter the no. of items:\t");

scanf("%d", &n);

printf("\nenter the weight of the each item:\n");

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

{

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

}

printf("\nenter the profit of each item:\n");

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

{

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

}

printf("\nenter the knapsack's capacity:\t");

scanf("%d", &m);

knapsack();

// getch();

}

void knapsack()

{

int x[10];

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

{

for (j = 0; j <= m; j++)

{

if (i == 0 || j == 0)

{

v[i][j] = 0;

}

else if (j - w[i] < 0)

{

v[i][j] = v[i - 1][j];

}

else

{

v[i][j] = max(v[i - 1][j], v[i - 1][j - w[i]] + p[i]);

}

}

}

printf("\nthe output is:\n");

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

{

for (j = 0; j <= m; j++)

{

printf("%d\t", v[i][j]);

}

printf("\n\n");

}

printf("\nthe optimal solution is %d", v[n][m]);

printf("\nthe solution vector is:\n");

for (i = n; i >= 1; i--)

{

if (v[i][m] != v[i - 1][m])

{

x[i] = 1;

m = m - w[i];

}

else

{

x[i] = 0;

}

}

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

{

printf("%d\t", x[i]);

}

}

int max(int x, int y)

{

if (x > y)

{

return x;

}

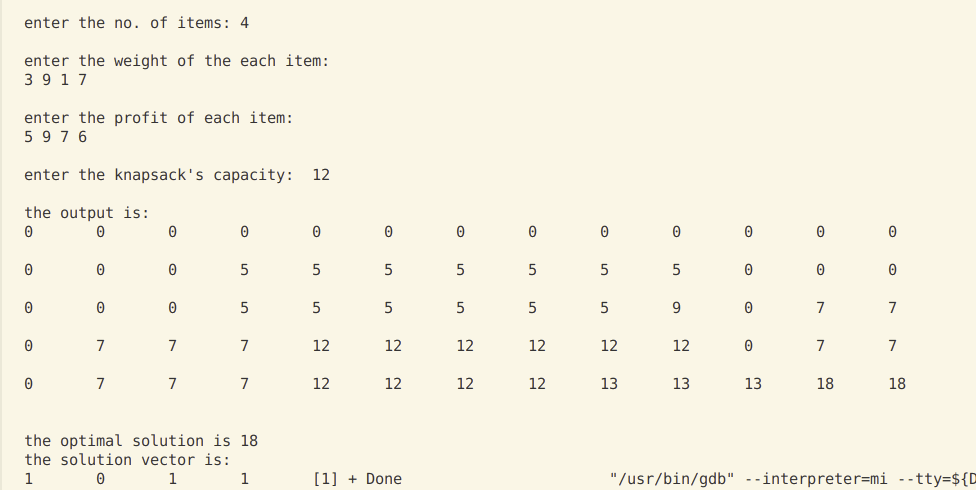
else

{

return y;

}

}

Output:

## 13. Prim’s Algorithm

#include <stdio.h>

int n, c[10][10];

void prims();

void main()

{

int i, j;

printf("Enter the number of vertices:\t");

scanf("%d", &n);

printf("Enter the cost matrix:\n");

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

{

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

{

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

}

}

prims();

}

void prims()

{

int i, j, u, v, min;

int ne = 0, mincost = 0;

int elec[10];

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

{

elec[i] = 0;

}

elec[1] = 1;

while (ne != n - 1)

{

min = 9999;

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

{

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

{

if (elec[i] == 1)

{

if (c[i][j] < min)

{

min = c[i][j];

u = i;

v = j;

}

}

}

}

if (elec[v] != 1)

{

printf("\n %d -----> %d=%d\n", u, v, min);

elec[v] = 1;

ne = ne + 1;

mincost = mincost + min;

}

c[u][v] = c[v][u] = 9999;

}

printf("\nmincost=%d", mincost);

}

Output:

## 

## 14. Kruskal’s Algorithm

#include <stdio.h>

// #include <conio.h>

void kruskals();

int c[10][10], n;

void main()

{

int i, j;

printf("Enter the no of vertices\n");

scanf("%d", &n);

printf("\nEnter the cost matrix:\n");

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

{

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

{

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

}

}

kruskals();

}

void kruskals()

{

int i, j, v, u, a, b, min;

int ne = 0, mincost = 0;

int parent[10];

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

{

parent[i] = 0;

}

while (ne != n - 1)

{

min = 9999;

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

{

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

{

if (c[i][j] < min)

{

min = c[i][j];

u = a = i;

v = b = j;

}

}

}

while (parent[u] != 0)

{

u = parent[u];

}

while (parent[v] != 0)

