1. Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to perform selection sort on an array

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

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

{

min\_idx = i; // Assume the current element is the minimum

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

{

if (arr[j] < arr[min\_idx])

{

min\_idx = j; // Update min\_idx if a smaller element is found

}

}

// Swap the found minimum element with the current element

int temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

// Function to generate an array of random numbers

void generateRandomNumbers(int arr[], int n)

{

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

{

arr[i] = rand() % 10000; // Generate random numbers between 0 and 9999

}

}

int main()

{

int n,arr[10000];

printf("Enter number of elements: ");

scanf("%d", &n); // Read the number of elements from the user

if (n <= 5000)

{

printf("Please enter a value greater than 5000\n");

return 1; // Exit if the number of elements is not greater than 5000

}

// Generate random numbers and store them in the array

generateRandomNumbers(arr, n);

// Measure the time taken to sort the array

clock\_t start = clock();

selectionSort(arr, n);

clock\_t end = clock();

// Calculate and print the time taken to sort the array

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to sort %d elements: %f seconds\n", n, time\_taken);

// Free the allocated memory

free(arr);

return 0;

}

Output:

Enter number of elements: 6000

Time taken to sort 6000 elements: 0.031000 seconds

Enter number of elements: 7000

Time taken to sort 7000 elements: 0.034000 seconds

Enter number of elements: 8000

Time taken to sort 8000 elements: 0.047000 seconds

Enter number of elements: 9000

Time taken to sort 9000 elements: 0.052000 seconds

2. Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

#include <stdio.h> //

#include <time.h> //clock()

#include<stdlib.h>//rand()

int merge(int b[], int c[], int a[], int p, int q, int n)

{

int i,j,k;

i=j=k=0;

while(i<p && j<q)

{

if(b[i]<=c[j])

{

a[k]=b[i];

i++;

}

else

{

a[k]=c[j];

j++;

}

k++;

}

if(i==p)

{

while(j<q)

{

a[k]=c[j];

k++;

j++;

}

}

else

{

while(i<p && k<n)

a[k++]=b[i++];

}

}

int mergesort(int a[],int n)

{

int b[n/2];

int c[n-n/2];

int i, j;

if(n>1)

{

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

b[i]=a[i];

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

c[j]=a[i];

mergesort(b, n/2);

mergesort(c, n-n/2);

merge(b, c, a, n/2 ,n-n/2, n);

}

}

int main()

{

// Calculate the time taken by fun()

int temp,min,j,i,n,a[100000],choicer;

clock\_t t;

printf("enter the number of elements");

scanf("%d",&n);//n=5

printf("Random number generator");

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

{

a[i]=rand()%1000;//11111%1000=111

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

}

t = clock(); // t=1000

mergesort(a,n);

t = clock() - t; //1003-1000=3

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC; // 3.000000/18.2

printf("entered numbers are after sorting\n");

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

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

printf("sort function took %f seconds to execute \n",time\_taken);

return 0;

}

Output:

Enter number of elements: 6000

Time taken to sort 6000 elements: 0.000709 seconds

Enter number of elements: 7000

Time taken to sort 7000 elements: 0.000752 seconds

Enter number of elements: 8000

Time taken to sort 8000 elements: 0.000916 seconds

Enter number of elements: 9000

Time taken to sort 9000 elements: 0.001493 seconds

3. Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.

#include<stdio.h>

#include<conio.h>

int temp[10],k=0;

void sort(int a[][10],int id[],int n)

{

int i,j;

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

{

if(id[i]==0)

{

id[i]=-1;

temp[++k]=i;

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

{

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

id[j]--;

}

i=0;

}

}

}

void main()

{

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

printf("\nEnter the n value:");

scanf("%d",&n);

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

id[i]=0;

printf("\nEnter the graph data:\n");

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

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

{

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

if(a[i][j]==1)

id[j]++;

}

sort(a,id,n);

if(k!=n)

printf("\nTopological ordering not possible");

else

{

printf("\nTopological ordering is:");

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

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

}

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPU1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the n value:6

Enter the graph data:

0 0 1 1 0 0

0 0 0 1 1 0

0 0 0 1 0 1

0 0 0 0 0 1

0 0 0 0 0 1

0 0 0 0 0 0

Topological ordering is: 1 2 3 4 5 6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPU2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the n value:4

Enter the graph data:

1 4 3 2

5 4 2 1

5 3 4 2

4 1 2 3

Topological ordering not possible

4. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim’s algorithm.

#include <stdio.h>

int main()

{

int mincost = 0, n, i, j, ne, a = 0, b = 0, min, u = 0, v = 0;

int cost[10][10], visited[10];

printf("Enter the number of vertices\n");

scanf("%d", &n);

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

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

{

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

{

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

if(cost[i][j] == 0)

cost[i][j] = 999;

}

}

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

visited[i] = 0;

visited[1] = 1;

ne = 1;

while(ne < n)

{

for(min = 999, i = 1; i <= n; i++)

{

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

{

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

{

if(visited[i] == 0)

continue;

else

{

min = cost[i][j];

a = u = i;

b = v = j;

}

}

}

}

if(visited[u] == 0 || visited[v] == 0)

{

printf("\n %d edge (%d, %d)=%d", ne++, a, b, min);

mincost = mincost + min;

visited[v] = 1;

}

cost[a][b] = cost[b][a] = 999;

}

printf("\nThe minimum cost of spanning tree is = %d", mincost);

return 0;

}

Output:

5. Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra’s algorithm.

#include <stdio.h>

#include <stdlib.h>

#define INF 999

void dijikstras(int cost[][100], int dist[], int n, int v);

int main() {

int n, source, i, j;

printf("Enter the number of vertices\n");

scanf("%d", &n);

int cost[100][100];

int dist[100];

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

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

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

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

if(cost[i][j] == 0)

cost[i][j] = INF;

}

printf("source\n");

scanf("%d", &source);

dijikstras(cost, dist, n, source);

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

if(source != i)

printf("%d->%d::%d\n", source, i, dist[i]);

return 0;

}

void dijikstras(int cost[][100], int dist[], int n, int v) {

int i, u = 0, w, count, min;

int flag[100] = {0};

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

flag[i] = 0;

dist[i] = cost[v][i];

}

flag[v] = 1;

dist[v] = 0;

count = 2;

while(count < n) {

for(i = 1, min = INF; i <= n; i++) {

if((dist[i] < min) && (flag[i] == 0)) {

min = dist[i];

u = i;

}

}

flag[u] = 1;

count++;

for(w = 1; w <= n; w++) {

if((dist[u] + cost[u][w] < dist[w]) && (flag[w] == 0))

dist[w] = dist[u] + cost[u][w];

}

}

}

6. Design and implement C/C++ Program to find the transitive closure using Warshal’s algorithm.

#include<stdio.h>

void warsh(int p[][10],int n)

{

int i,j,k;

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

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

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

p[i][j]=p[i][j] || p[i][k] && p[k][j];

}

int main()

{

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

printf("\nEnter the n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

warsh(a,n);

printf("\nResultant path matrix\n");

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

{

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

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

printf("\n");

}

return 0;

}

Output:

Enter the n value:4

Enter the graph data:

0 1 0 0

0 0 0 1

0 0 0 0

1 0 1 0

Resultant path matrix

1 1 1 1

1 1 1 1

0 0 0 0

1 1 1 1

7. . Design and implement C/C++ Program to find a matching sub string in a given text using Horspool's Algorithm

#include <stdio.h>

#include <string.h>

const int MAX = 256;

int fnHorspool(char string[], char pattern[],int []);

void fnGenShiftTable(char \*,int []);

int main(void)

{

char text[MAX];

char pattern[MAX];

int shiftTable[MAX];

int found;

puts("Enter the source string : ");

gets(text);

puts("Enter the pattern string : ");

gets(pattern);

fnGenShiftTable(pattern,shiftTable);

found = fnHorspool(text,pattern,shiftTable);

if(found==-1)

puts("\nMatching Substring not found.\n");

else

printf("\nMatching Substring found at position: %d\n",found+1);

return 0;

}

void fnGenShiftTable(char p[], int t[])

{

int m, i, j;

m = strlen(p);

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

{

t[i]=m;

}

for(j=0; j<m-1; j++)

{

t[p[j]] = m-1-j;

}

}

int fnHorspool(char s[],char p[],int t[])

{

int i, n, m, k;

n = strlen(s);

m = strlen(p);

i = m-1;

while(i<n)

{

k = 0;

while((k<m)&&(p[m-1-k]==s[i-k]))

k++;

if (k == m)

return i-m+1;

else

i = i+t[s[i]];

}

return -1;

}

Output:

Enter the source string :

I want you to rebel!

Enter the pattern string :

you

Matching Substring found at position: 8

Enter the source string :

Be a rebel!

Enter the pattern string :

coward

Matching Substring not found.

8. Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

#include<stdio.h>

int w[10],p[10],n;

int max(int a,int b)

{

return a>b?a:b;

}

int knap(int i,int m)

{

if(i==n) return w[i]>m?0:p[i];

if(w[i]>m) return knap(i+1,m);

return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);

}

int main()

{

int m,i,max\_profit;

printf("\nEnter the no. of objects:");

scanf("%d",&n);

printf("\nEnter the knapsack capacity:");

scanf("%d",&m);

printf("\nEnter profit followed by weight:\n");

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

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

max\_profit=knap(1,m);

printf("\nMax profit=%d",max\_profit);

return 0;

}

Output:

Enter the no. of objects:4

Enter the knapsack capacity:5

Enter profit followed by weight:

12 3

43 5

45 2

55 3

Max profit=100

9. Design and implement C/C++ Program for N Queen’s problem using Backtracking.

#include <stdio.h>

#include <stdlib.h>

int x[10];

int place(int k,int i)

{

int j;

for(j=1;j<=k-1;j++)

if(x[j]==i || abs(x[j]-i)==abs(j-k))

return 0;

return 1;

}

void display(int n)

{

int k,i,j;

char cb[n][n];

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

cb[k][x[k]]='Q';

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

{

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

{

if(j!=x[i])

cb[i][j]='-';

}

}

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

{

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

printf("%c\t",cb[i][j]);

printf("\n");

}

printf("\n\n");

}

void NQueens(int k,int n)

{

int i;

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

if(place(k,i))

{

x[k]=i;

if(k==n)

{

printf("Solution\n");

display(n);

}

else

NQueens(k+1,n);

}

}

int main(void)

{

int n,k=1;

printf("Enter the dimensions of the chessboard\n");

scanf("%d",&n);

if(n==2 || n==3)

{

printf("No solution\n");

exit(0);

}

NQueens(k,n);

return 0;

}

Ouput :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT-1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the number of queens: 4

# # Q #

Q # # #

# # # Q

# Q # #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT-2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the number of queens: 3

Solution does not exist