{

v = parent[v];

}

if (u != v)

{

printf("\n%d----->%d=%d\n ", a, b, min);

parent[v] = u;

ne = ne + 1;

mincost = mincost + min;

}

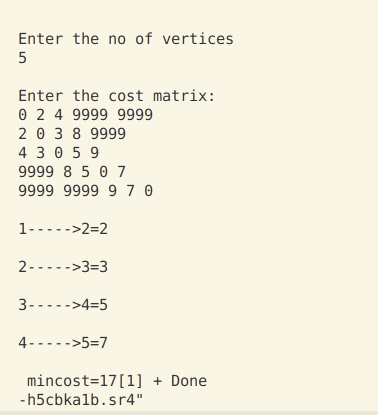
c[a][b] = c[b][a] = 9999;

}

printf("\n mincost=%d", mincost);

}

Output:



## 15. Djikstras Algorithm

#include <stdio.h>

int n, c[10][10], src;

void dijkstras();

void main()

{

int i, j;

printf("\n Enter the no of vertices:\t");

scanf("%d", &n);

printf("\nEnter the cost matrix:\n");

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

{

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

{

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

}

}

printf("\n Enter the source node:\t");

scanf("%d", &src);

dijkstras();

}

void dijkstras()

{

int vis[10], dist[10], u, j, count, min;

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

{

dist[j] = c[src][j];

}

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

{

vis[j] = 0;

}

dist[src] = 0;

vis[src] = 1;

count = 1;

while (count != n)

{

min = 9999;

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

{

if (dist[j] < min && vis[j] != 1)

{

min = dist[j];

u = j;

}

}

vis[u] = 1;

count++;

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

{

if (min + c[u][j] < dist[j] && vis[j] != 1)

{

dist[j] = min + c[u][j];

}

}

}

printf("\nthe shortest distance is: \n");

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

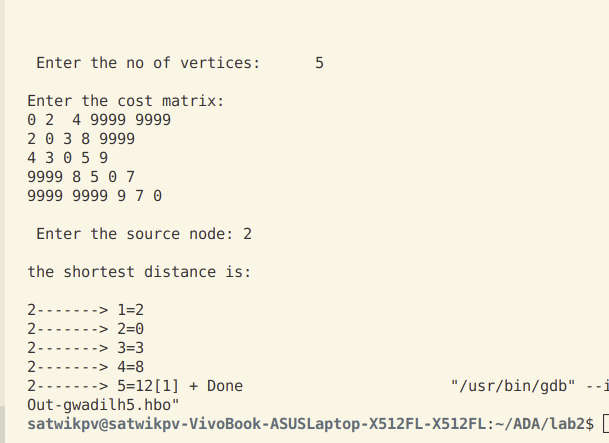
{

printf("\n%d-------> %d=%d", src, j, dist[j]);

}

}

Output:



## 16. Heap Sort

#include <stdio.h>

// #include <conio.h>

void Adjust(int Heap\_of\_Numbers[], int i)

{

int j;

int copy;

int Number;

int Reference = 1;

Number = Heap\_of\_Numbers[0];

while (2 \* i <= Number && Reference == 1)

{

j = 2 \* i;

if (j + 1 <= Number && Heap\_of\_Numbers[j + 1] > Heap\_of\_Numbers[j])

j = j + 1;

if (Heap\_of\_Numbers[j] < Heap\_of\_Numbers[i])

Reference = 0;

else

{

copy = Heap\_of\_Numbers[i];

Heap\_of\_Numbers[i] = Heap\_of\_Numbers[j];

Heap\_of\_Numbers[j] = copy;

i = j;

}

}

}

void Make\_Heap(int heap[])

{

int i;

int Number\_of\_Elements;

Number\_of\_Elements = heap[0];

for (i = Number\_of\_Elements / 2; i >= 1; i--)

Adjust(heap, i);

}

int main()

{

int heap[30];

int NumberofElements;

int i;

int LastElement;

int CopyVariable;

printf("Enter the number of elements present in the unsorted Array:");

scanf("%d", &NumberofElements);

printf("nEnter the members of the array one by one:"); /\* Asking for the elements of the unsorted array\*/

for (i = 1; i <= NumberofElements; i++)

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

heap[0] = NumberofElements;

Make\_Heap(heap);

while (heap[0] > 1)

{

LastElement = heap[0];

CopyVariable = heap[1];

heap[1] = heap[LastElement];

heap[LastElement] = CopyVariable;

heap[0]--;

Adjust(heap, 1);

}

printf("nSorted Array:n");

for (i = 1; i <= NumberofElements; i++)

printf("%d ", heap[i]);

return 0;

}

Output:

