



CITY ENGINEERING COLLEGE

Doddakallasandra, off Kanakapura Main Road,
Bangalore – 560061

Affiliated to Visvesvaraya Technological University,
Belagavi & Approved by AICTE, New Delhi



LAB MANUAL

DESIGN AND ANALYSIS OF ALGORITHM LABORATORY

(BCSL404)

Integrated Lab Component

for

IV SEMESTER

NAME: _____

USN: _____

SEMESTER: _____

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR: 2024

CITY ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

To contribute to Global Development by producing Knowledgeable and Quality professionals who are Innovative and Successful in advanced field of Computer Science & Engineering to adapt the changing Employment demands and social needs.

MISSION

M1: To provide Quality Education for students, to build Confidence by developing their Technical Skills to make them Competitive Computer Science Engineers.

M2: To facilitate Innovation & Research for students and faculty and to provide Internship opportunities

M3: To Collaborate with educational institutions and industries for Excellence in Teaching and Research.

Analysis & Design of Algorithms Lab		Semester	4
Course Code	BCSL404	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	2
Examination type (SEE)	Practical		
Course objectives: <ul style="list-style-type: none">• To design and implement various algorithms in C/C++ programming using suitable development tools to address different computational challenges.• To apply diverse design strategies for effective problem-solving.• To Measure and compare the performance of different algorithms to determine their efficiency and suitability for specific tasks.			
Sl.No	Experiments		
1	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.		
2	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.		
3	a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.		
4	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.		
5	Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.		
6	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.		
7	Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.		
8	Design and implement C/C++ Program to find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d .		
9	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.		
10	Design and implement C/C++ Program to sort a given set of n integer elements using Quick 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.		
11	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.		
12	Design and implement C/C++ Program for N Queen's problem using Backtracking.		

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

1. Develop programs to solve computational problems using suitable algorithm design strategy.
2. Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).
3. Make use of suitable integrated development tools to develop programs
4. Choose appropriate algorithm design techniques to develop solution to the computational and complex problems.
5. Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.

- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in - 60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

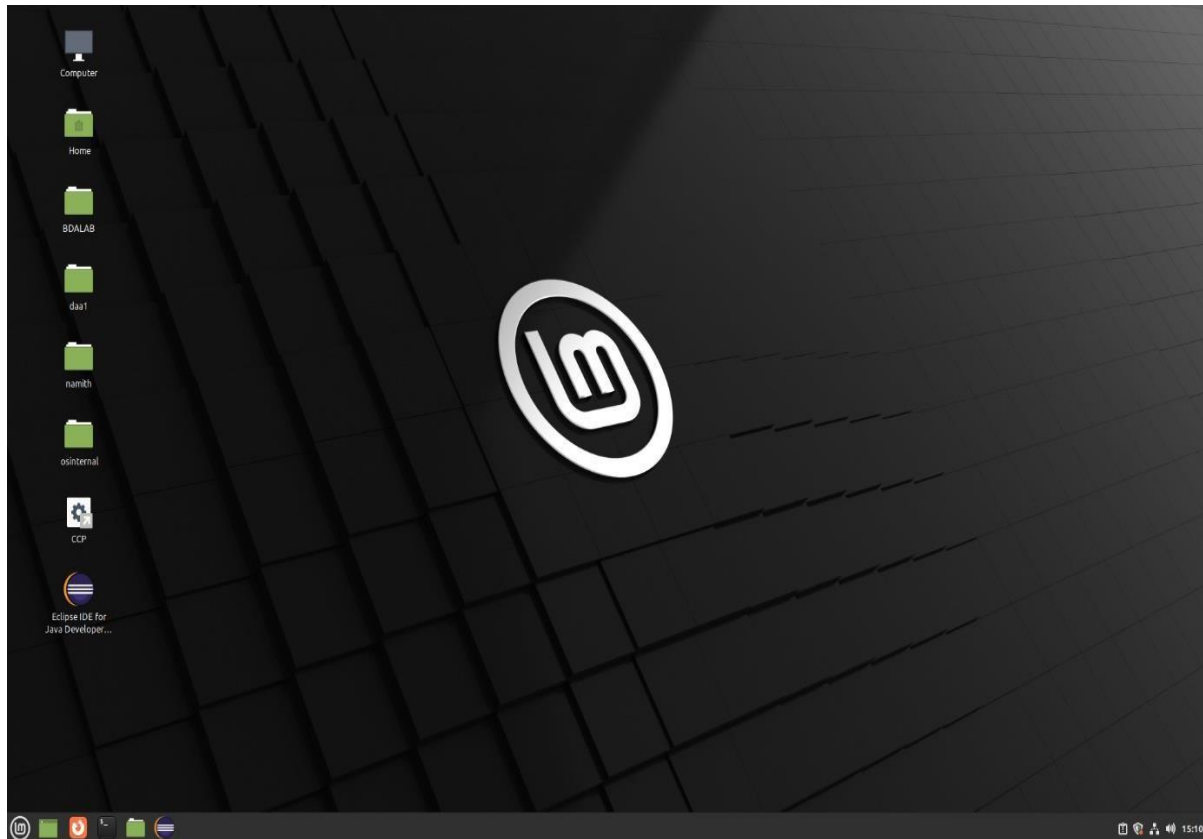
Suggested Learning Resources:

- Virtual Labs (CSE): <http://cse01-iiith.vlabs.ac.in/>

Steps for creation and execution of project using Eclipse:

Step 1: Initial step for project creation.

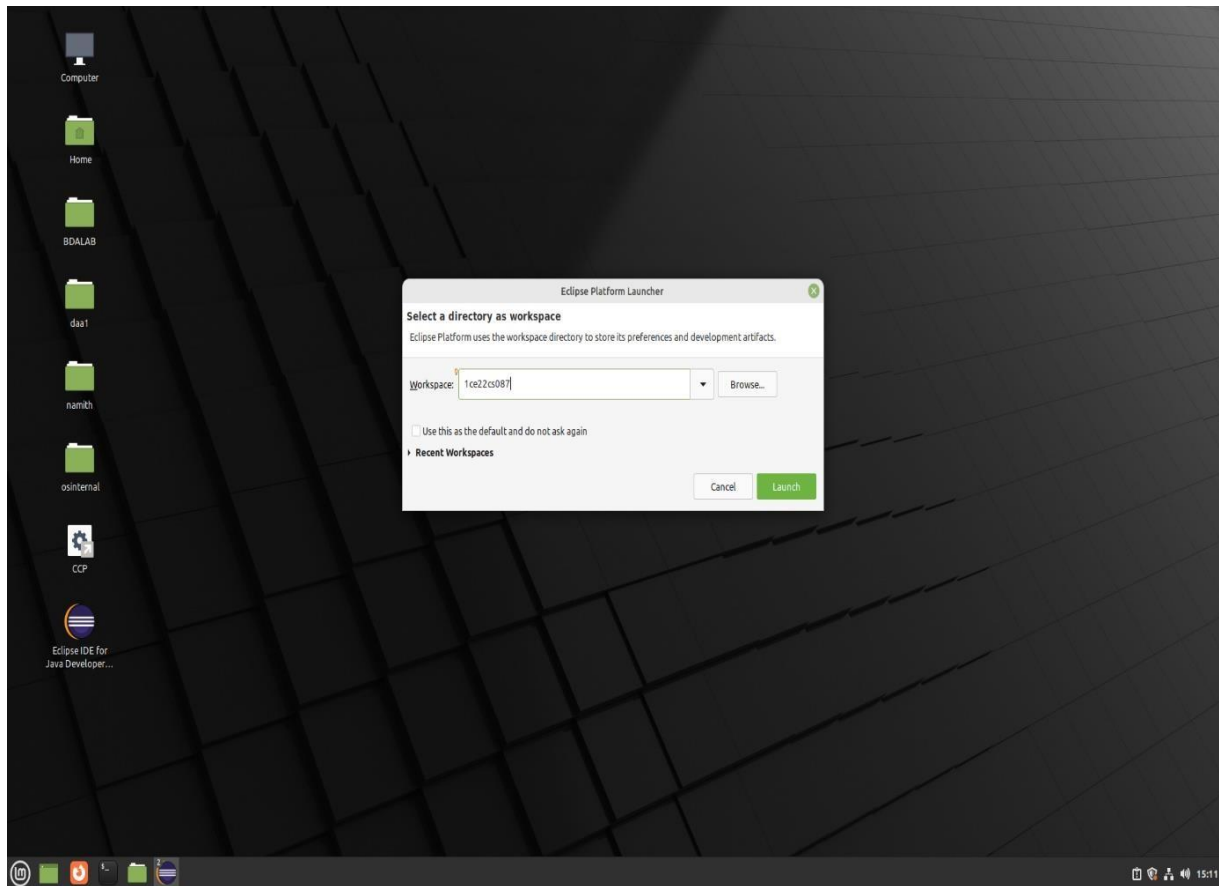
With **Linux** as operating system and using **Eclipse IDE** as software, the programs shall be executed using C language. As seen in the snapshot 1.1 double click on **CCP executable file** which is the initial step to start the project.



Snapshot 1.1

Step 2: Creation of workspace.

An **Eclipse Launcher Platform** will appear on the display, this eclipse platform uses the **workspace directory** to store its preferences and development artifacts. The user can create a folder on desktop and specify the path in the **workspace** given by giving a **project name**, e.g.: 1ce22cs001 or daaprogram1 etc...



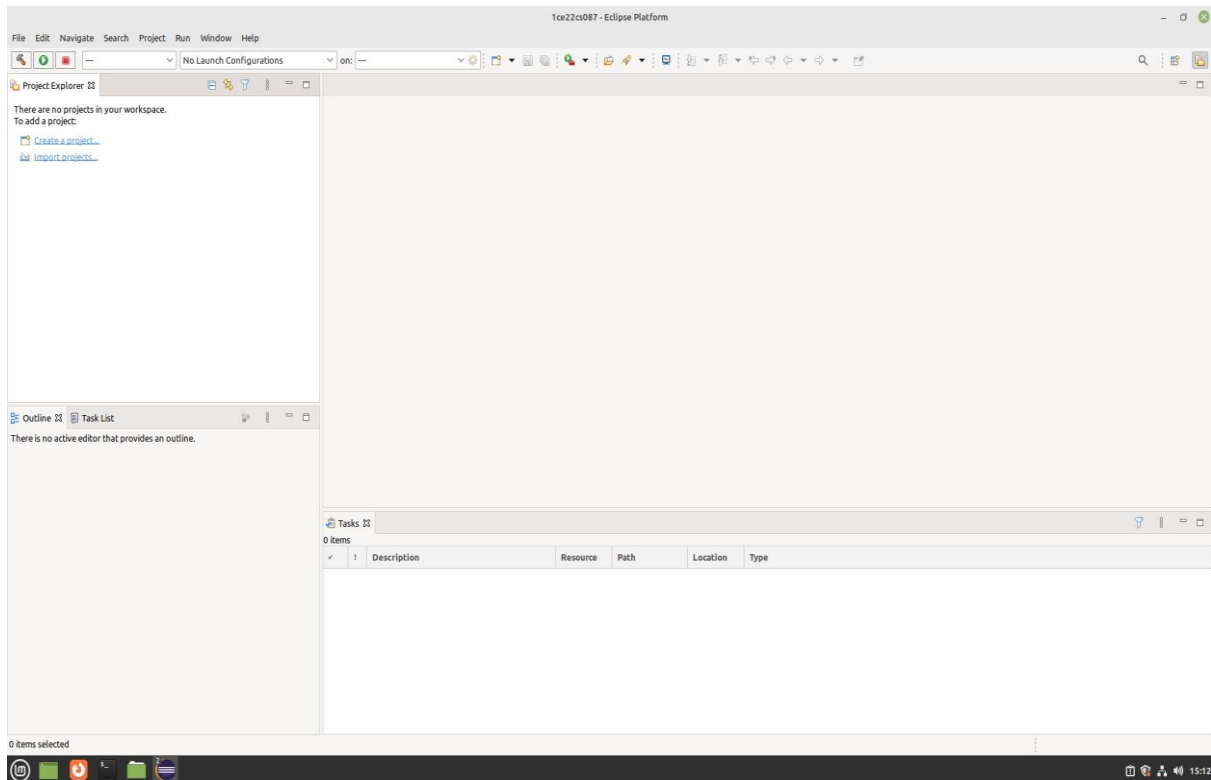
Snapshot 1.2

Step 3: Usage of Eclipse Platform

This is the workbench where the project is been created and executed, in this Eclipse platform on the left side we can find project explorer. Here we can create and view project, the main class is also created under project in this particular section. After creation of project, the editor will appear where we can place the program code.

In the top section we can see toolbars where we can use various tools for project execution. Here we use build project for checking debugging and run for program execution.

In the bottom section we come across console to view the output. The errors, warnings also appear in this section.



Snapshot 1.3

Step 4: Creation of Project in Eclipse IDE

First step is to click on File ->

New ->

Project -> Choose C/C++ -> double click on it ->

Choose C Project -> Double click on C/C++ project -> Choose C project ->

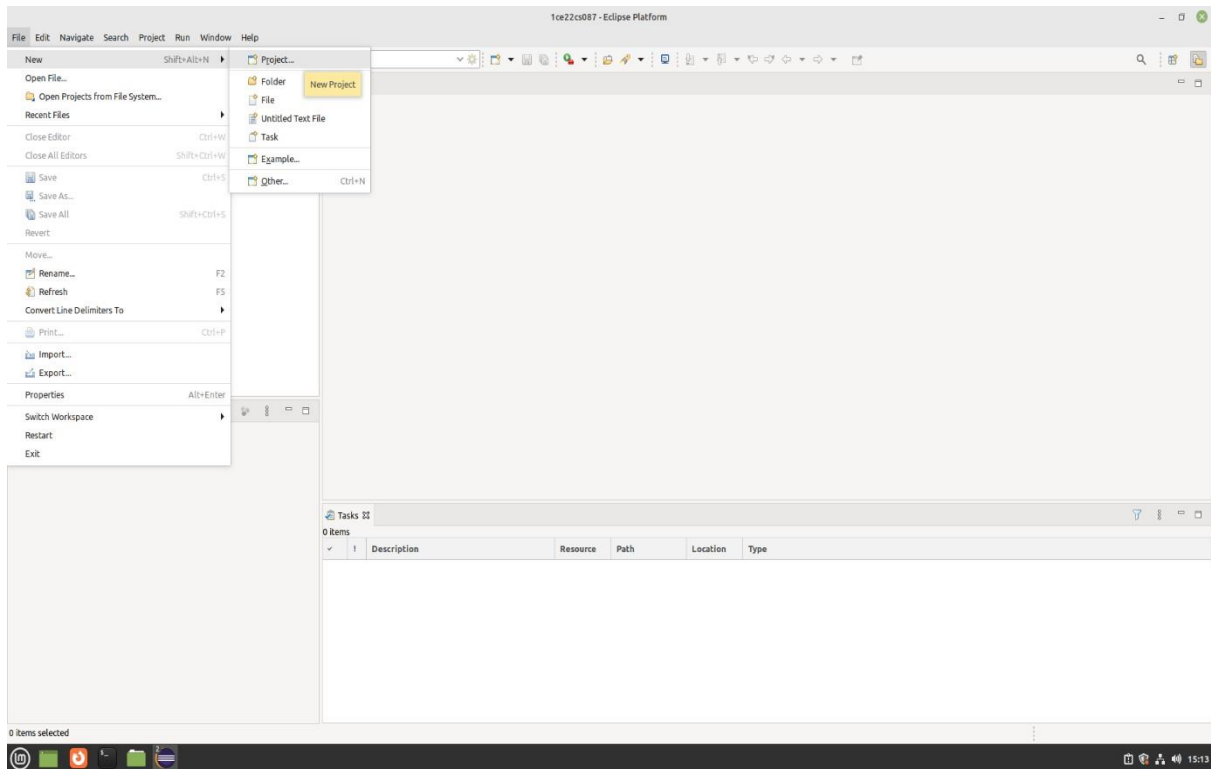
And click on next ->

Give a project name (the project name should not have special character, space and capitals letters) ->

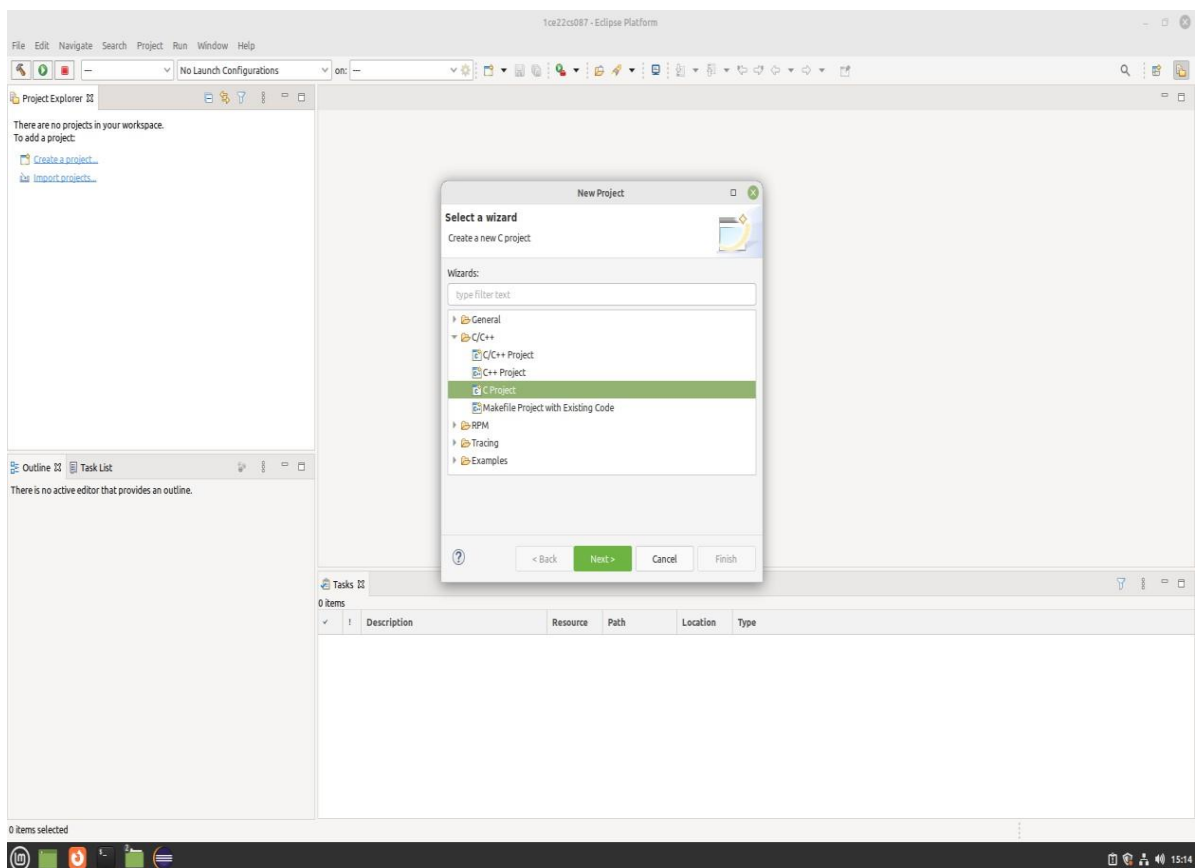
Choose Linux GCC -> and click on Finish ->

Click on Open perspective.

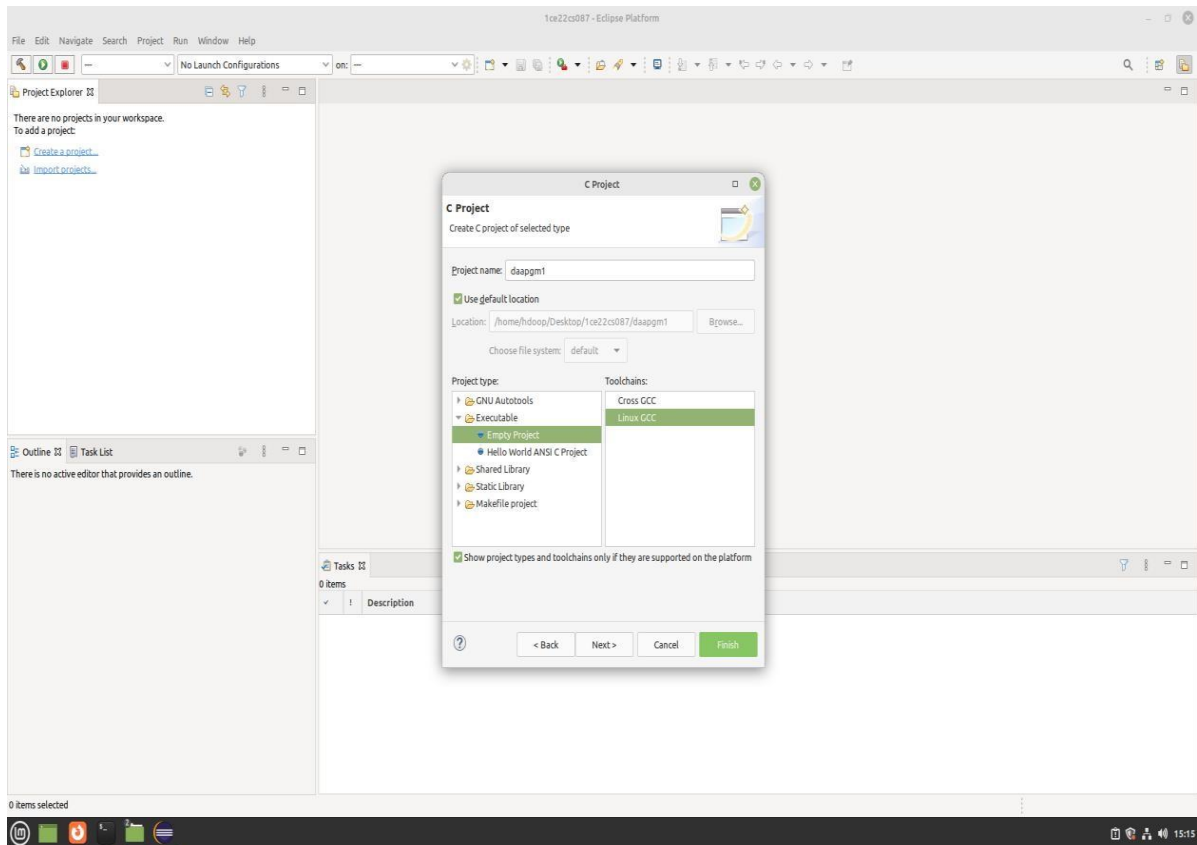
(Refer Snapshot from 1.4 to 1.7)



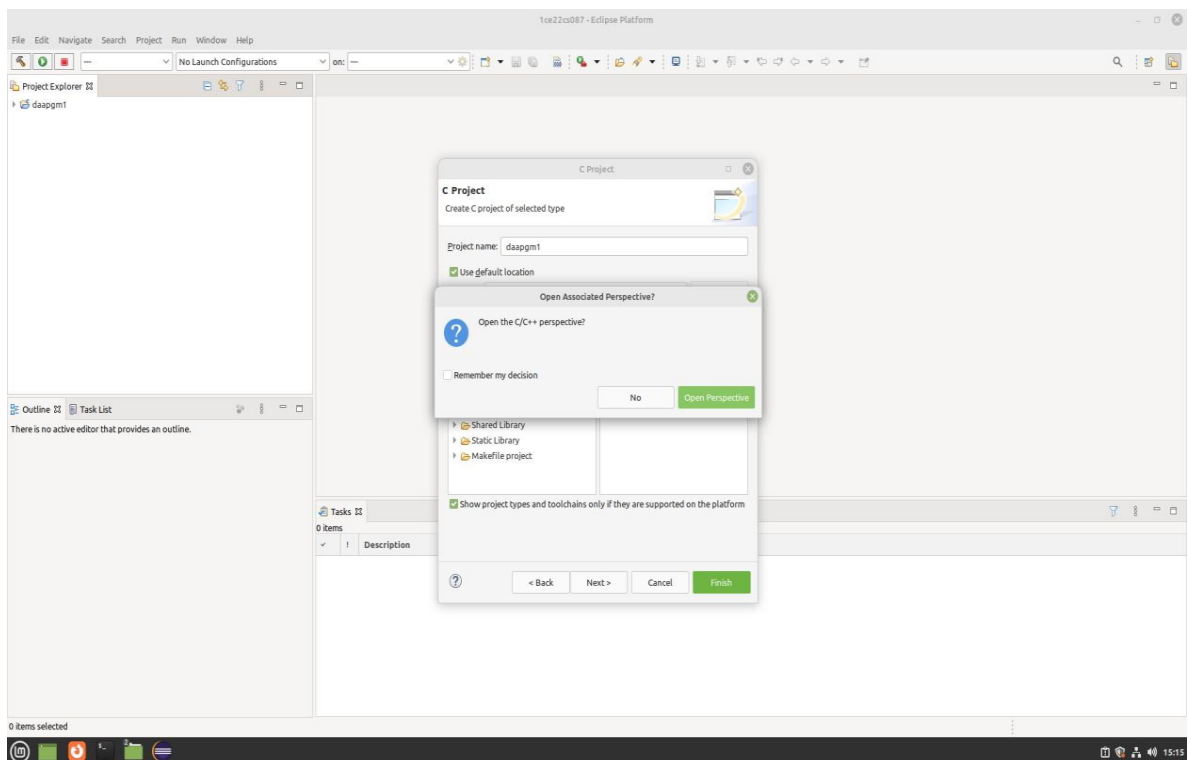
Snapshot 1.4



Snapshot 1.5



Snapshot 1.6



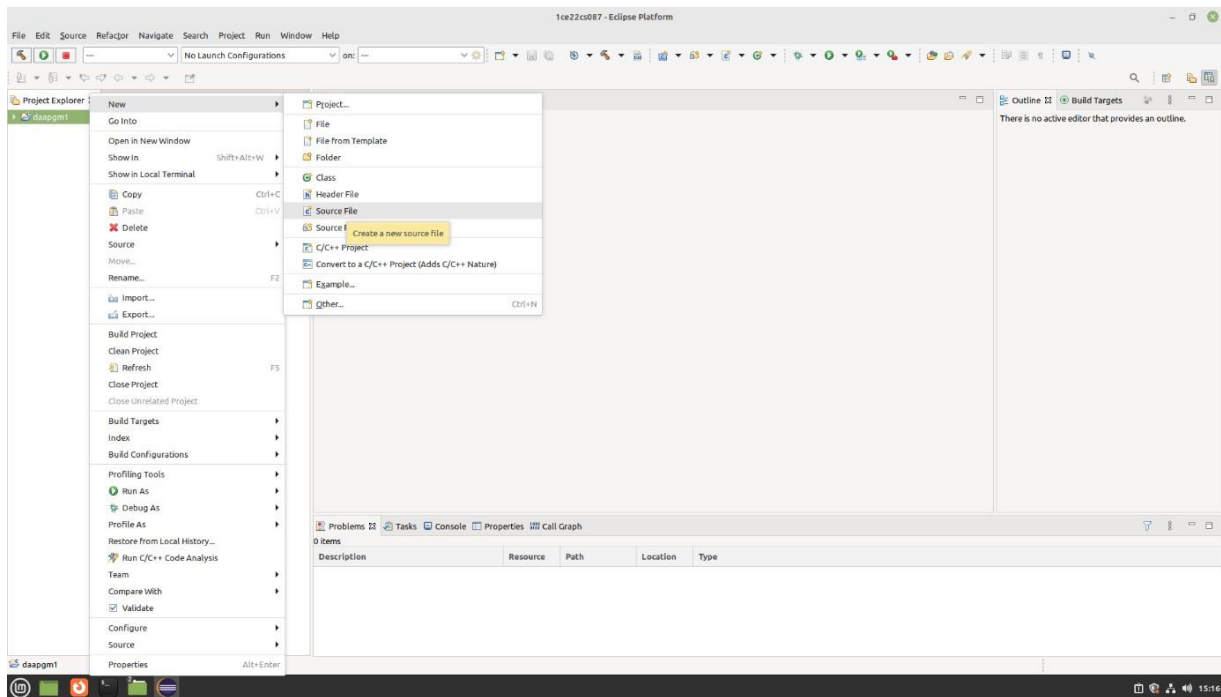
Snapshot 1.7

Step 5: Creation of main class in Eclipse IDE

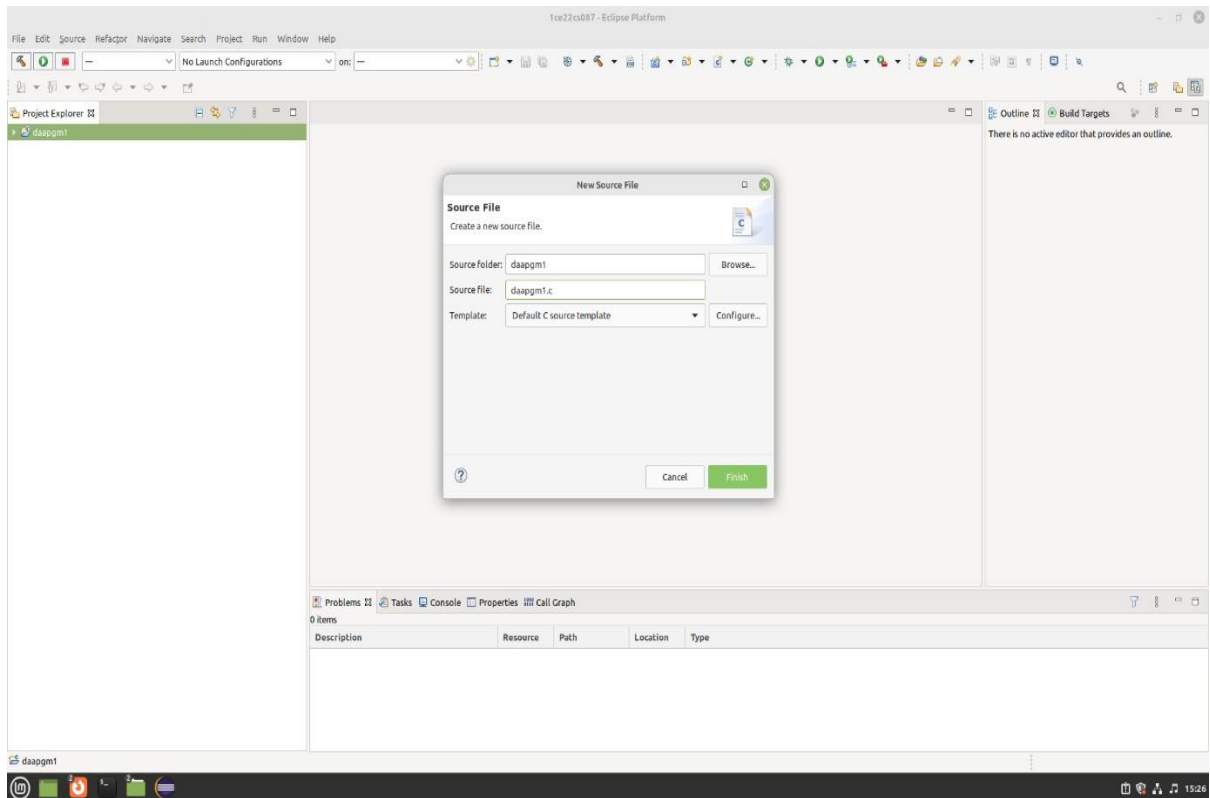
Right click on project name ->

Click on New ->

Click on Source File -> in source file save the main class name with the same project name using “.c” extension and click on finish (Refer Snapshot 1.8 and 1.9)



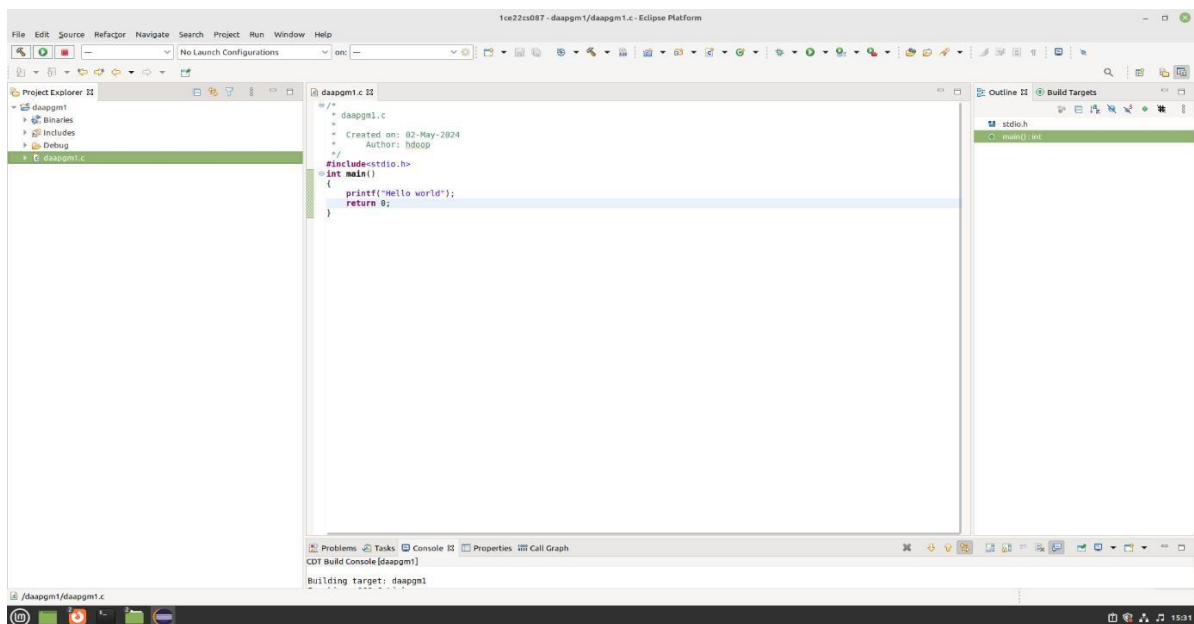
Snapshot 1.8



Snapshot 1.9

Step 6: Program Execution

Once the main class is created the editor will appear where the program code is executed.



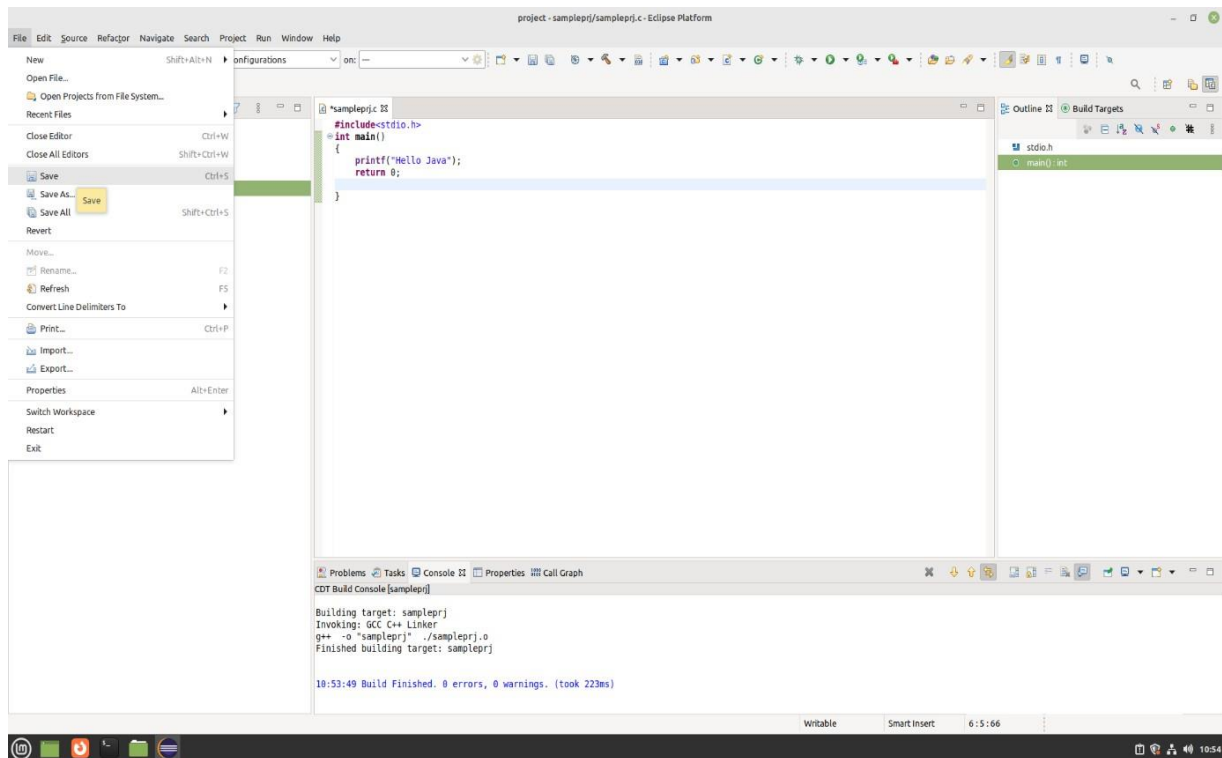
Snapshot 1.10

Step 7: Saving project

Once the typing of program code is done

Click on File ->

and click on Save or else just click Ctrl +S, to save the program.



Snapshot 1.11

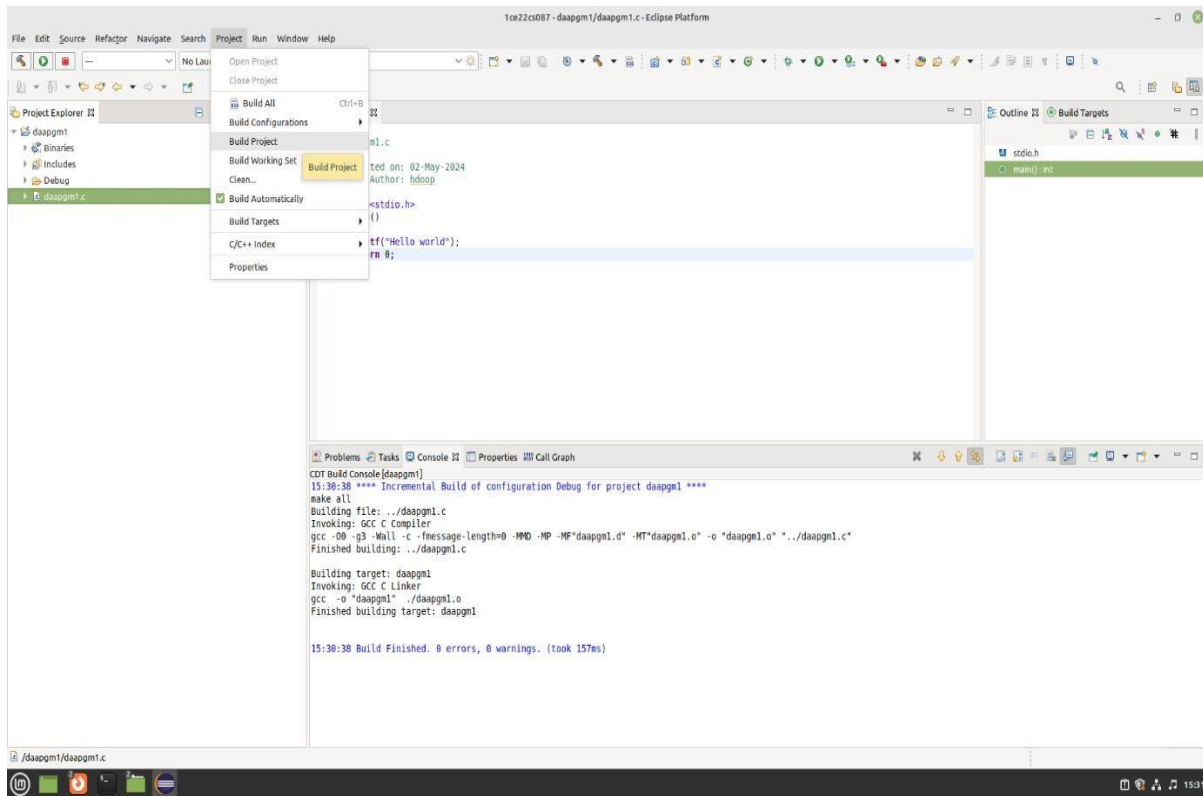
Step 8: Debugging project:

Click on Project ->

Click on Build Project.

If there are no errors in the program, there will be 0 errors and 0 warnings in the console.

If there are errors in the program, in console the errors and warnings are shown in that specific line.



Snapshot 1.12

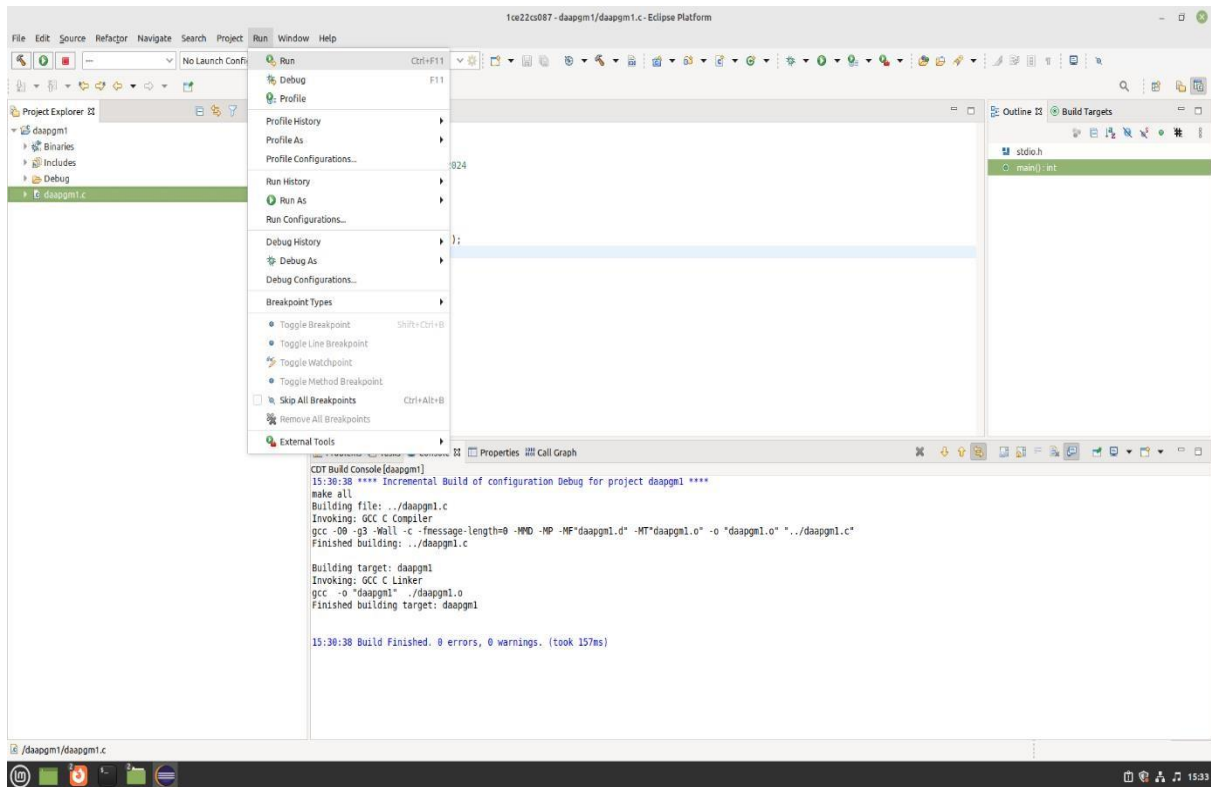
Step 9: Run the Program

Once there are no errors in the program

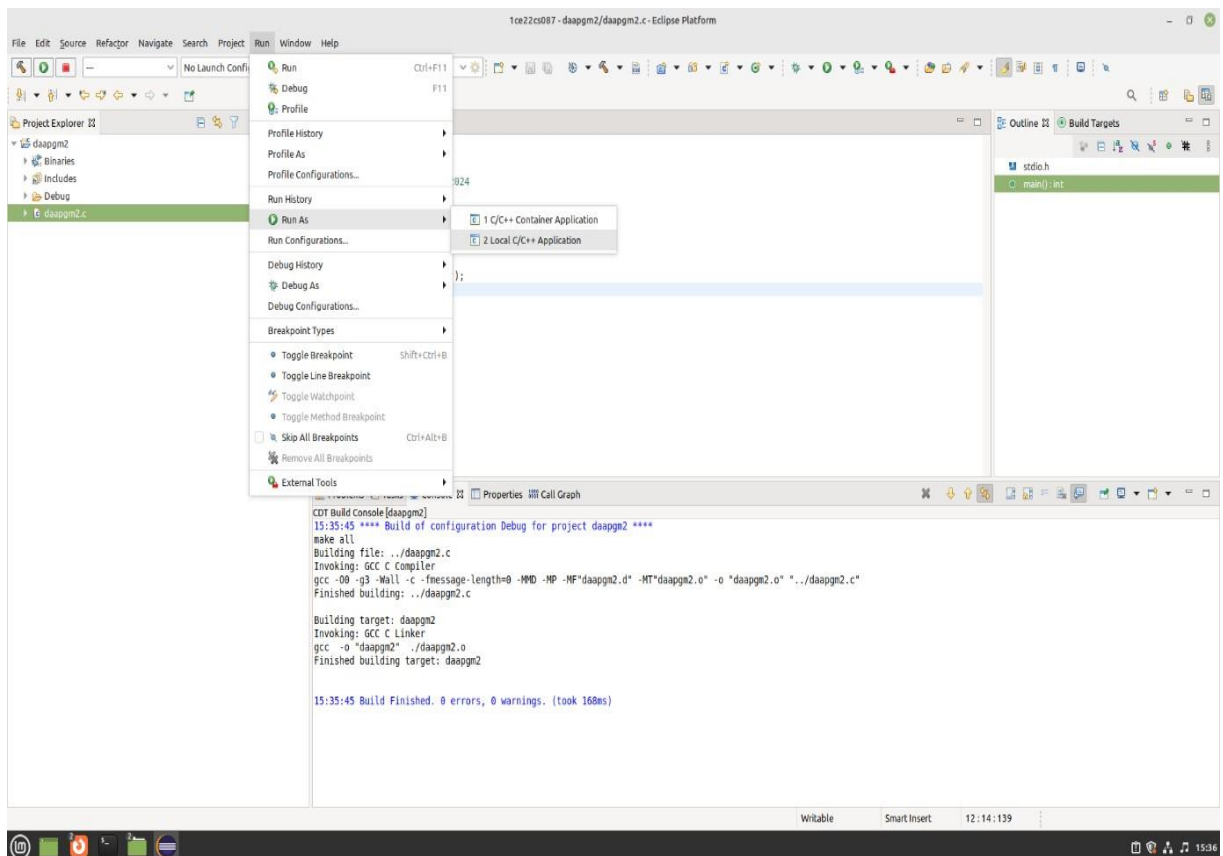
Click on -> Run or Run As ->

Choose second option Local C/C++ Application.

(Refer snapshot 1.13 & 1.14)



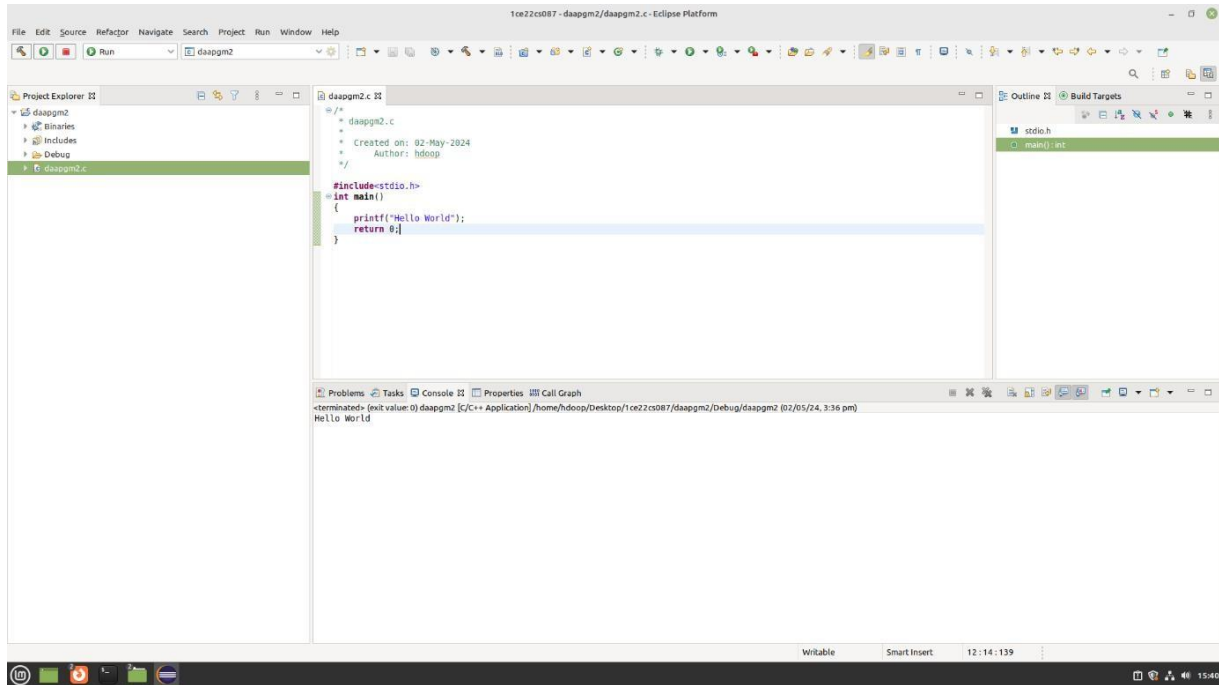
Snapshot 1.13



Snapshot 1.14

Step 10: To check program output.

In the console section the output will be executed.



Snapshot 1.15

PROGRAM: 1

AIM: Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

DEFINITION: Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

If the graph is not connected then it finds a minimum spanning forest. It is an example of a greedy algorithm.

ALGORITHM: Start with an empty set A, and select at every stage the shortest edge that has not been chosen or rejected, regardless of where this edge is situated in graph.

- Initially, each vertex is in its own tree in forest.
- Then, algorithm consider each edge in turn, order by increasing weight.
- If an edge (u, v) connects two different trees, then (u, v) is added to the set of edges of the MST, and two trees connected by an edge (u, v) are merged into a single tree.
- On the other hand, if an edge (u, v) connects two vertices in the same tree, then edge (u, v) is discarded.

Kruskal's algorithm can be implemented using disjoint set data structure or priority queue data structure. Kruskal's algorithm implemented with disjoint-sets data structure.

MST_KRUSKAL (G, w)

1. $A \leftarrow \{ \}$ // A will ultimately contain the edges of the MST
2. for each vertex v in $V[G]$
3. do Make Set (v)
4. Sort edge of E by nondecreasing weights w
5. for each edge (u, v) in E
6. do if FIND_SET (u) \neq FIND_SET (v)
7. then $A = A \cup \{ (u, v) \}$
8. UNION (u, v)
9. Return A

PROGRAM CODE:

```
#include<stdio.h>
#define INF 999
#define MAX 100
int p[MAX], c[MAX][MAX],t[MAX][2];
int find (int v)
{
while(p[v])
v=p[v];
return v;
}
void union1(int i,int j)
{
p[j]=i;
}
void kruskal(int n)
{
int i,j,k,u,v,min,res1,res2,sum=0;
for(k=1;k<n;k++)
{
min=INF;
for(i=1;i<n-1;i++)
{
for(j=1;j<=n;j++)
{
if(i==j)continue;
if(c[i][j]<min)
{
u=find(i);
v=find(j);
if(u!=v)
{
res1=i;
res2=j;
min=c[i][j];
}
}
}
}
union1(res1, find(res2));
t[k][1] =res1;
```

DAA LAB (BCSL404)

```
t[k][2]=res2;
sum=sum+min;
}
printf("\nCost of spanning tree is=%d",sum);
printf("\nEdgesof spanning tree are:\n");
for(i=1;i<n;i++)
printf("%d -> %d\n",t[i][1],t[i][2]);
}
int main()
{
int i,j,n;
printf("\nEnter the n value:");
scanf("%d",&n);
for(i=1;i<=n;i++)
p[i]=0;
printf("\nEnter the graph data:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&c[i][j]);
kruskal(n);
return 0;
}
```

OUTPUT:

```
Enter the n value:5
Enter the graph data:
0 10 15 9 999
10 0 999 17 15
15 999 0 20 999
9 17 20 0 18
999 15 999 18 0
Cost of spanning tree is=49
```

```
Edges of spanning tree are:
1 -> 4
1 -> 2
1 -> 3
2 -> 5
```

PROGRAM: 2

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

DEFINITION: Prim's is an algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all edges in the tree is minimized. Prim's algorithm is an example of a greedy algorithm.

ALGORITHM:

MST_PRIM (G, w, v)

1. $Q \leftarrow V[G]$
2. for each u in Q do
3. $\text{key}[u] \leftarrow \infty$
4. $\text{key}[r] \leftarrow 0$
5. $\pi[r] \leftarrow \text{Nil}$
6. while queue is not empty do
7. $u \leftarrow \text{EXTRACT_MIN}(Q)$
8. for each v in $\text{Adj}[u]$ do
9. if v is in Q and $w(u, v) < \text{key}[v]$
10. then $\pi[v] \leftarrow w(u, v)$
11. $\text{key}[v] \leftarrow w(u, v)$

PROGRAM CODE:

```
#include<stdio.h>
#define INF 999
int prim(int c[10][10],int n,int s)
{
int v[10],i,j,sum=0,ver[10],d[10],min,u;
for(i=1;i<=n;i++)
{
ver[i]=s;
d[i]=c[s][i];
v[i]=0;
}
v[s]=1;
for(i=1;i<=n-1;i++)
```

```
{
min=INF;
for(j=1;j<=n;j++)
if(v[j]==0 && d[j]<min)
{
min=d[j];
u=j;
}
v[u]=1;
sum=sum+d[u];
printf("\n%d -> %d sum=%d",ver[u],u,sum);
for(j=1;j<=n;j++)
if(v[j]==0 && c[u][j]<d[j])
{
d[j]=c[u][j];
ver[j]=u;
}
}
return sum;
}
void main()
{
int c[10][10],i,j,res,s,n;
printf("\nEnter 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",&c[i][j]);
printf("\nEnter the source node:");
scanf("%d",&s);
res=prim(c,n,s);
printf("\nCost=%d",res);
}
```

OUTPUT:

```
Enter n value:3
Enter the graph data:
0 10 1
10 0 6
1 6 0
Enter the source node:1
```

DAA LAB (BCSL404)

1 -> 3 sum=1

3 -> 2 sum=7

Cost=7

PROGRAM: 3A

AIM: Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

DEFINITION: The Floyd algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph (with positive or negative edge weights). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices though it does not return details of the paths themselves. The algorithm is an example of dynamic programming.

ALGORITHM:

Floyd's Algorithm

Accept no. of vertices

Call graph function to read weighted graph // $w(i,j)$

Set $D[] <-$ weighted graph matrix // get $D \{d(i,j)\}$ for $k=0$

// If there is a cycle in graph, abort. How to find?

Repeat for $k = 1$ to n

 Repeat for $i = 1$ to n

 Repeat for $j = 1$ to n

$D[i,j] = \min \{D[i,j], D[i,k] + D[k,j]\}$

 Print D

PROGRAM CODE:

```
#include<stdio.h>
#define INF 999
int min(int a, int b)
{
    return(a<b)?a:b;
}
void floyd(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]=min(p[i][j],p[i][k]+p[k][j]);
}
void main()
{
```

DAA LAB (BCSL404)

```
int a[10][10],n,i,j;
printf("\n Enter the n value:");
scanf("%d",&n);
printf("\n Enter the graph data:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
floyd(a,n);
printf("\n Shortest path matrix\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%d ",a[i][j]);
printf("\n");
}
}
```

OUTPUT:

Enter the n value:4

Enter the graph data:

0 999 3 999

2 0 999 999

999 7 0 1

6 999 999 0

Shortest path matrix

0 10 3 4

2 0 5 6

7 7 0 1

6 16 9 0

PROGRAM: 3B

AIM: Design and implement C/C++ Program to find the transitive closure using Warshall's algorithm.

DEFINITION: The Floyd Warshall's algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph. A single execution of the algorithm will find the lengths of the shortest path between all pairs of vertices though it does not return details of the paths themselves. The algorithm is an example of Dynamic programming.

ALGORITHM:

```
//Input: Adjacency matrix of digraph
//Output: R, transitive closure of digraph
Accept no. of vertices
Call graph function to read directed graph
Set R[ ] <- digraph matrix // get R {r(i,j)} for k=0
Print digraph
Repeat for k = 1 to n
    Repeat for i = 1 to n
        Repeat for j = 1 to n
            R(i,j) = 1 if
                {  $r_{ij}^{(k-1)} = 1$  OR
                   $r_{ik}^{(k-1)} = 1$  and  $r_{kj}^{(k-1)} = 1$  }
            Print R
```

PROGRAM CODE:

```
#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:");
```

DAA LAB (BCSL404)

```
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

PROGRAM: 4

AIM: 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.

DEFINITION: Dijkstra's algorithm, for a given source vertex(node) in the graph, the algorithm finds the path with lowest cost between that vertex and every other vertex. It can also be used for finding cost of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined.

Efficiency: 1) $\theta(|V|^2)$ -graph represented by weighted matrix and priority queue as unordered array

2) $O(|E| \log |V|)$ -graph represented by adjacency lists and priority queue as min-heap.

ALGORITHM:

Dijkstra(G,s)

//Dijkstra's algorithm for single source shortest path

//input: A weighted connected graph with non-negative weights and its vertex s

//output: The length d_v of a shortest path from s to v and penultimate vertex p_v for every vertex v in V

Initialize(Q)

for every vertex v in V do

$d_v \leftarrow -\infty$; $p_v \leftarrow \text{null}$

Insert(Q,v, d_v)

$D_s \leftarrow 0$; Decrease(Q,s, d_s); $V_T \leftarrow \emptyset$

for i \leftarrow 0 to $|V| - 1$ do

$u^* \leftarrow \text{DeleteMin}(Q)$

$V_T \leftarrow V_T \cup \{u^*\}$

For every vertex u in $V - V_T$ that is adjacent to u^* do

If $d_{u^*} + w(u^*, u) < d_u$

$d_u \leftarrow d_{u^*} + w(u^*, u)$; $p_u \leftarrow u^*$

Decrease(Q,u, d_u)

PROGRAM CODE:

```
#include<stdio.h>
```

```
#define INF 999
```

```
void dijkstra(int c[10][10],int n,int s,int d[10])
```

```
{
```

DAA LAB (BCSL404)

```
int v[10],min,u,i,j;
for(i=1;i<=n;i++)
{
d[i]=c[s][i];
v[i]=0;
}
v[s]=1;
for(i=1;i<=n;i++)
{
min=INF;
for(j=1;j<=n;j++)
if(v[j]==0 && d[j]<min)
{
min=d[j];
u=j;
}
v[u]=1;
for(j=1;j<=n;j++)
if(v[j]==0 && (d[u]+c[u][j])<d[j])
d[j]=d[u]+c[u][j];
}
}
```

```
int main()
{
int c[10][10],d[10],i,j,s,sum,n;
printf("\nEnter 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",&c[i][j]);
printf("\nEnter the souce node:");
scanf("%d",&s);
dijkstra(c,n,s,d);
for(i=1;i<=n;i++)
printf("\nShortest distance from %d to %d is %d",s,i,d[i]);
return 0;
}
```

OUTPUT:

Enter n value:6

DAA LAB (BCSL404)

Enter the graph data:

0 15 10 999 45 999

999 0 15 999 20 999

20 999 0 20 999 999

999 10 999 0 35 999

999 999 999 30 0 999

999 999 999 4 999 0

Enter the source node:2

Shortest distance from 2 to 1 is 35

Shortest distance from 2 to 2 is 0

Shortest distance from 2 to 3 is 15

Shortest distance from 2 to 4 is 35

Shortest distance from 2 to 5 is 20

Shortest distance from 2 to 6 is 999

PROGRAM: 5

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

DEFINITION: Topological ordering that for every edge in the graph, the vertex where the edge starts is listed before the edge where the edge ends.

ALGORITHM:

1. repeatedly identify in a remaining digraph a source which is a vertex with no incoming edges and delete it along with all edges outgoing from it
2. The order in which the vertices are deleted yields a solution to the topological sorting.

PROGRAM CODE:

```
#include<stdio.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

{
```

DAA LAB (BCSL404)

```
printf("\nTopological ordering is:");  
  
for(i=1;i<=k;i++)  
  
printf("%d ",temp[i]);  
  
}  
  
}
```

OUTPUT:

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

PROGRAM: 6

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

DEFINITION: It gives us a way to design custom algorithms which systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency). We are given a set of n items from which we are to select some number of items to be carried in a knapsack (BAG).

Each item has both a weight and a profit. The objective is to choose the set of items that fits in the knapsack and maximizes the profit.

Given a knapsack with maximum capacity W , and a set S consisting of n items, Each item i has some weight w_i and benefit value b_i (all w_i , b_i and W are integer values).

ALGORITHM:

```
//(n items, W weight of sack) Input: n,  $w_i$ ,  $v_i$  and  $W$  – all integers
//Output:  $V(n,W)$ 
// Initialization of first column and first row elements
Repeat for  $i = 0$  to  $n$ 
    set  $V(i,0) = 0$ 
Repeat for  $j = 0$  to  $W$ 
    Set  $V(0,j) = 0$ 
//complete remaining entries row by row
Repeat for  $i = 1$  to  $n$ 
    repeat for  $j = 1$  to  $W$ 
        if (  $w_i \leq j$  )  $V(i,j) = \max \{ V(i-1,j), V(i-1,j-w_i) + v_i \}$ 
        if (  $w_i > j$  )  $V(i,j) = V(i-1,j)$ 
Print  $V(n,W)$ 
```

PROGRAM CODE:

```
#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:6

Enter profit followed by weight:

78 2

45 3

92 4

71 5

Max profit=170

PROGRAM: 7

AIM: Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

DEFINITION: This program first calculates the profit-to-weight ratio for each item, then sorts the items based on this ratio in non-increasing order. It then fills the knapsack greedily by selecting items with the highest ratio until the knapsack is full.

If there's space left in the knapsack after selecting whole items, it adds fractional parts of the next item. Finally, it prints the optimal solution and the solution vector. Here's a simplified version of the C program to solve discrete Knapsack and continuous Knapsack problems using the greedy approximation method.

PROGRAM CODE:

```
#include <stdio.h>
#define MAX 50
int p[MAX], w[MAX], x[MAX];
double maxprofit;
int n, m, i;
void greedyKnapsack(int n, int w[], int p[], int m) {
    double ratio[MAX];
    // Calculate the ratio of profit to weight for each item
    for (i = 0; i < n; i++) {
        ratio[i] = (double)p[i] / w[i];
    }
    // Sort items based on the ratio in non-increasing order
    for (i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (ratio[i] < ratio[j]) {
                double temp = ratio[i];
                ratio[i] = ratio[j];
                ratio[j] = temp;

                int temp2 = w[i];
                w[i] = w[j];
                w[j] = temp2;

                temp2 = p[i];
                p[i] = p[j];
                p[j] = temp2;
            }
        }
    }
}
```

```
    }
}

int currentWeight = 0;
maxprofit = 0.0;

// Fill the knapsack with items
for (i = 0; i < n; i++) {
    if (currentWeight + w[i] <= m) {
        x[i] = 1; // Item i is selected
        currentWeight += w[i];
        maxprofit += p[i];
    } else {
        // Fractional part of item i is selected
        x[i] = (m - currentWeight) / (double)w[i];
        maxprofit += x[i] * p[i];
        break;
    }
}

printf("Optimal solution for greedy method: %.1f\n", maxprofit);
printf("Solution vector for greedy method: ");
for (i = 0; i < n; i++)
    printf("%d\t", x[i]);
}

int main() {
    printf("Enter the number of objects: ");
    scanf("%d", &n);

    printf("Enter the objects' weights: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);

    printf("Enter the objects' profits: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);

    printf("Enter the maximum capacity: ");
    scanf("%d", &m);

    greedyKnapsack(n, w, p, m);
```

```
    return 0;  
}
```

OUTPUT:

Enter the number of objects: 4
Enter the objects' weights: 2 1 3 2
Enter the objects' profits: 12 10 20 15
Enter the maximum capacity: 5
Optimal solution for greedy method: 25.0
Solution vector for greedy method: 1 1 0 0

PROGRAM: 8

AIM: Design and implement C/C++ Program to find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d .

DEFINITION: An instance of the Subset Sum problem is a pair (S, t) , where $S = \{x_1, x_2, \dots, x_n\}$ is a set of positive integers and t (the target) is a positive integer. The decision problem asks for a subset of S whose sum is as large as possible, but not larger than t .

ALGORITHM:

SumOfSub (s, k, r)

//Values of $x[j]$, $1 \leq j < k$, have been determined

//Node creation at level k taking place: also call for creation at level $K+1$ if possible

// s = sum of 1 to $k-1$ elements and r is sum of k to n elements

//generating left child that means including k in solution

Set $x[k] = 1$

If $(s + s[k] = d)$ then subset found, print solution

If $(s + s[k] + s[k+1] \leq d)$

 then SumOfSum ($s + s[k], k+1, r - s[k]$)

//Generate right child i.e. element k absent

If $(s + r - s[k] \geq d)$ AND $(s + s[k+1]) \leq d$

THEN { $x[k]=0$;

 SumOfSub($s, k+1, r - s[k]$)

PROGRAM CODE:

```
#include<stdio.h>
```

```
#define MAX 10
```

```
int s[MAX],x[MAX],d;
```

```
void sumofsub(int p,int k,int r)
```

```
{
```

```
int i;
```

```
x[k]=1;
```

```
if((p+s[k])==d)
{
for(i=1;i<=k;i++)
if(x[i]==1)
printf("%d ",s[i]);
printf("\n");
}
else
if(p+s[k]+s[k+1]<=d)
sumofsub(p+s[k],k+1,r-s[k]);
if((p+r-s[k]>=d) && (p+s[k+1]<=d))
{
x[k]=0;
sumofsub(p,k+1,r-s[k]);
}
}

int main()
{
int i,n,sum=0;
printf("\nEnter the n value:");
scanf("%d",&n);
printf("\nEnter the set in increasing order:");
for(i=1;i<=n;i++)
scanf("%d",&s[i]);
printf("\nEnter the max subset value:");
scanf("%d",&d);
for(i=1;i<=n;i++)
```

DAA LAB (BCSL404)

```
sum=sum+s[i];  
if(sum<d || s[1]>d)  
printf("\nNo subset possible");  
else  
sumofsub(0,1,sum);  
return 0;  
}
```

OUTPUT:

Enter the n value:9

Enter the set in increasing order:1 2 3 4 5 6 7 8 9

Enter the max subset value:9

1 2 6

1 3 5

1 8

2 3 4

2 7

3 6

4 5

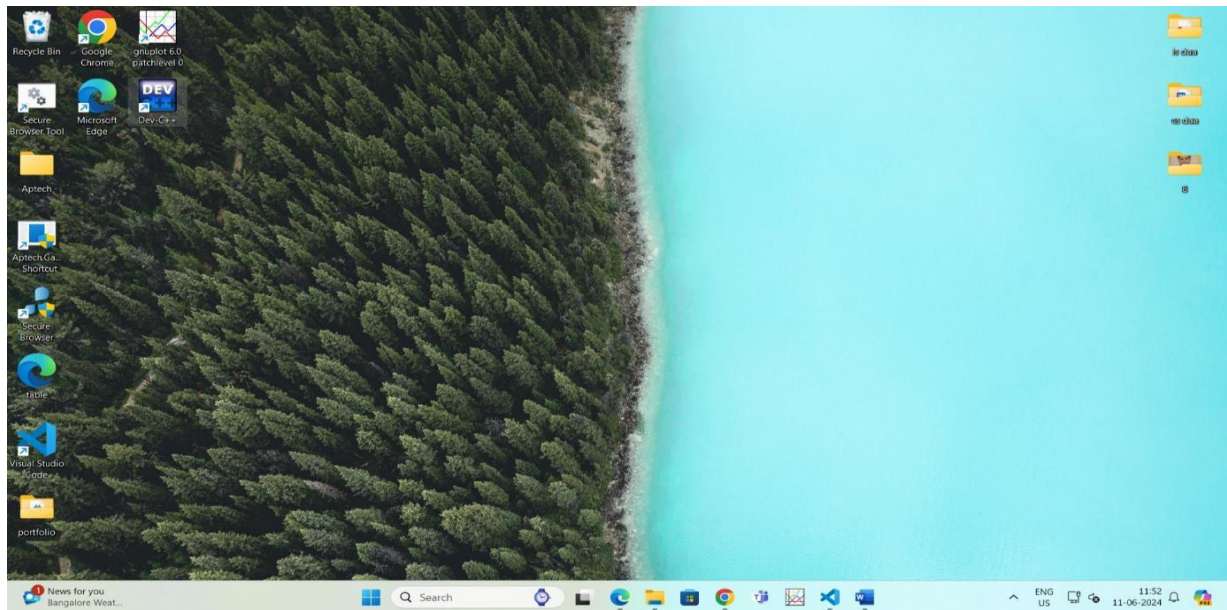
9

Steps for Program 9, 10 and 11:

Plotting graph is required for these programs, we require DEV C/C++ in Windows operating system for the execution of program, data generation and plotting graph.

Step 1: Initial Step for Project Creation using DEV C/C++.

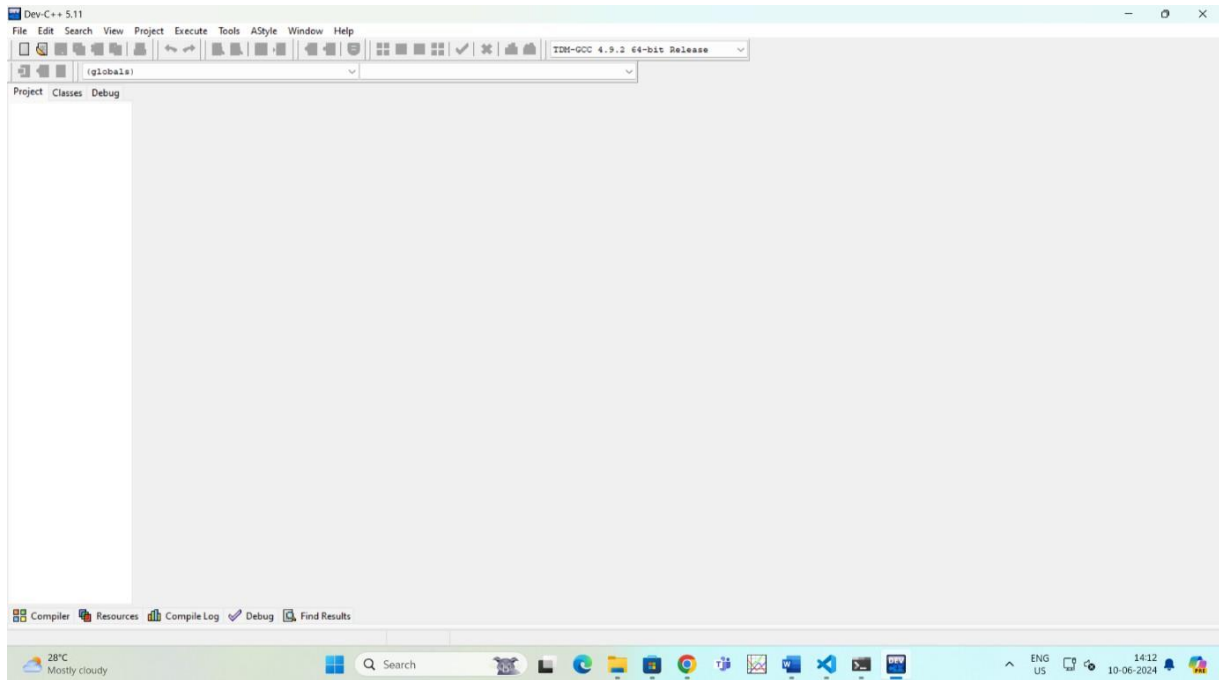
Double click on DEV C/C++. The editor will appear on the display as shown in snapshot 2.2



Snapshot 2.1

Dev-C/C++ is an **Integrated Development Environment (IDE)** for the **C** and **C++** programming languages. It is designed to provide developers with a complete set of tools to write, compile, and debug their C/C++ programs. Here is a small explanation covering its key features and functionality. We are using C language. It is user friendly, open source and has less complexities.

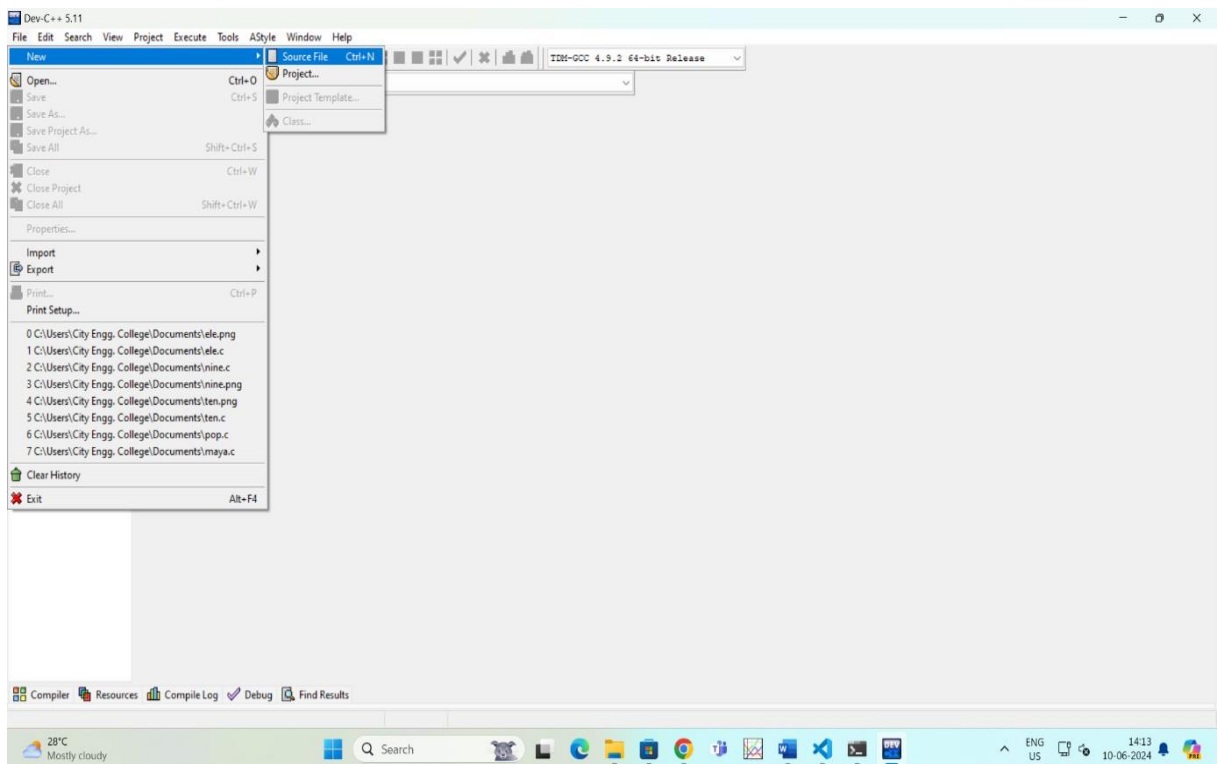
C is a general-purpose programming language created by **Dennis Ritchie** at the Bell Laboratories in 1972. It is a very popular language, despite being old. The main reason for its popularity is because it is a fundamental language in the field of computer science. C is strongly associated with UNIX, as it was developed to write the UNIX operating system.



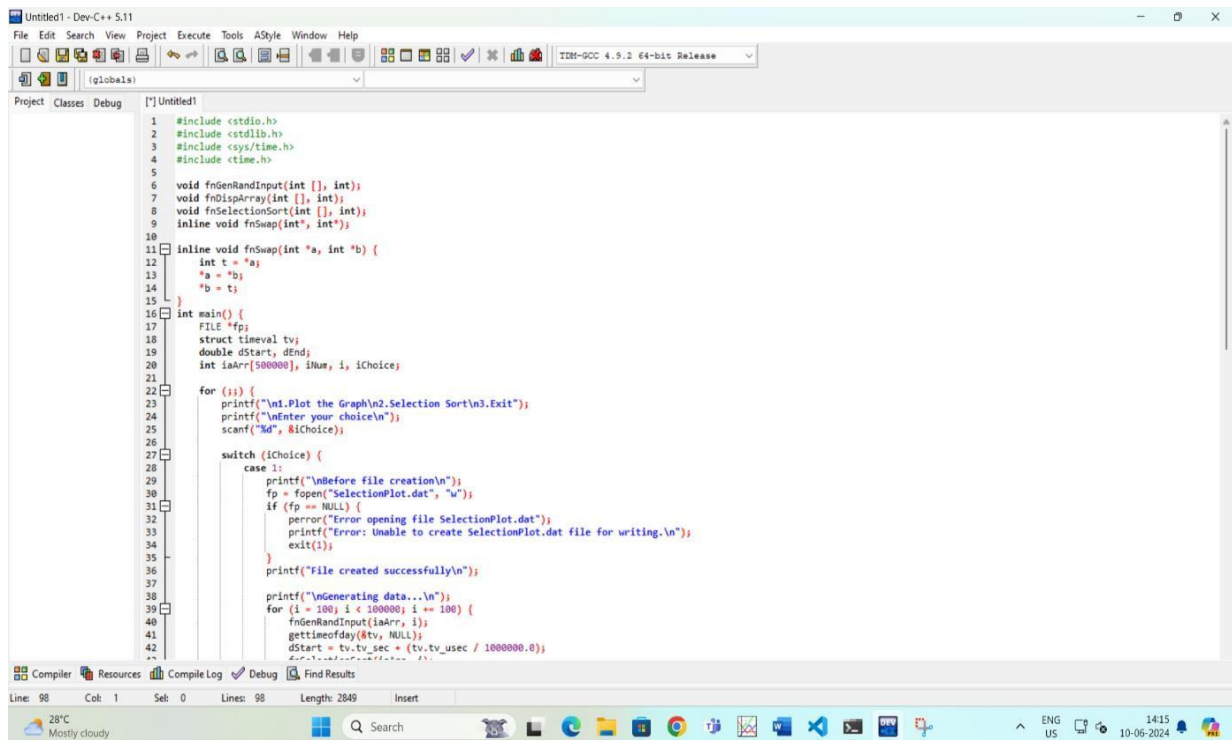
Snapshot 2.2

Step 2: Creation of Project

Click on File -> New-> Source file -> Editor will appear where we execute the program.



Snapshot 2.3

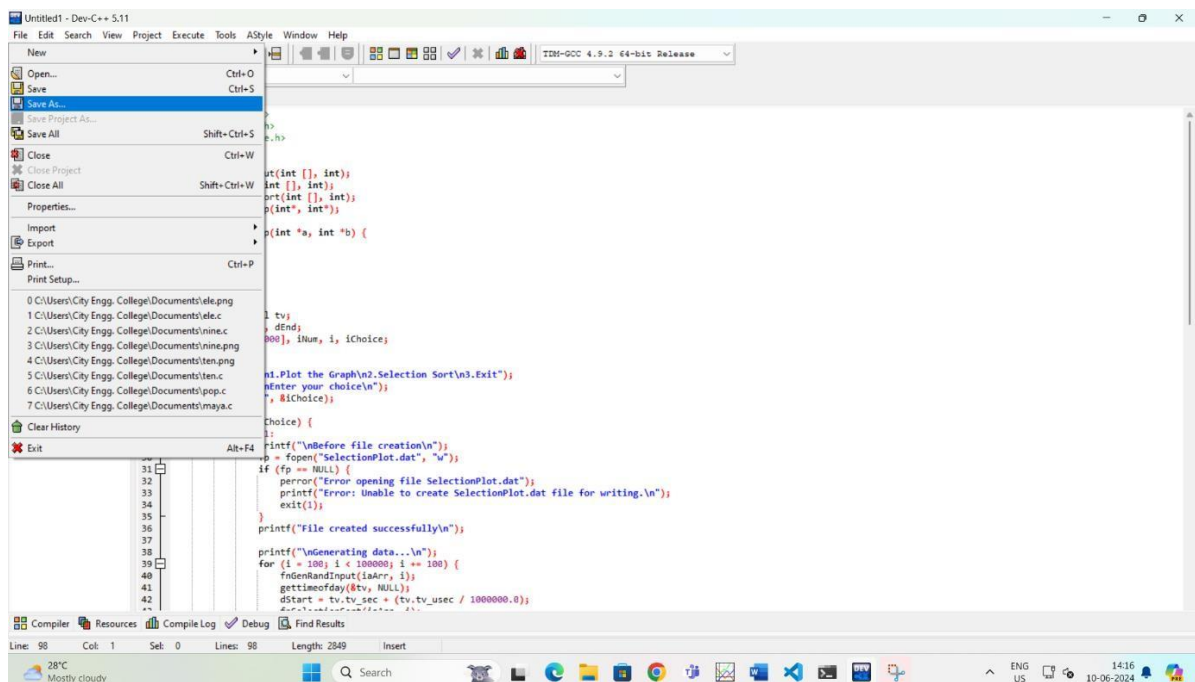


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/time.h>
4 #include <time.h>
5
6 void fnGenRandInput(int [], int);
7 void fnDisplayArray(int [], int);
8 void fnSelectionSort(int [], int);
9 inline void fnSwap(int*, int*);
10
11 inline void fnSwap(int *a, int *b) {
12     int t = *a;
13     *a = *b;
14     *b = t;
15 }
16
17 int main() {
18     FILE *fp;
19     struct timeval tv;
20     double dStart, dEnd;
21     int iaArr[500000], iNum, i, iChoice;
22
23     for (i = 1; i < 100; i++) {
24         printf("\n1.Plot the Graph\n2.Selection Sort\n3.Exit");
25         printf("\nEnter your choice\n");
26         scanf("%d", &iChoice);
27
28         switch (iChoice) {
29             case 1:
30                 printf("\nBefore file creation\n");
31                 fp = fopen("SelectionPlot.dat", "w");
32                 if (fp == NULL) {
33                     perror("Error opening file SelectionPlot.dat");
34                     printf("Error: Unable to create SelectionPlot.dat file for writing.\n");
35                     exit(1);
36                 }
37                 printf("File created successfully\n");
38
39                 printf("\nGenerating data...\n");
40                 for (i = 100; i < 100000; i++) {
41                     fnGenRandInput(iaArr, i);
42                     gettimeofday(&tv, NULL);
43                     dStart = tv.tv_sec + (tv.tv_usec / 1000000.0);
44                 }
45             case 2:
46                 fnSelectionSort(iaArr, iNum);
47                 fnDisplayArray(iaArr, iNum);
48             case 3:
49                 return 0;
50         }
51     }
52 }
```

Snapshot 2.4

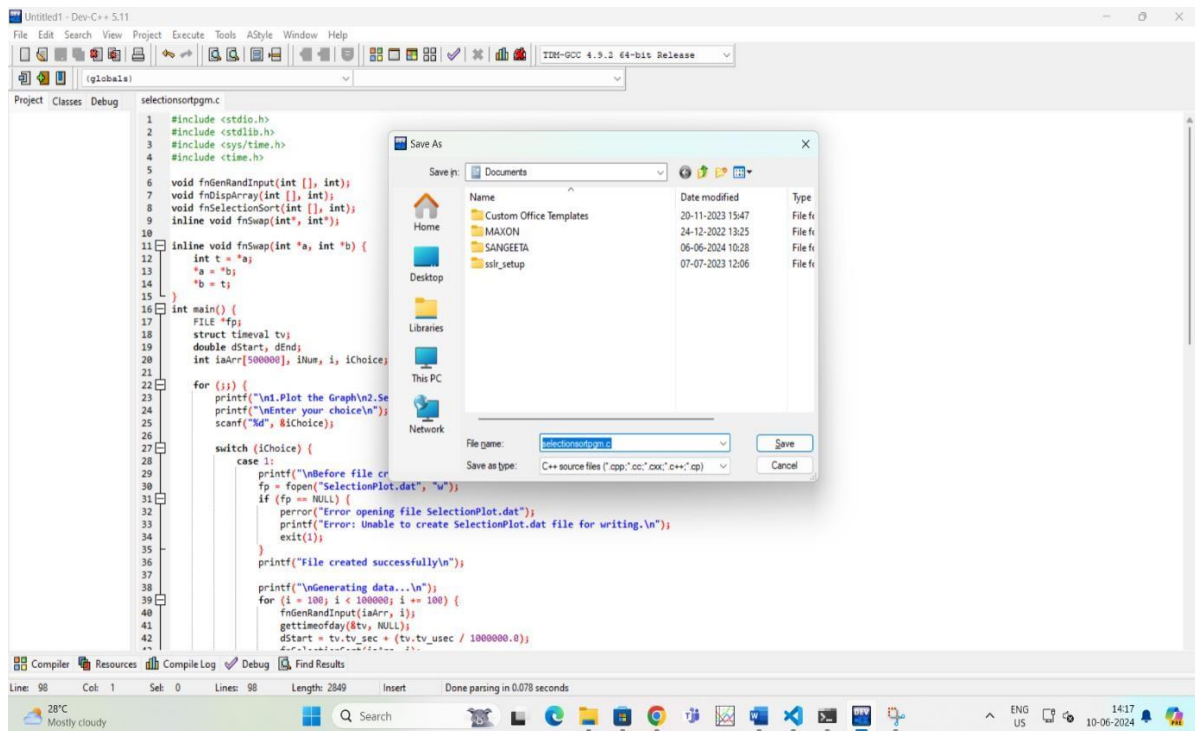
Step 3: Save the project

The program code has to be saved with **.c extension**. Click on File -> Save As -> filename.c (the file name has to be the name given to the project). (Refer snapshots 2.5 and 2.6)



```
File Edit Search View Project Execute Tools AStyle Window Help
New
Open... Ctrl+O
Save Ctrl+S
Save As...
Save Project As...
Save All Shift+Ctrl+S
Close Ctrl+W
Close Project
Close All Shift+Ctrl+W
Properties...
Import
Export
Print... Ctrl+P
Print Setup...
0 C:\Users\City Engg. College\Documents\ele.png
1 C:\Users\City Engg. College\Documents\ele.c
2 C:\Users\City Engg. College\Documents\nine.c
3 C:\Users\City Engg. College\Documents\nine.png
4 C:\Users\City Engg. College\Documents\ten.c
5 C:\Users\City Engg. College\Documents\ten.png
6 C:\Users\City Engg. College\Documents\pop.c
7 C:\Users\City Engg. College\Documents\maya.c
Clear History
Exit Alt+F4
```

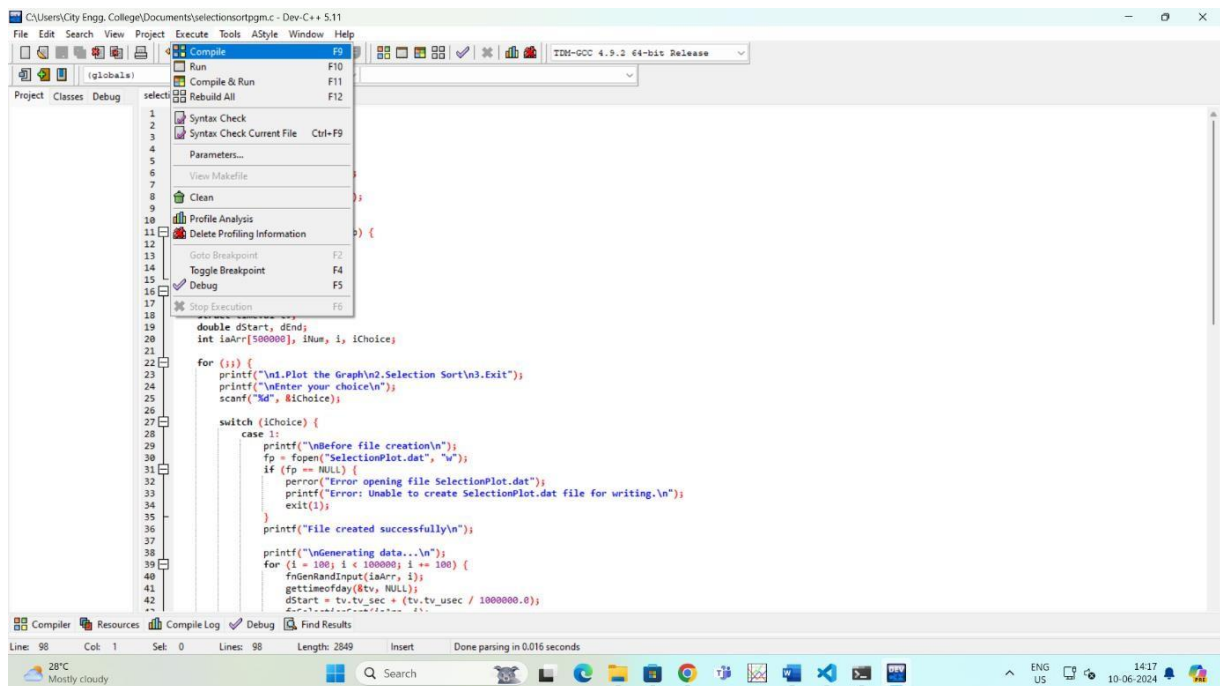
Snapshot 2.5



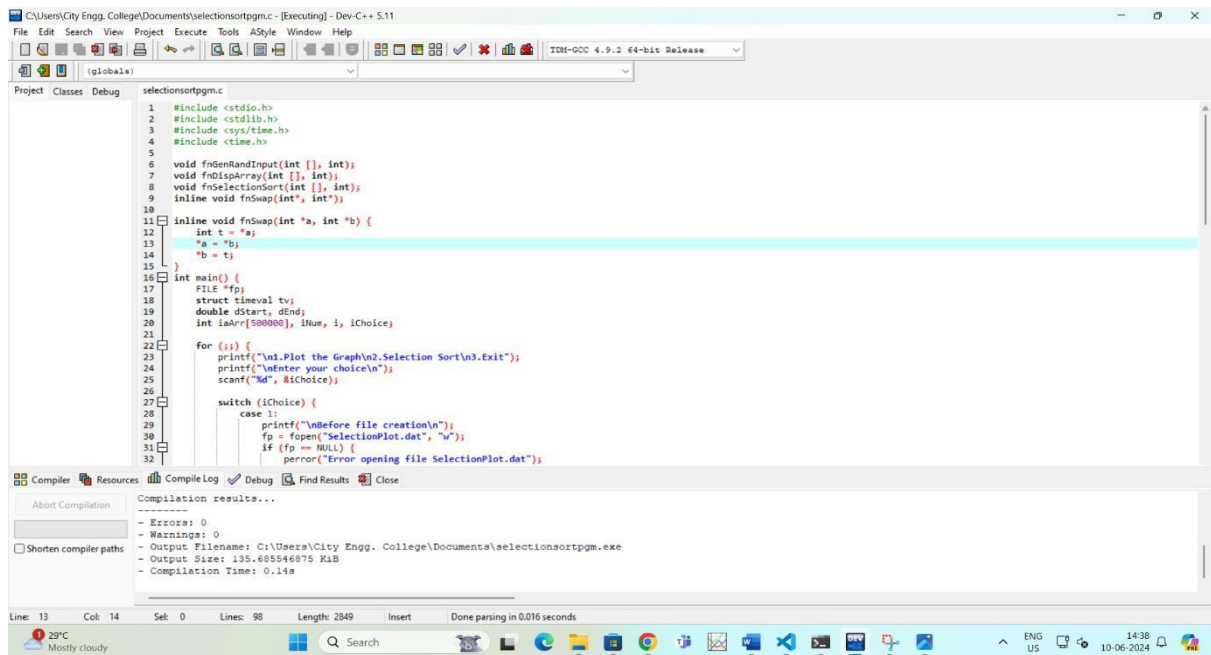
Snapshot 2.6

Step 4: Debug and Execution of the program

Click on Execute -> Compile/F9. This checks for the errors and warnings in the program. If there are no errors in the code then after Compile, we can Run the program or just click F10 for the output. (Refer snapshots 2.7 and 2.8)



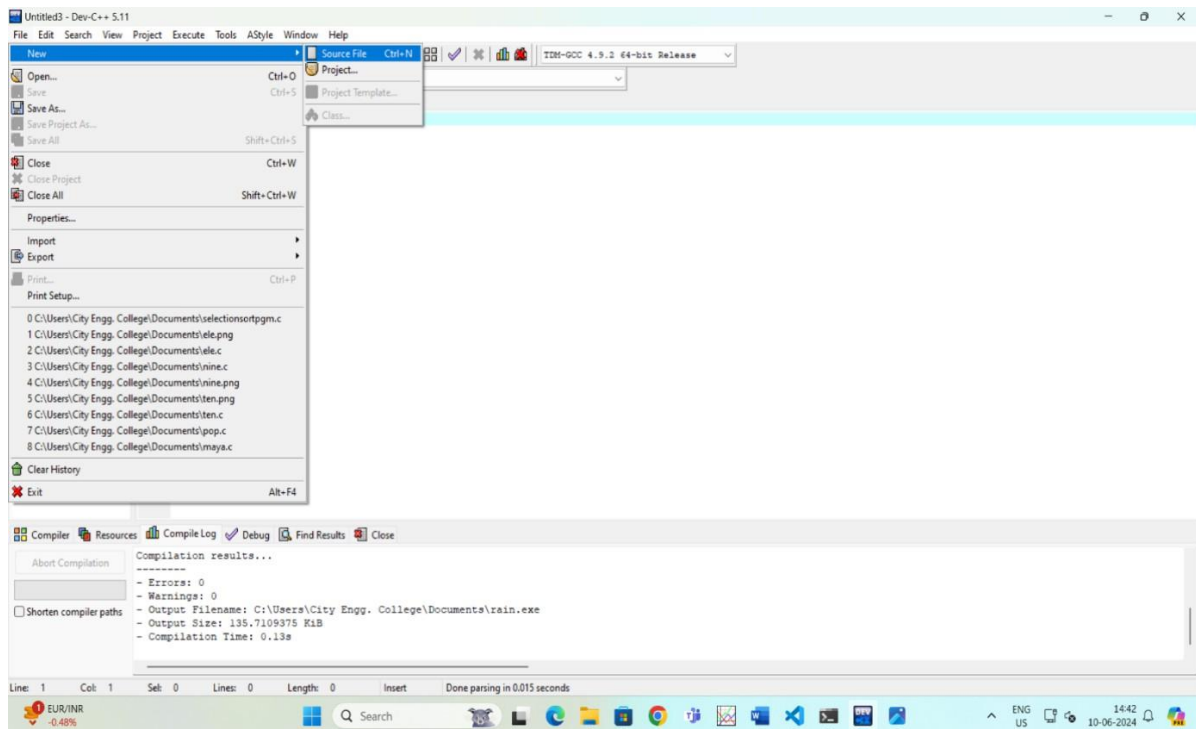
Snapshot 2.7



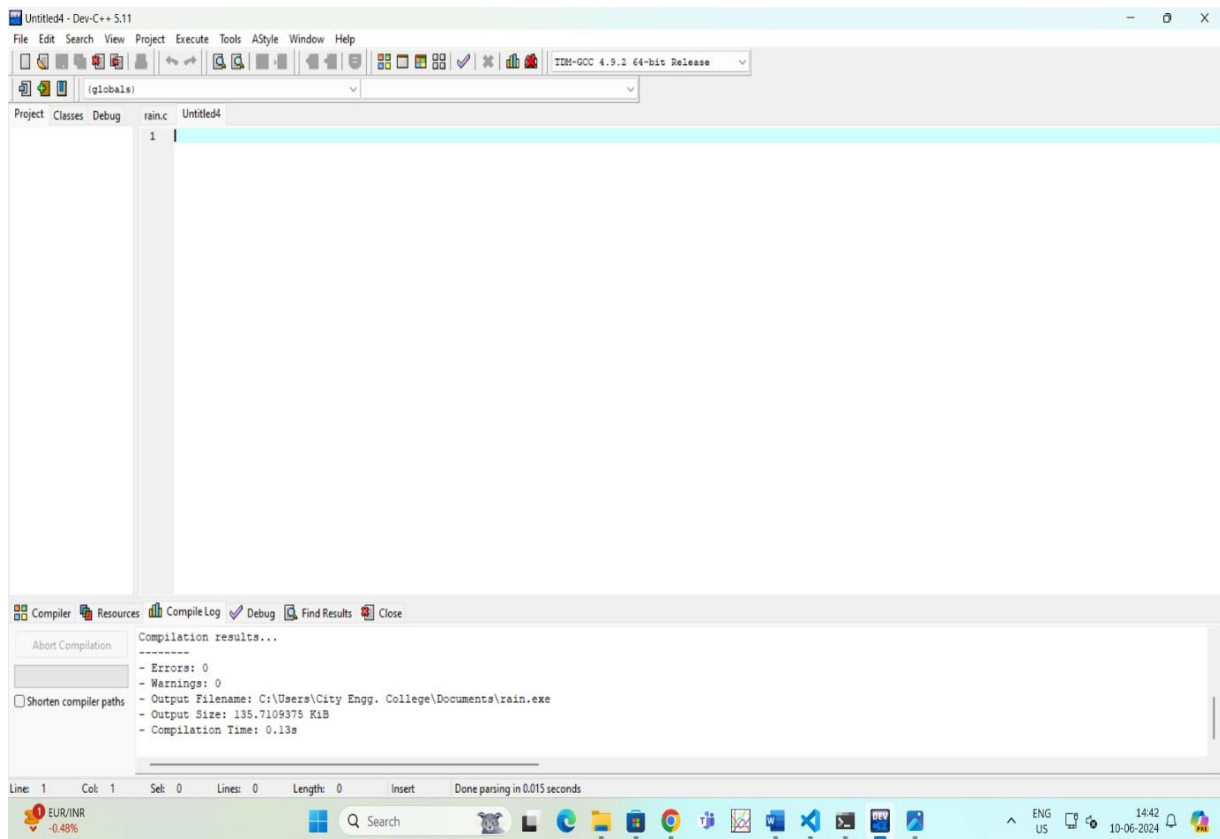
Snapshot 2.8

Step 5: Creation of PNG file.

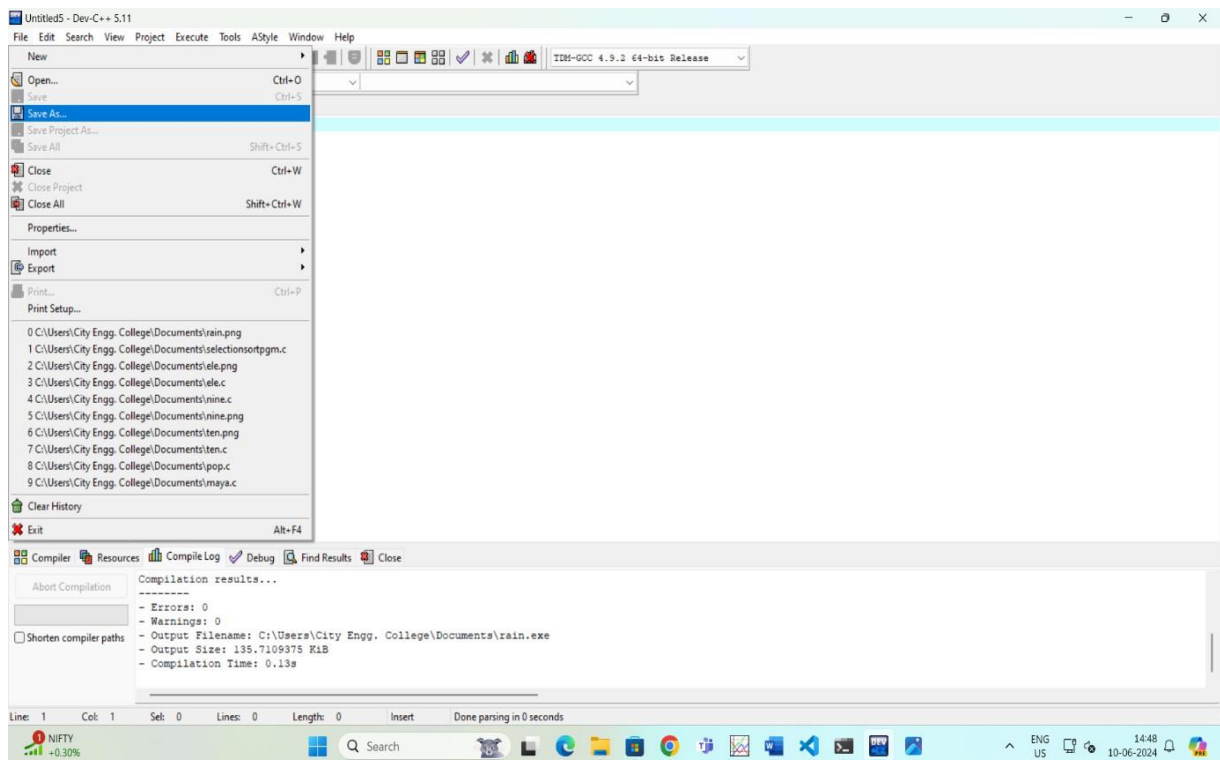
The graph plotted is displayed as a **PNG image**, hence we need a PNG file. Click on File -> New -> Source File -> Now save this blank/untitled file using “.png” extension by Clicking File -> Save As -> filename.png (Refer Snapshot 2.0 to 2.12)



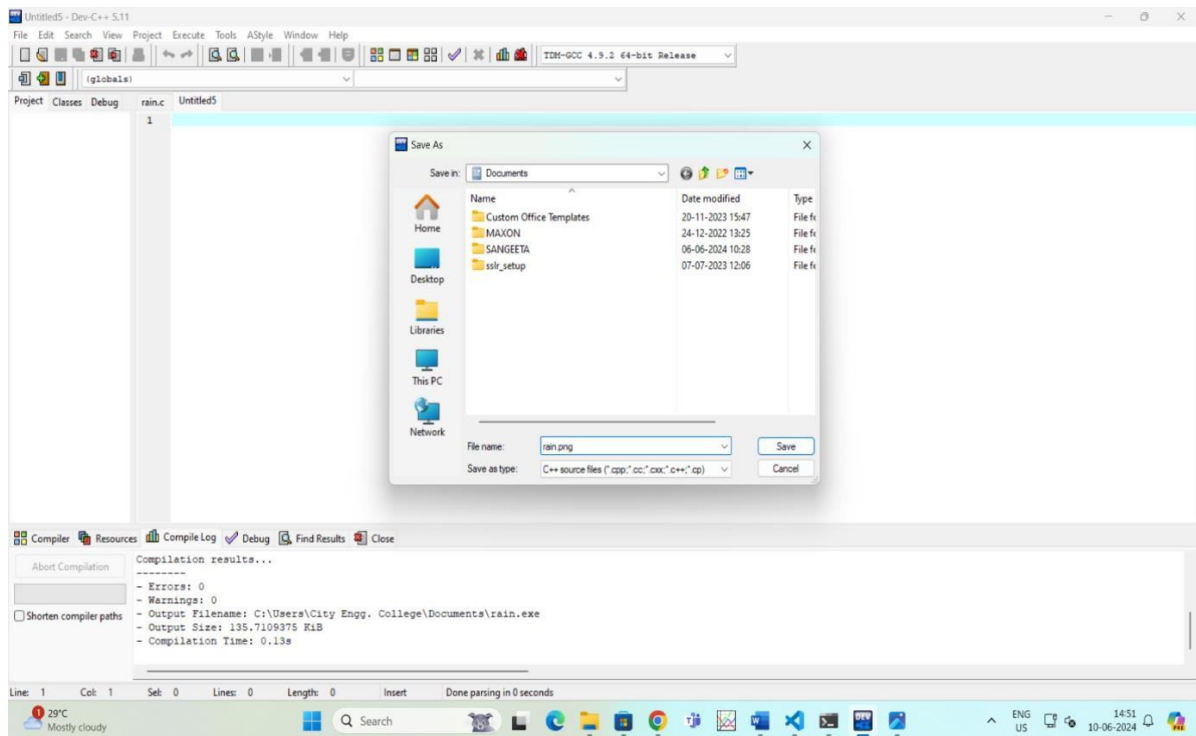
Snapshot 2.9



Snapshot 2.10



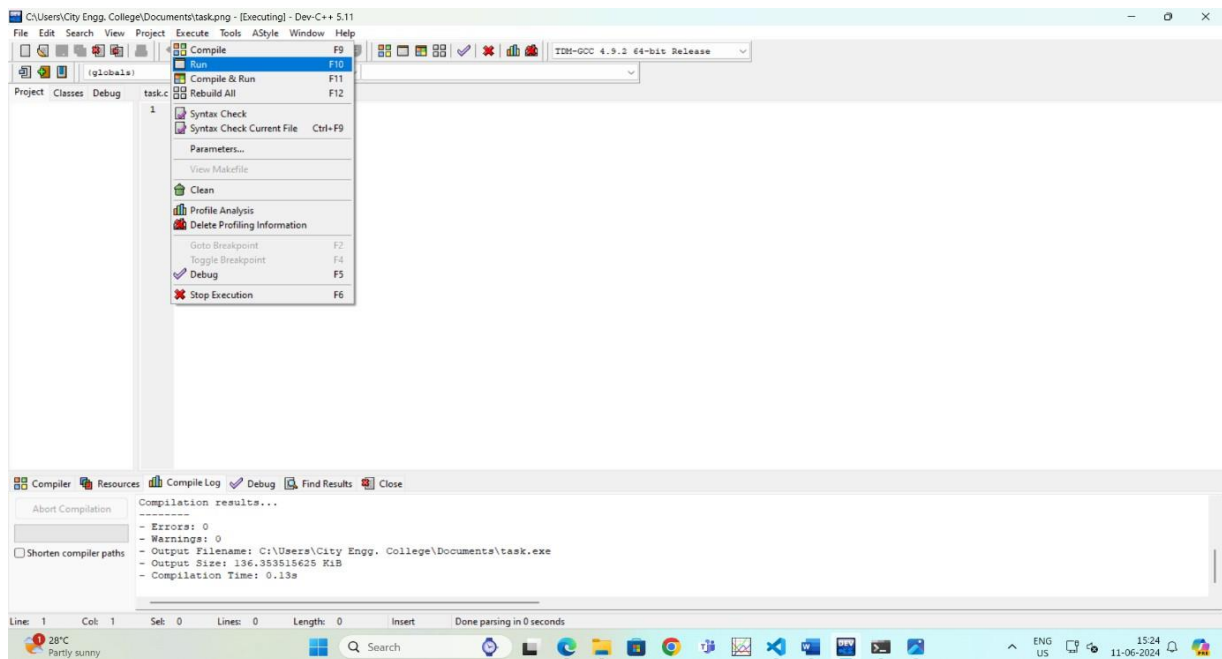
Snapshot 2.11



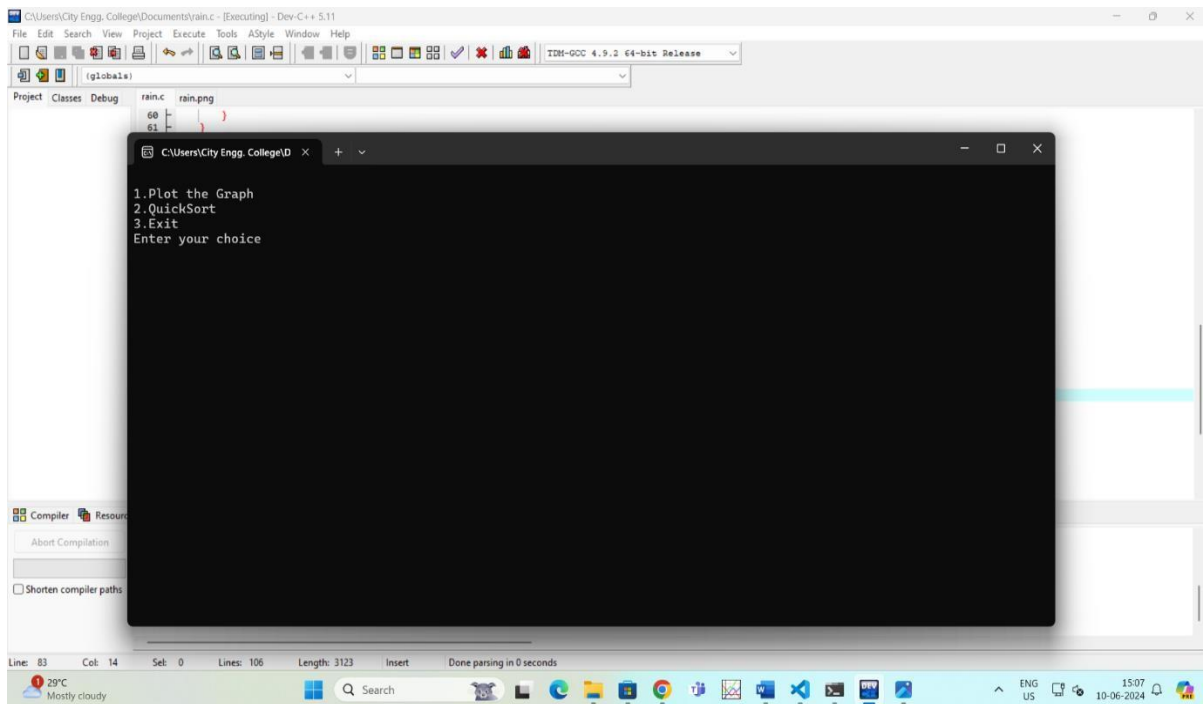
Snapshot 2.12

Step 6: Execution of data file

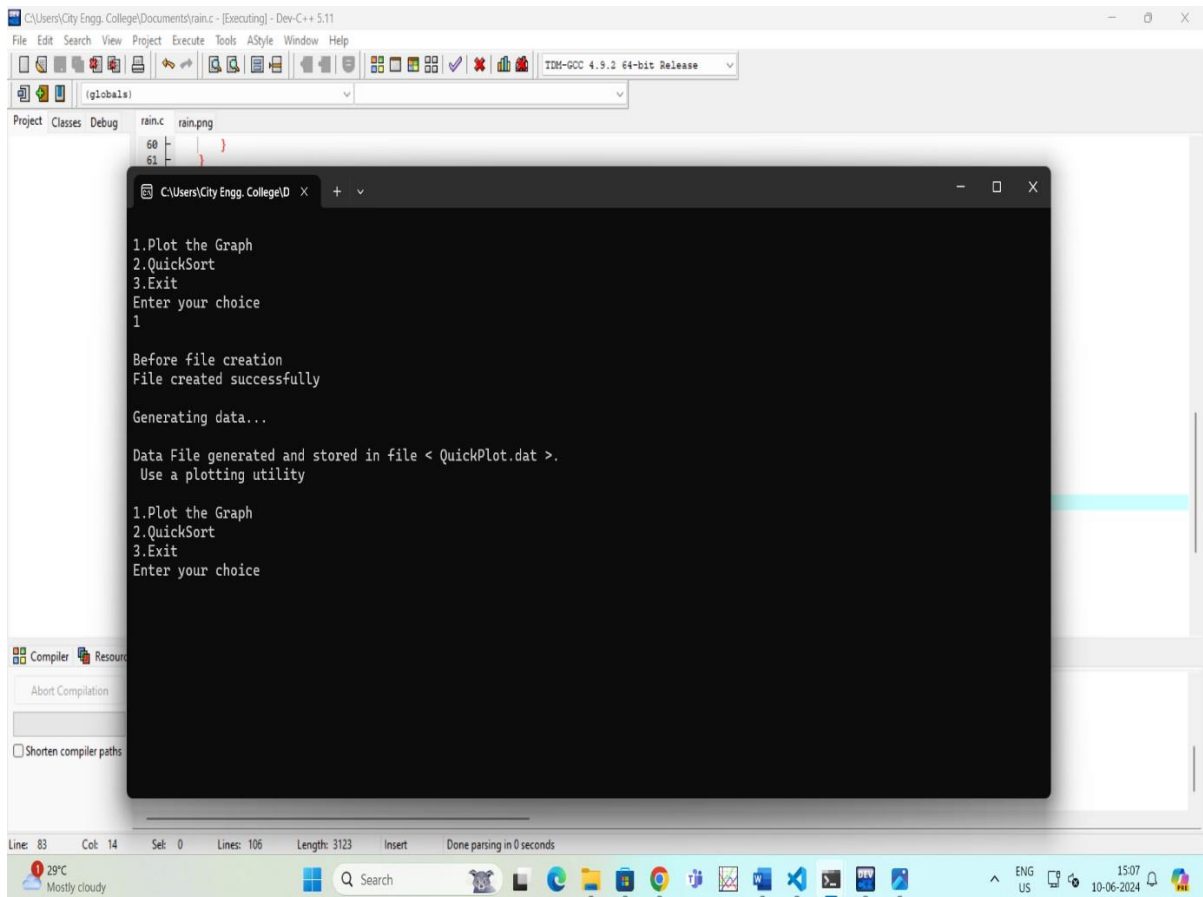
Click on Execute -> select Run option/F10, a console window will appear where we output is executed.



Snapshot 2.14



Snapshot 2.15

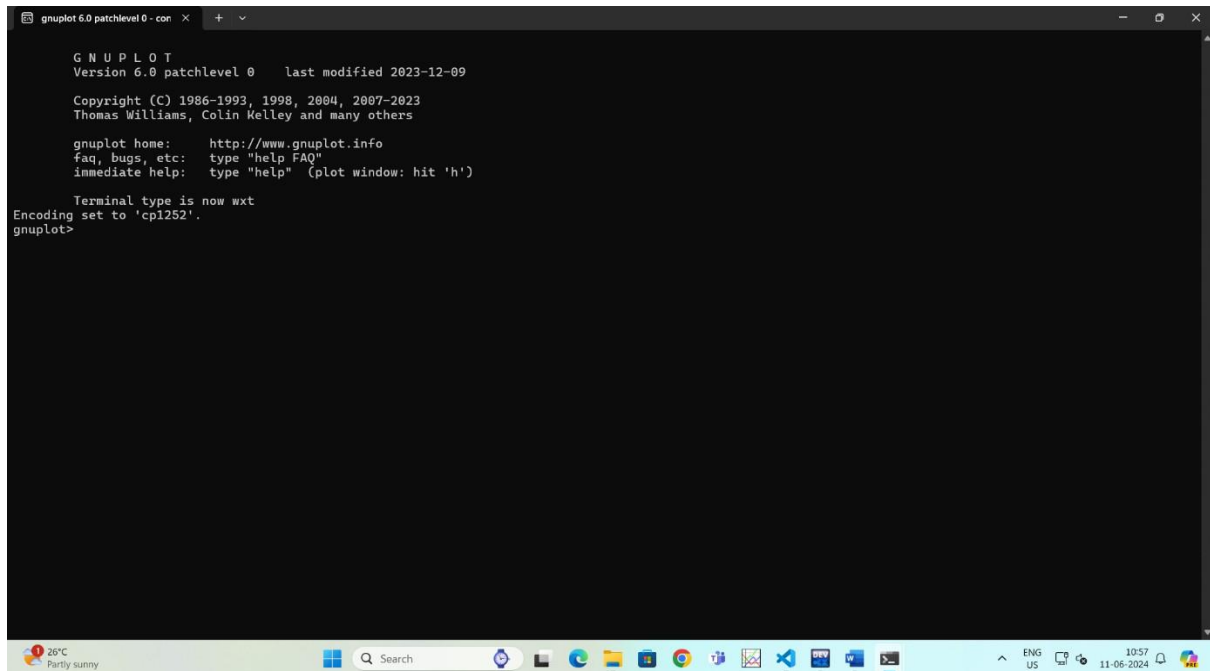


Snapshot 2.16

Step 7: Using GNU Plot

GNU Plot: gnuplot is a command-line and GUI program that can generate two- and three-dimensional plots of functions, data, and data fits. The program runs on all major computers and operating systems (Linux, Unix, Microsoft Windows, macOS, FreeDOS, and many others).

Double click on **GNU plot**, a console will appear where certain commands has to be given for plotting graph.

A screenshot of a terminal window titled 'gnuplot 6.0 patchlevel 0 - con'. The terminal displays the following text:

```
GNU PLOT
Version 6.0 patchlevel 0    last modified 2023-12-09

Copyright (C) 1986-1993, 1998, 2004, 2007-2023
Thomas Williams, Colin Kelley and many others

gnuplot home:      http://www.gnuplot.info
faq, bugs, etc.:   type "help FAQ"
immediate help:    type "help" (plot window: hit 'h')

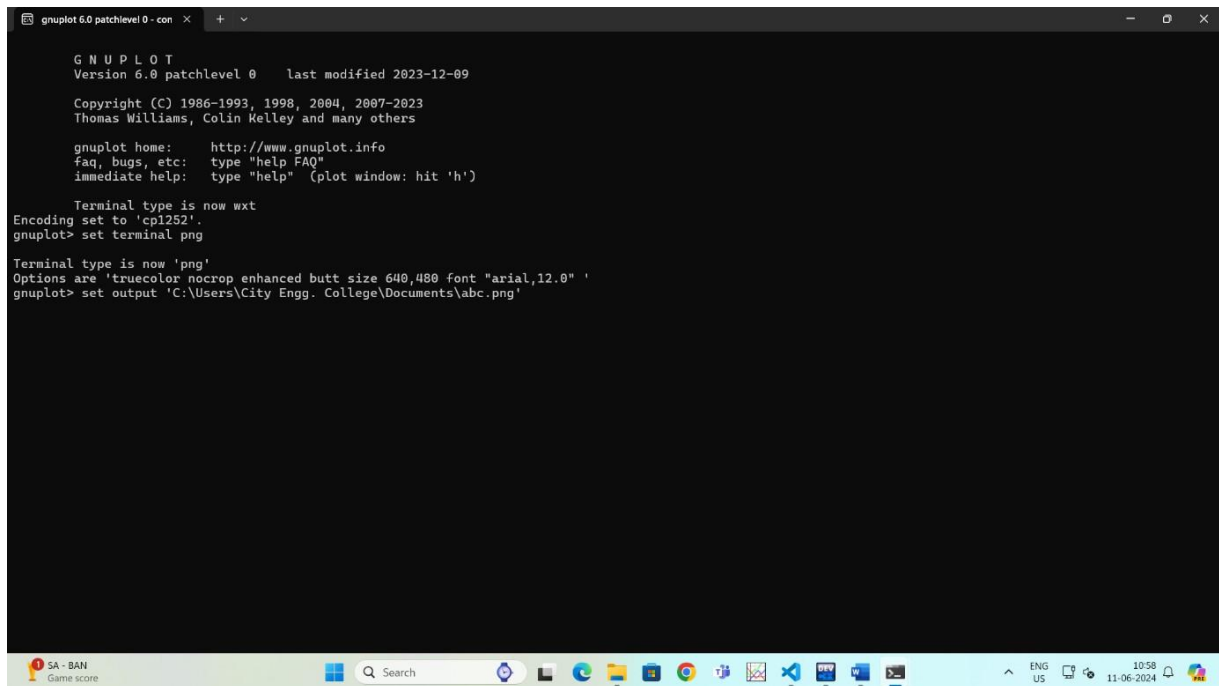
Terminal type is now wxt
Encoding set to 'cp1252'.
gnuplot>
```

The terminal window is open on a Windows desktop, with the taskbar visible at the bottom showing the date as 11-06-2024 and time as 10:57.

Snapshot 2.17

Step 8: Setting terminal as PNG

Give command **gnuplot > set terminal png**



```
gnuplot 6.0 patchlevel 0 - con
+
-
x

GNU PLOT
Version 6.0 patchlevel 0    last modified 2023-12-09

Copyright (C) 1986-1993, 1998, 2004, 2007-2023
Thomas Williams, Colin Kelley and many others

gnuplot home:      http://www.gnuplot.info
faq, bugs, etc.:   type "help FAQ"
immediate help:    type "help" (plot window: hit 'h')

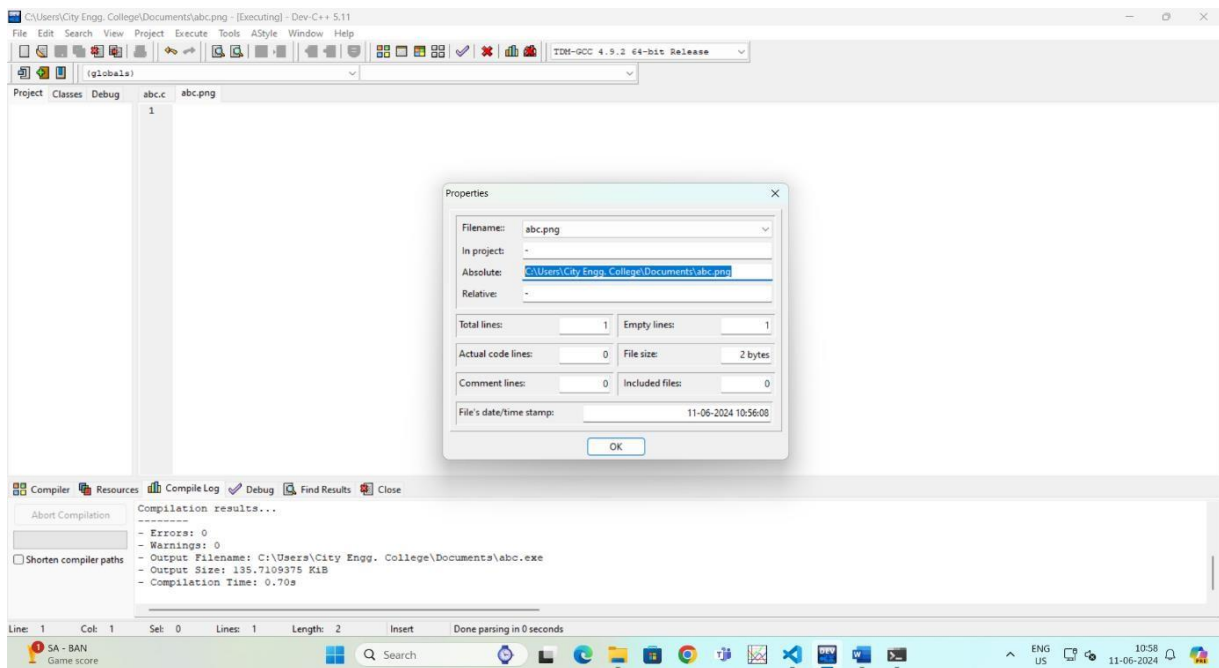
Terminal type is now wxt
Encoding set to 'cp1252'.
gnuplot> set terminal png

Terminal type is now 'png'
Options are 'truecolor nocrop enhanced butt size 640,480 font "arial,12.0" '
gnuplot> set output 'C:\Users\City Engg. College\Documents\abc.png'
```

Snapshot 2.18

Step 9: Setting path for PNG file

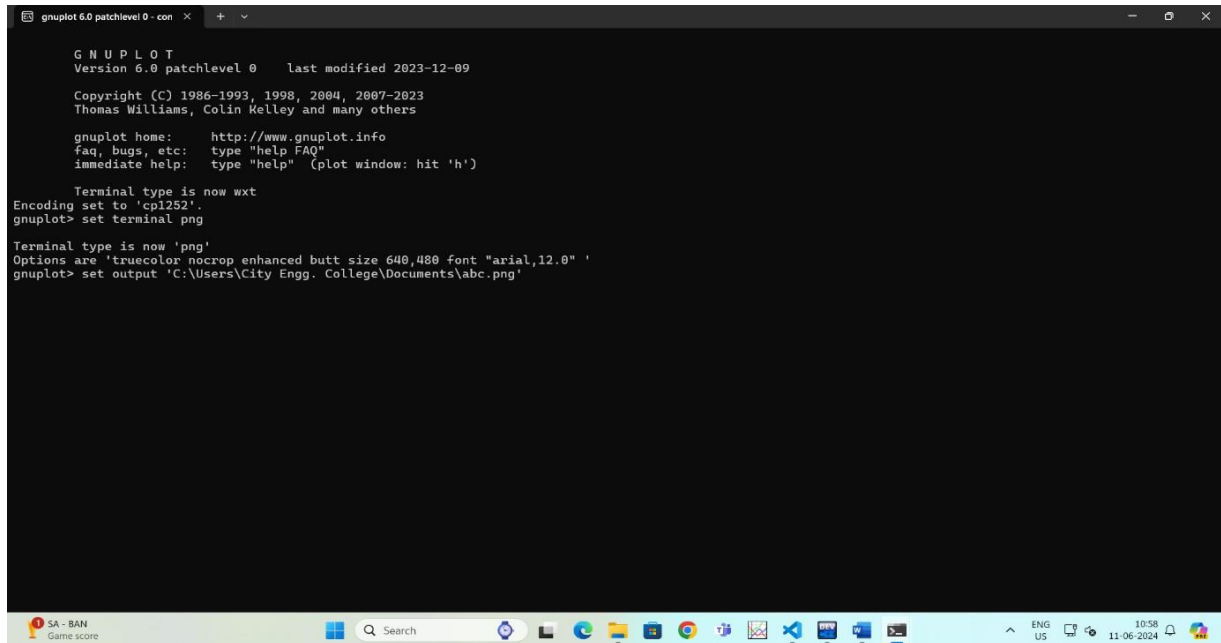
In DEV C/C++, right click on png file, copy the png file location path which is available in absolute section.



Snapshot 2.19

Step 10: Commands using GNU Plot

Give command **gnuplot > set output 'location path of png file\ filename.png'**
(within single quotes)



```
gnuplot 6.0 patchlevel 0 - con
GNU PLOT
Version 6.0 patchlevel 0    Last modified 2023-12-09
Copyright (C) 1986-1993, 1998, 2004, 2007-2023
Thomas Williams, Colin Kelley and many others

gnuplot home:      http://www.gnuplot.info
faq, bugs, etc:   type "help FAQ"
immediate help:   type "help" (plot window: hit 'h')

Terminal type is now wxt
Encoding set to 'cp1252'.
gnuplot> set terminal png

Terminal type is now 'png'
Options are 'truecolor nocrop enhanced butt size 640,480 font "arial,12.0" '
gnuplot> set output 'C:\Users\City Engg. College\Documents\abc.png'
```

Snapshot 2.20

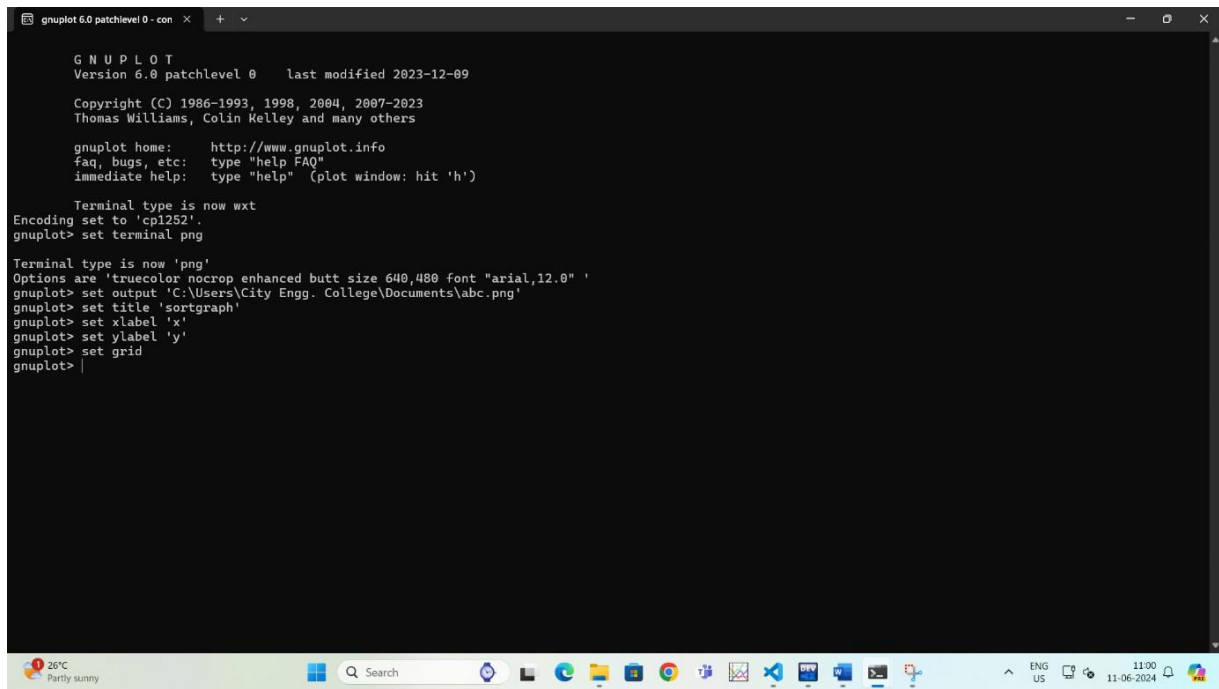
Now give the following commans

gnuplot > set title 'name' (within single quotes give the graph title)

gnuplot > set xlabel 'x' (within single quotes give xlabel)

gnuplot > set ylabel 'y' (within single quotes give ylabel)

gnuplot > set grid



```
gnuplot 6.0 patchlevel 0 - con x + v

G N U P L O T
Version 6.0 patchlevel 0    last modified 2023-12-09

Copyright (C) 1986-1993, 1998, 2004, 2007-2023
Thomas Williams, Colin Kelley and many others

gnuplot home:      http://www.gnuplot.info
faq, bugs, etc:    type "help FAQ"
immediate help:    type "help" (plot window: hit 'h')

Terminal type is now wxt
Encoding set to 'cp1252'.
gnuplot> set terminal png

Terminal type is now 'png'
Options are 'truecolor nocrop enhanced butt size 640,480 font "arial,12.0" '
gnuplot> set output 'C:\Users\City Engg. College\Documents\abc.png'
gnuplot> set title 'sortgraph'
gnuplot> set xlabel 'x'
gnuplot> set ylabel 'y'
gnuplot> set grid
gnuplot> |
```

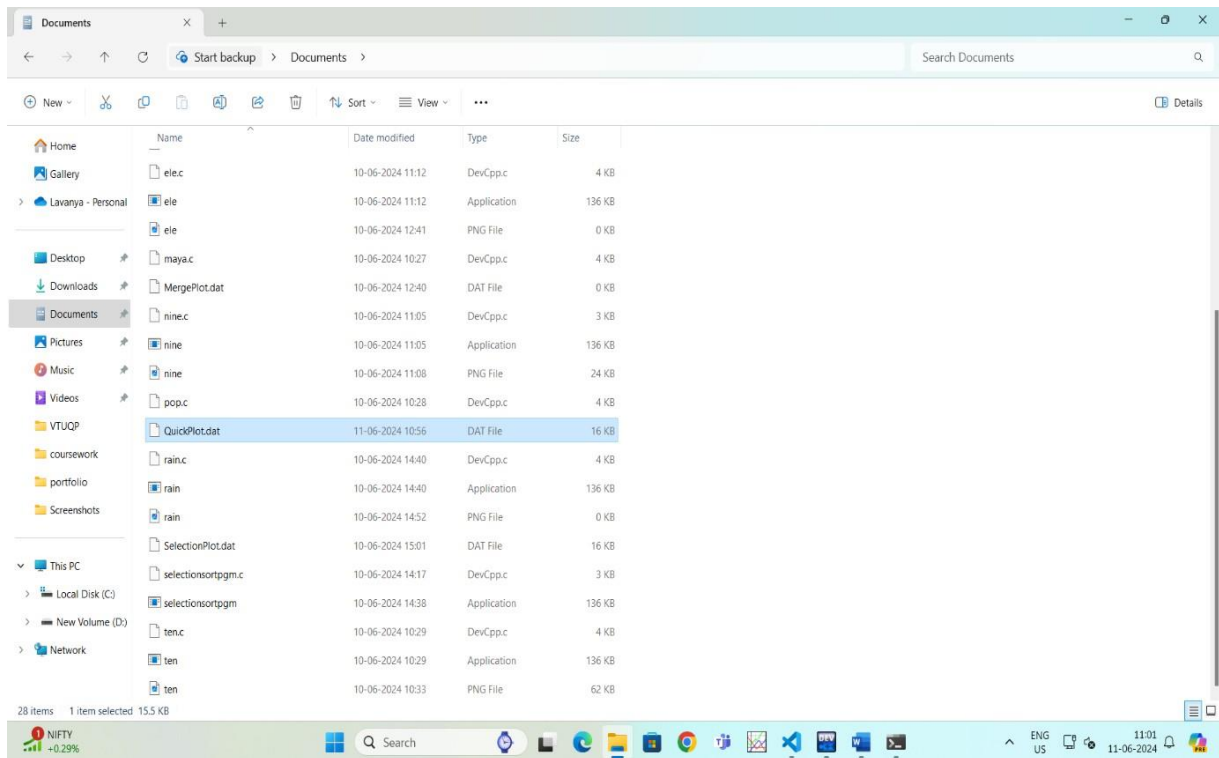
Snapshot 2.21

Step 11: Setting data file path

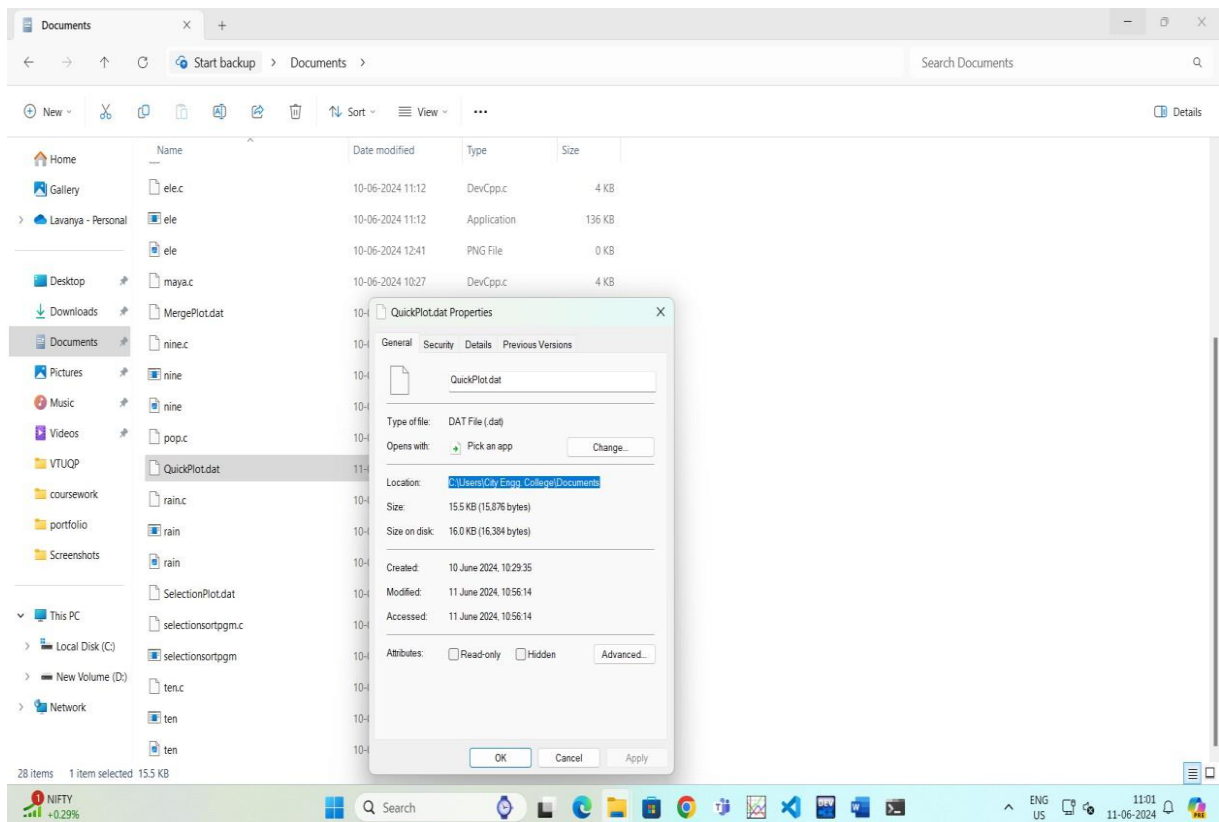
By default, the “.dat” or the data file will be located in document section.

Right click on the “filename.dat” and go to properties, there copy the filepath along with filename.

(Refer Snapshots 2.22 and 2.23)



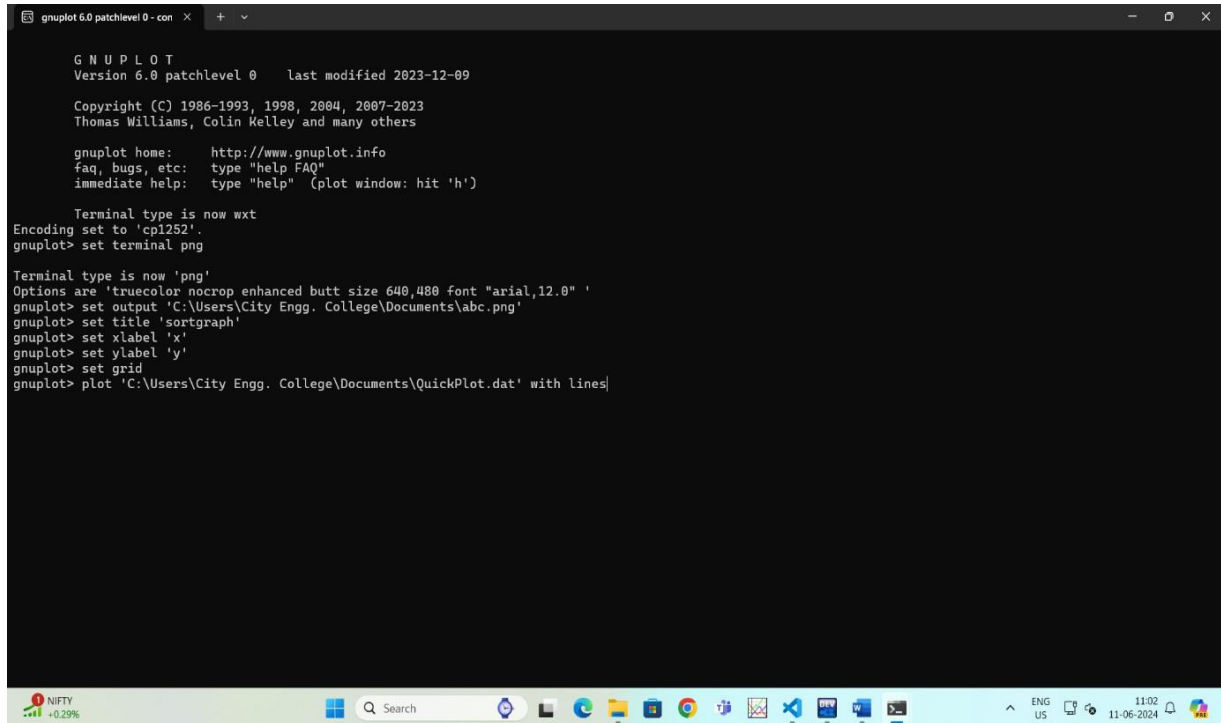
Snapshot 2.22



Snapshot 2.23

Step 12: Command for plotting graph

Now give the command **gnuplot > plot 'file location of .dat file\filename.dat' with lines** and click enter. (Refer Snapshot 2.24 to 2.25)



```
gnuplot 6.0 patchlevel 0 - con x + v

GNU PLOT
Version 6.0 patchlevel 0 last modified 2023-12-09

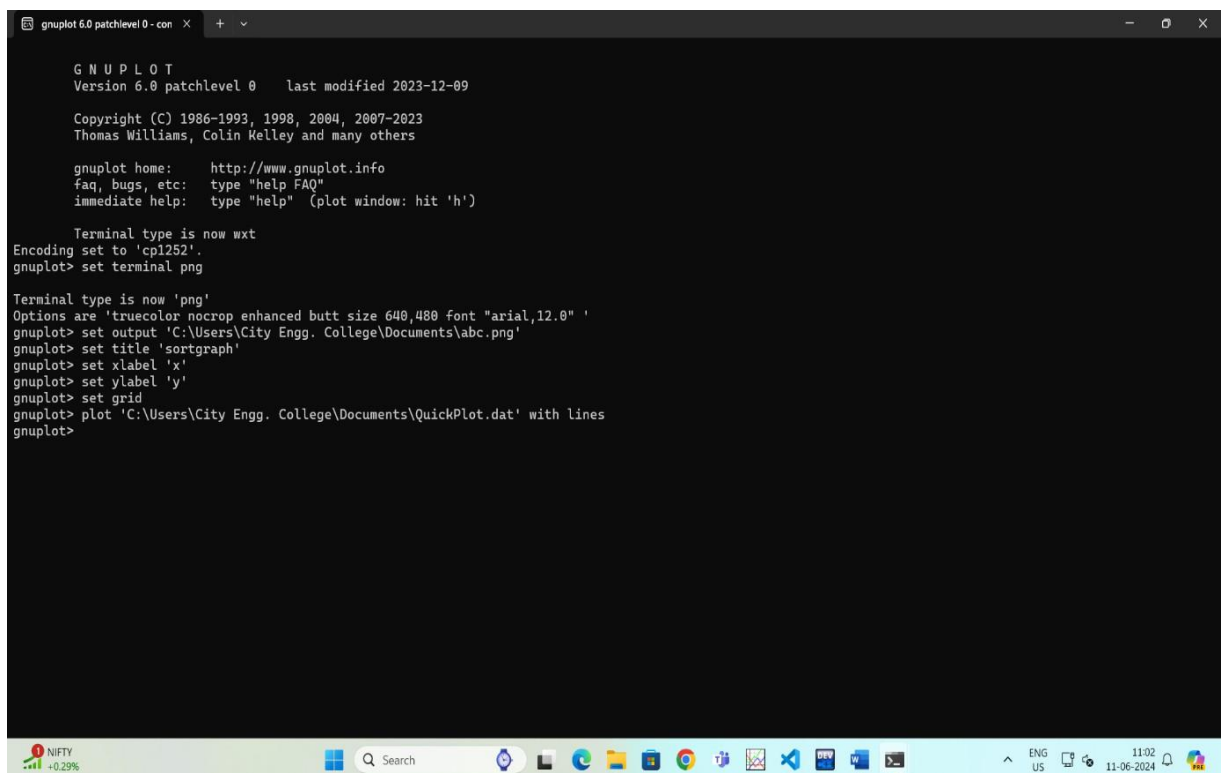
Copyright (C) 1986-1993, 1998, 2004, 2007-2023
Thomas Williams, Colin Kelley and many others

gnuplot home: http://www.gnuplot.info
faq, bugs, etc: type "help FAQ"
immediate help: type "help" (plot window: hit 'h')

Terminal type is now wxt
Encoding set to 'cp1252'.
gnuplot> set terminal png

Terminal type is now 'png'
Options are 'truecolor nocrop enhanced butt size 640,480 font "arial,12.0" '
gnuplot> set output 'C:\Users\City Engg. College\Documents\abc.png'
gnuplot> set title 'sortgraph'
gnuplot> set xlabel 'x'
gnuplot> set ylabel 'y'
gnuplot> set grid
gnuplot> plot 'C:\Users\City Engg. College\Documents\QuickPlot.dat' with lines
```

Snapshot 2.24



```
gnuplot 6.0 patchlevel 0 - con x + v

GNU PLOT
Version 6.0 patchlevel 0 last modified 2023-12-09

Copyright (C) 1986-1993, 1998, 2004, 2007-2023
Thomas Williams, Colin Kelley and many others

gnuplot home: http://www.gnuplot.info
faq, bugs, etc: type "help FAQ"
immediate help: type "help" (plot window: hit 'h')

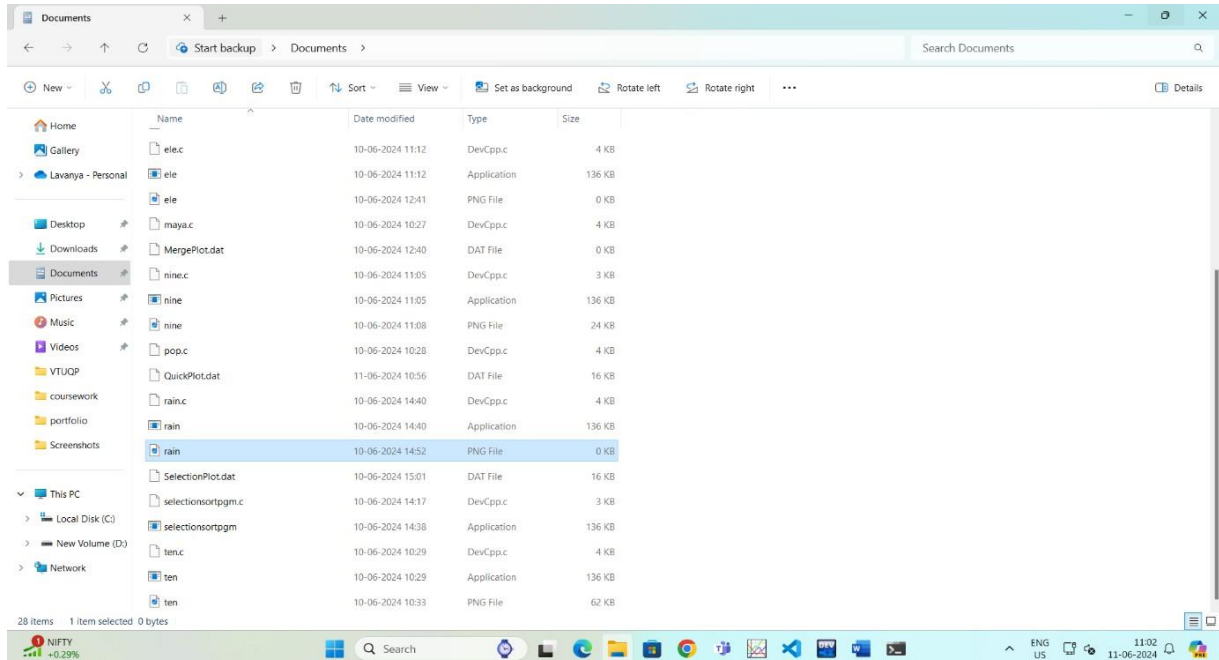
Terminal type is now wxt
Encoding set to 'cp1252'.
gnuplot> set terminal png

Terminal type is now 'png'
Options are 'truecolor nocrop enhanced butt size 640,480 font "arial,12.0" '
gnuplot> set output 'C:\Users\City Engg. College\Documents\abc.png'
gnuplot> set title 'sortgraph'
gnuplot> set xlabel 'x'
gnuplot> set ylabel 'y'
gnuplot> set grid
gnuplot> plot 'C:\Users\City Engg. College\Documents\QuickPlot.dat' with lines
gnuplot>
```

Snapshot 2.25

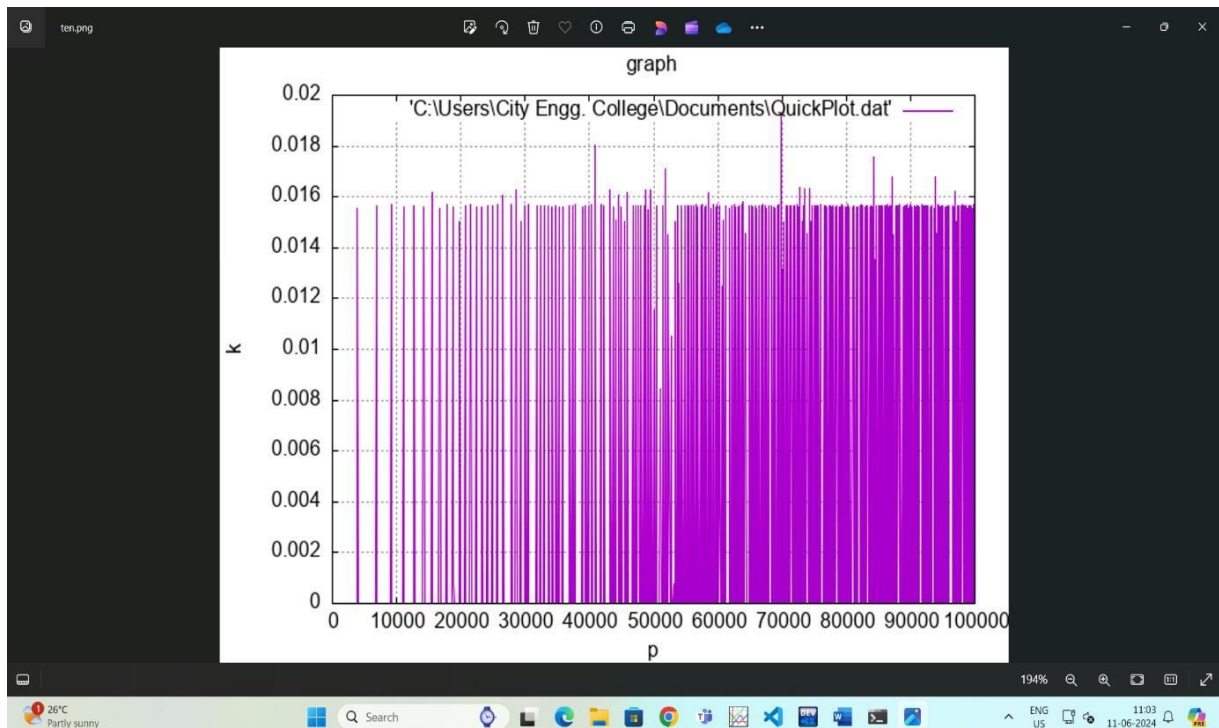
Step 13: PNG file for graph

By default the PNG file will be present in documents, double click on the PNG file to view the image of graph plotted.



Snapshot 2.26

Step 14: Graph Plotted



Snapshot 2.27

PROGRAM: 9

AIM: 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.

DEFINITION: Selection sort is a sorting routine that scans a list of items repeatedly and, on each pass, selects the item with the lowest value and places it in its final position. It is based on brute force approach. Sequential search is a $\Theta(n^2)$ algorithm on all inputs.

Sort a given set of elements using Selection sort and determine the time required to sort elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

ALGORITHM:

SelectionSort (A [0...n-1])

//sort a given array by selection sort

//input: A[0...n-1] of orderable elements

Output: Array a[0...n-1] Sorted in ascending order

for i ← 0 to n-2 do

 min ← i

 for j ← i+1 to n-1 do

 if A[j] < A[min] min ← j

 swap A[i] and A[min]

PROGRAM CODE:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```



```
#include <sys/time.h>
#include <time.h>

void fnGenRandInput(int [], int);
void fnDispArray(int [], int);
void fnSelectionSort(int [], int);
inline void fnSwap(int*, int*);

inline void fnSwap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int main() {
    FILE *fp;
    struct timeval tv;
    double dStart, dEnd;
    int iaArr[500000], iNum, i, iChoice;

    for (;;) {
        printf("\n1.Plot the Graph\n2.Selection Sort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d", &iChoice);

        switch (iChoice) {
            case 1:
                printf("\nBefore file creation\n");
                fp = fopen("SelectionPlot.dat", "w");
                if (fp == NULL) {
                    perror("Error opening file SelectionPlot.dat");
                    printf("Error: Unable to create SelectionPlot.dat file for\nwriting.\n");
                    exit(1);
                }
                printf("File created successfully\n");

                printf("\nGenerating data...\n");
```

```
        for (i = 100; i < 100000; i += 100) {
            fnGenRandInput(iaArr, i);
            gettimeofday(&tv, NULL);
            dStart = tv.tv_sec + (tv.tv_usec / 1000000.0);
            fnSelectionSort(iaArr, i);
            gettimeofday(&tv, NULL);
            dEnd = tv.tv_sec + (tv.tv_usec / 1000000.0);
            fprintf(fp, "%d\t%lf\n", i, dEnd - dStart);
        }
        fclose(fp);
        printf("\nData File generated and stored in file < SelectionPlot.dat
>.\n Use a plotting utility\n");
        break;
        case 2:
            printf("\nEnter the number of elements to sort\n");
            scanf("%d", &iNum);
            printf("\nUnsorted Array\n");
            fnGenRandInput(iaArr, iNum);
            fnDispArray(iaArr, iNum);
            fnSelectionSort(iaArr, iNum);
            printf("\nSorted Array\n");
            fnDispArray(iaArr, iNum);
            break;

        case 3:
            exit(0);
    }
}
return 0;
}

void fnSelectionSort(int arr[], int n) {
    int i, j, min_idx;
    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }
    }
}
```

```
        }
        fnSwap(&arr[min_idx], &arr[i]);
    }
}

void fnGenRandInput(int X[], int n) {
    int i;
    srand(time(NULL));
    for (i = 0; i < n; i++) {
        X[i] = rand() % 10000;
    }
}

void fnDispArray(int X[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf(" %5d \n", X[i]);
    }
}
```

OUTPUT:

Enter your choice

1

Before file creation

File created successfully

Generating data...

Enter your choice

2

Enter the number of elements to sort

6

Unsorted Array

7409

5636

2361

4521

7916

3406

Sorted Array

2361

3406

4521

5636

7409

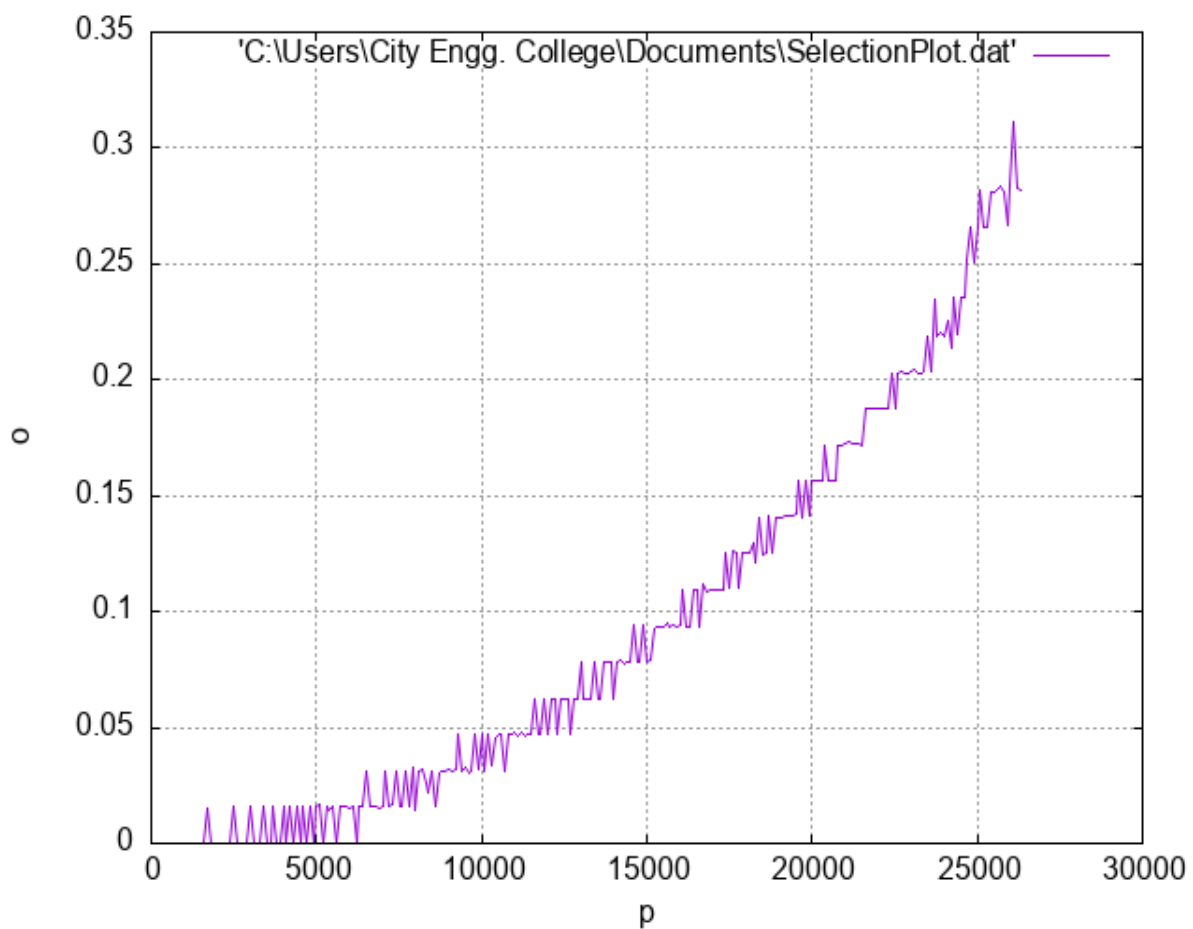
7916

1. Plot the Graph

2. Selection Sort

3. Exit

Enter your choice



PROGRAM:10

AIM: Design and implement C/C++ Program to sort a given set of n integer elements using Quick 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.

DEFINITION: Quick sort is based on the Divide and conquer approach. Quick sort divides array according to their value. Partition is the situation where all the elements before some position s are smaller than or equal to $A[s]$ and all the elements after position s are greater than or equal to $A[s]$.

Efficiency: $C_{best}(n) \in \Theta(n \log_2 n)$, $C_{worst}(n) \in \Theta(n^2)$, $C_{avg}(n) \in 1.38n \log_2 n$.

ALGORITHM:

Quick sort (A[l...r])

```
// Sorts a sub array by quick sort
//Input: A sub array A[l..r] of A[0..n-1] ,defined by its left and right indices l
//and r
// Output: The sub array A[l..r] sorted in non-decreasing order
    if  $l < r$ 
         $s = \text{Partition}(A[l..r])$  //s is a split position
        Quick sort (A [l ...s-1])
        Quick sort (A [s+1...r])
```

Partition (A[l...r])

```
//Partition a sub array by using its first element as a pivot
// Input: A sub array A [l...r] of A[0...n-1] defined by its left and right indices
l and // r ( $l < r$ )
// Output: A partition of A [l...r], with the split position returned as this
function's value
```

```
p=A[l]
i=l;
```

```
j=r+1;
repeat
    delay (500);
    repeat i= i+1 until A[i] >= p
    repeat j=j-1 until A[J] <= p
    Swap (A[i], A[j])
until I >=j
Swap (A[i], A[j]) // Undo last Swap when i>= j
Swap (A[l], A[j])
Return j
```

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
void fnGenRandInput(int [], int);
void fnDispArray(int [], int);
int fnPartition(int [], int, int);
void fnQuickSort(int [], int, int);
inline void fnSwap(int*, int*);
inline void fnSwap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
int main() {
    FILE *fp;
    struct timeval tv;
    double dStart, dEnd;
    int iaArr[500000], iNum, iPos, iKey, i, iChoice;
    for (;;) {
        printf("\n1.Plot the Graph\n2.QuickSort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d", &iChoice);
        switch (iChoice) {
```

```
case 1:
    printf("\nBefore file creation\n");
    fp = fopen("QuickPlot.dat", "w");
    if (fp == NULL) {
        perror("Error opening file QuickPlot.dat");
        printf("Error: Unable to create QuickPlot.dat file for writing.\n");
        exit(1);
    }
    printf("File created successfully\n");

    printf("\nGenerating data...\n");
    for (i = 100; i < 100000; i += 100) {
        fnGenRandInput(iaArr, i);
        gettimeofday(&tv, NULL);
        dStart = tv.tv_sec + (tv.tv_usec / 1000000.0);
        fnQuickSort(iaArr, 0, i - 1);
        gettimeofday(&tv, NULL);
        dEnd = tv.tv_sec + (tv.tv_usec / 1000000.0);
        fprintf(fp, "%d\t%lf\n", i, dEnd - dStart);
    }
    fclose(fp);
    printf("\nData File generated and stored in file < QuickPlot.dat >.\n\n");
    Use a plotting utility\n");
    break;
case 2:
    printf("\nEnter the number of elements to sort\n");
    scanf("%d", &iNum);
    printf("\nUnsorted Array\n");
    fnGenRandInput(iaArr, iNum);
    fnDispArray(iaArr, iNum);
    fnQuickSort(iaArr, 0, iNum - 1);
    printf("\nSorted Array\n");
    fnDispArray(iaArr, iNum);
    break;
case 3:
    exit(0);
}
```

```
    }
    return 0;
}

int fnPartition(int a[], int l, int r) {
    int i, j, temp;
    int p;
    p = a[l];
    i = l;
    j = r + 1;
    do {
        do {
            i++;
        } while (a[i] < p);

        do {
            j--;
        } while (a[j] > p);
        fnSwap(&a[i], &a[j]);
    }
    while (i < j);
    fnSwap(&a[i], &a[j]);
    fnSwap(&a[l], &a[j]);
    return j;
}

void fnQuickSort(int a[], int l, int r) {
    int s;
    if (l < r) {
        s = fnPartition(a, l, r);
        fnQuickSort(a, l, s - 1);
        fnQuickSort(a, s + 1, r);
    }
}

void fnGenRandInput(int X[], int n) {
    int i;
    srand(time(NULL));
    for (i = 0; i < n; i++) {
        X[i] = rand() % 10000;
    }
}
```



```
    }  
}  
void fnDispArray(int X[], int n) {  
    int i;  
    for (i = 0; i < n; i++) {  
        printf(" %5d \n", X[i]);  
    }  
}
```

OUTPUT:

Enter your choice

1

Before file creation

File created successfully

Generating data...

Data File generated and stored in file < QuickPlot.dat >.

Use a plotting utility

1.Plot the Graph

2.QuickSort

3.Exit

Enter your choice

2

Enter the number of elements to sort

5

Unsorted Array

6814

2738

7882

5910

1207

Sorted Array

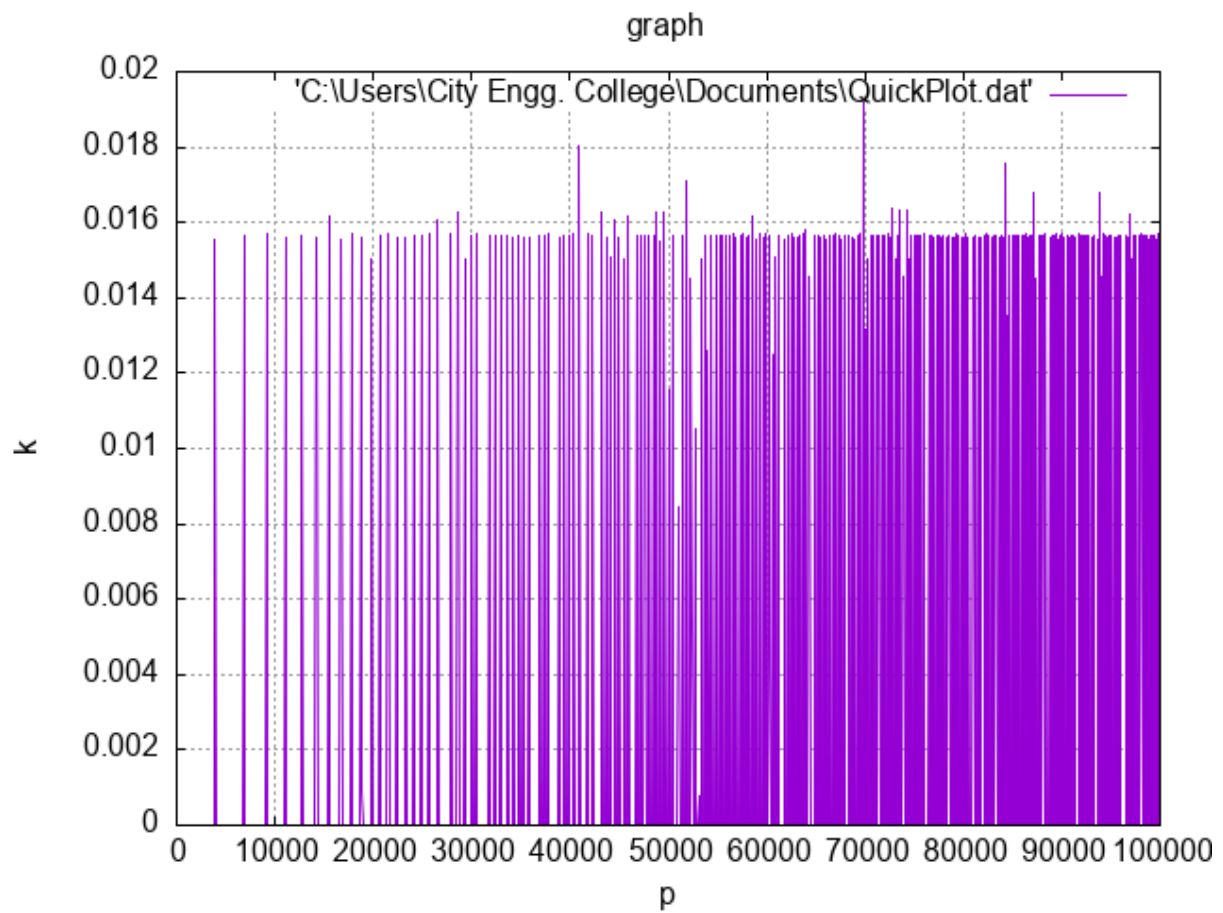
1207

2738

5910

6814

7882



PROGRAM: 11

AIM: 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.

DEFINITION:

Merge sort is a sort algorithm based on divide and conquer technique. It divides the array element based on the position in the array. The concept is that we first break the list into two smaller lists of roughly the same size, and then use merge sort recursively on the subproblems, until they cannot subdivide anymore. Then, we can merge by stepping through the lists in linear time. Its time efficiency is $\Theta(n \log n)$.

ALGORITHM:

Merge sort ($A[0 \dots n-1]$)

// Sorts array $A[0..n-1]$ by Recursive merge sort

// Input : An array $A[0..n-1]$ elements

// Output : Array $A[0..n-1]$ sorted in non decreasing order

If $n > 1$

Copy $A[0 \dots (n/2)-1]$ to $B[0 \dots (n/2)-1]$

Copy $A[(n/2) \dots n-1]$ to $C[0 \dots (n/2)-1]$

Mergesort ($B[0 \dots (n/2)-1]$)

Mergesort ($C[0 \dots (n/2)-1]$)

Merge(B, C, A)

Merge ($B[0 \dots p-1], C[0 \dots q-1], A[0 \dots p+q-1]$)

// merges two sorted arrays into one sorted array

// Input : Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted

// Output : Sorted array $A[0 \dots p+q-1]$ of the elements of B and C

$I = 0;$

$J = 0;$

$K = 0;$

While $I < p$ and $j < q$ do

If $B[i] \leq C[j]$

$A[k] = B[I]; \quad I = I + 1;$

```
    Else
        A[k] = B[i];  I=i+1
    K=k+1;
    If I == p
        Copy C [ j.. q-1] to A[k....p+q-1]
    else
        Copy B [I ... p-1] to A[k ...p+q-1
```

PROGRAM CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <time.h>


void fnGenRandInput(int [], int);

void fnDispArray(int [], int);

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

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


int main() {

    FILE *fp;

    struct timeval tv;

    double dStart, dEnd;

    int iaArr[500000], iNum, i, iChoice;


    do {

        printf("\n1. Plot the Graph\n2. Merge Sort\n3. Exit\n");

        printf("Enter your choice: ");
```

```
scanf("%d", &iChoice);

switch (iChoice) {
    case 1:
        printf("\nBefore file creation\n");
        fp = fopen("MergePlot.dat", "w");
        if (fp == NULL) {
            perror("Error opening file MergePlot.dat");
            printf("Error: Unable to create MergePlot.dat file for writing.\n");
            exit(1);
        }

        printf("File created successfully\n");
        printf("\nGenerating data...\n");

        for (i = 100; i < 100000; i += 100) {
            fnGenRandInput(iaArr, i);

            gettimeofday(&tv, NULL);
            dStart = tv.tv_sec + (tv.tv_usec / 1000000.0);

            fnMergeSort(iaArr, 0, i - 1);

            gettimeofday(&tv, NULL);
            dEnd = tv.tv_sec + (tv.tv_usec / 1000000.0);
```

```
        fprintf(fp, "%d\t%lf\n", i, dEnd - dStart);
    }

    fclose(fp);

    printf("\nData File generated and stored in file <MergePlot.dat>.\nUse
a plotting utility\n");
    break;

case 2:
    printf("\nEnter the number of elements to sort: ");
    scanf("%d", &iNum);
    printf("\nUnsorted Array:\n");

    fnGenRandInput(iaArr, iNum);
    fnDispArray(iaArr, iNum);

    fnMergeSort(iaArr, 0, iNum - 1);
    printf("\nSorted Array:\n");
    fnDispArray(iaArr, iNum);

    printf("\nSorting completed.\n");
    break;

case 3:
    printf("\nExiting the program...\n");
    exit(0);
```

```
        default:
            printf("\nInvalid choice! Please enter a valid option.\n");
        }
    } while (iChoice != 3);

    return 0;
}

void fnMerge(int a[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    for (i = 0; i < n1; i++)
        L[i] = a[l + i];
    for (j = 0; j < n2; j++)
        R[j] = a[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
```

```
    if (L[i] <= R[j]) {  
        a[k] = L[i];  
        i++;  
    } else {  
        a[k] = R[j];  
        j++;  
    }  
    k++;  
}
```

```
while (i < n1) {  
    a[k] = L[i];  
    i++;  
    k++;  
}
```

```
while (j < n2) {  
    a[k] = R[j];  
    j++;  
    k++;  
}
```

```
free(L);  
free(R);  
}
```



```
void fnMergeSort(int a[], int l, int r) {  
    if (l < r) {  
        int m = l + (r - l) / 2;  
        fnMergeSort(a, l, m);  
        fnMergeSort(a, m + 1, r);  
        fnMerge(a, l, m, r);  
    }  
}
```

```
void fnGenRandInput(int X[], int n) {  
    int i;  
    srand(time(NULL));  
    for (i = 0; i < n; i++) {  
        X[i] = rand() % 10000;  
    }  
}
```

```
void fnDispArray(int X[], int n) {  
    int i;  
    for (i = 0; i < n; i++) {  
        printf("%5d", X[i]);  
    }  
    printf("\n");  
}
```

OUTPUT:

1. Plot the Graph

2. Merge Sort

3. Exit

Enter your choice: 1

Before file creation

File created successfully

Generating data...

Enter your choice: 2

Enter the number of elements to sort: 5

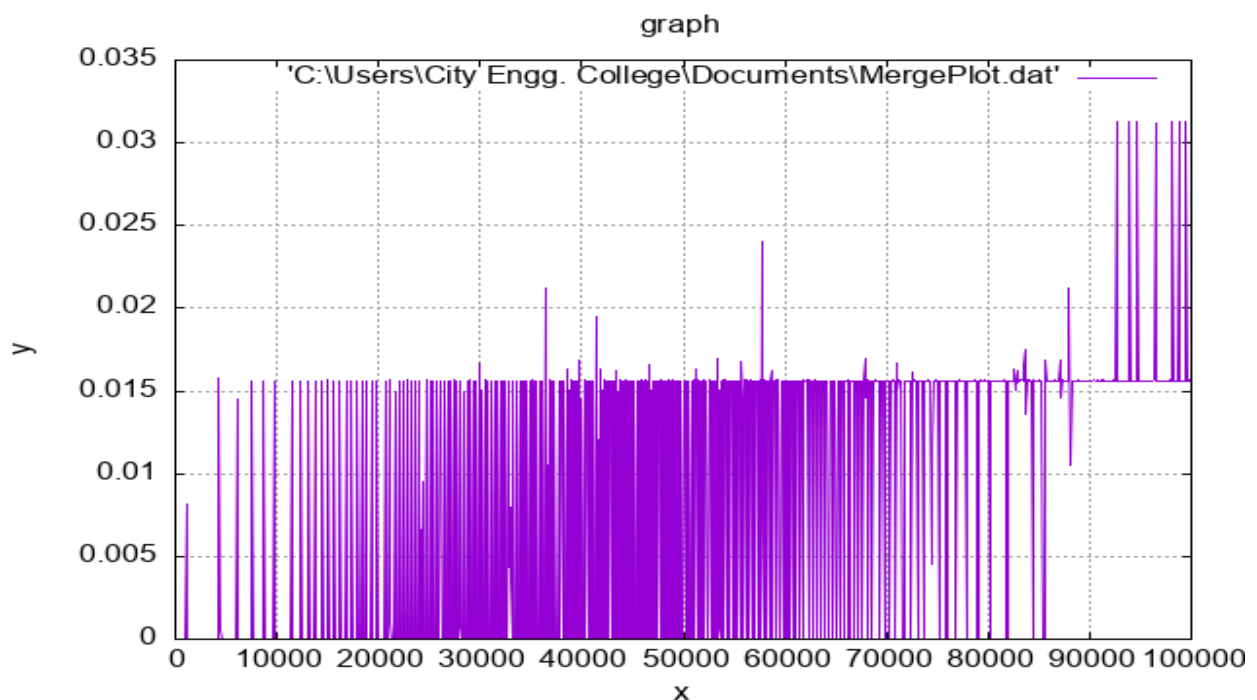
Unsorted Array:

6239 1563 9516 7839 8082

Sorted Array:

1563 6239 7839 8082 9516

Sorting completed.



PROGRAM: 12

AIM: Design and implement C/C++ Program for N Queen's problem using Backtracking.

DEFINITION: The object is to place queens on a chess board in such as way as no queen can capture another one in a single move
Recall that a queen can move horizontal, vert, or diagonally an infinite distance
This implies that no two queens can be on the same row, col, or diagonal

ALGORITHM:

```
/* outputs all possible acceptable positions of n queens on n x n chessboard */
// Initialize x [ ] to zero
// Set k = 1 start with first queen
Repeat for i = 1 to n // try all columns one by one for kth queen
if Place (k, i) true then
    {
        x(k) = i // place kth queen in column i
        if (k=n) all queens placed and hence print output (x[ ])
        else NQueens(K+1,n) //try for next queen
    }
Place (k,i)
/* finds if kth queen in kth row can be placed in column i or not; returns true if
queen can be placed */
// x[1,2, . . . k-1] have been defined
//queens at (p, q) & (r, s) attack if |p-r| = |q-s|
Repeat for j = 1 to (k-1)
    if any earlier jth queen is in ith column ( x[j]= i)
or in same diagonal ( abs(x[ j] - i) = abs( j - k) )
    then kth queen cannot be placed (return false)
return true (as all positions checked and no objection)
```

PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
```

```
#define MAX 50
```

```
int can_place(int c[],int r)
{
    int i;
    for(i=0;i<r;i++)
        if(c[i]==c[r] || (abs(c[i]-c[r])==abs(i-r)))
            return 0;
    return 1;
}
```

```
void display(int c[],int n)
{
    int i,j;
    char cb[10][10];
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            cb[i][j]='-';
    for(i=0;i<n;i++)
        cb[i][c[i]]='Q';
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%c",cb[i][j]);
        printf("\n");
    }
}
```

```
void n_queens(int n)
{
    int r;
    int c[MAX];
    c[0]=-1;
    r=0;
    while(r>=0)
    {
        c[r]++;
        while(c[r]<n && !can_place(c,r))
            r++;
        if(r==n)
            return;
        r--;
    }
}
```

```
        c[r]++;
        if(c[r]<n)
        {      if(r==n-1)
                  {      display(c,n);
                        printf("\n\n");
                  }
                else
                {      r++;
                        c[r]=-1;
                }
        }
        else
        r--;
    }
}

void main()
{
    int n;
    clrscr();
    printf("\nEnter the no. of queens:");
    scanf("%d",&n);
    n_queens(n);
    getch();
}
```

OUTPUT:

Enter the no. of queens:4

-Q--

---Q

Q---

--Q-

--Q-

Q---

---Q

-Q---

Sample Viva Questions and Answers

1) Explain what is an algorithm in computing?

An algorithm is a well-defined computational procedure that take some value as input and generate some value as output. In simple words, it's a sequence of computational steps that converts input into the output.

2) Explain what is time complexity of Algorithm?

Time complexity of an algorithm indicates the total time needed by the program to run to completion. It is usually expressed by using the big O notation.

3) The main measure for efficiency algorithm are-Time and space

4. The time complexity of following code:

```
int a = 0;
for (i = 0; i < N; i++) {
    for (j = N; j > i; j--) {
        a = a + i + j;
    }
}
```

} Ans $O(n*n)$

5) What does the algorithmic analysis count?

6) The number of arithmetic and the operations that are required to run the program

7) Examples of $O(1)$ algorithms are _____

A Multiplying two numbers.

B assigning some value to a variable

C displaying some integer on console

8) Examples of $O(n^2)$ algorithms are_____.

A Adding of two Matrices

B Initializing all elements of matrix by zero

9) The complexity of three algorithms is given as: $O(n)$, $O(n^2)$ and $O(n^3)$. Which should execute slowest for large value of n ?

All will execute in same time.

10) In quick sort, the number of partitions into which the file of size n is divided by a selected record is 2.

The three factors contributing to the sort efficiency considerations are the efficiency in coding, machine run time and the space requirement for running the procedure.

11) How many passes are required to sort a file of size n by bubble sort method?

$N-1$

12) How many number of comparisons are required in insertion sort to sort a file if the file is sorted in reverse order?

A. N^2

13) How many number of comparisons are required in insertion sort to sort a file if the file is already sorted?

$N-1$

14) In quick sort, the number of partitions into which the file of size n is divided by a selected record is 2

15) The worst-case time complexity of Quick Sort is_____. $O(n^2)$

16)The worst-case time complexity of Bubble Sort is_____. $O(n^2)$

17) The worst-case time complexity of Merge Sort is_____. $O(n \log n)$

18) The algorithm like Quick sort does not require extra memory for carrying out the sorting procedure. This technique is called _____. in-place

19) Which of the following sorting procedures is the slowest?

A. Quick sort

B. Heap sort

C. Shell sort

D. Bubble sort

20) The time factor when determining the efficiency of algorithm is measured by Counting the number of key operations

21) The concept of order Big O is important because

A. It can be used to decide the best algorithm that solves a given problem

22) The running time of insertion sort is

A. $O(n^2)$

23) A sort which compares adjacent elements in a list and switches where necessary is ____.

A. insertion sort

24) The correct order of the efficiency of the following sorting algorithms according to their overall running time comparison is

bubble > selection > insertion

25) The total number of comparisons made in quick sort for sorting a file of size n , is

A. $O(n \log n)$

26) Quick sort efficiency can be improved by adopting

A. non-recursive method

27) For the improvement of efficiency of quick sort the pivot can be _____.
“the mean element”

28) What is the time complexity of linear search? $\Theta(n)$

29) What is the time complexity of binary search? $\Theta(\log_2 n)$

30) What is the major requirement for binary search?

The given list should be sorted.

31) What are important problem types? (or) Enumerate some important types of problems.

1. Sorting 2. Searching
3. Numerical problems 4. Geometric problems
5. Combinatorial Problems 6. Graph Problems
7. String processing Problems

32) Name some basic Efficiency classes

1. Constant 2. Logarithmic 3. Linear 4. $n \log n$
5. Quadratic 6. Cubic 7. Exponential 8. Factorial

33) What are algorithm design techniques?

Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing. General design techniques are: (i) Brute force (ii) divide and conquer (iii) decrease and conquer (iv) transform and conquer (v) greedy technique (vi) dynamic programming (vii) backtracking (viii) branch and bound

34). How is an algorithm's time efficiency measured?

Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm's innermost loop.

35) Explain the greedy method.

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one can devise an algorithm that works in stages considering one input at a time.

36) Define feasible and optimal solution.

Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution. A feasible solution either maximizes or minimizes the given objective function is called an optimal solution.

37) What are the constraints of knapsack problem?

To maximize $\sum p_i x_i$

The constraint is : $\sum w_i x_i \leq m$ and $0 \leq x_i \leq 1$ $1 \leq i \leq n$

where m is the bag capacity, n is the number of objects and for each object i w_i and p_i are the weight and profit of object respectively.

38) Specify the algorithms used for constructing Minimum cost spanning tree.

a) Prim's Algorithm

b) Kruskal's Algorithm

39). State single source shortest path algorithm (Dijkstra's algorithm).

For a given vertex called the source in a weighted connected graph, find shortest paths to all its other vertices. Dijkstra's algorithm applies to graph with non-negative weights only.

40). State efficiency of prim's algorithm.

$O(|V|^2)$ (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)

$O(|E| \log |V|)$ (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

41) State Kruskal Algorithm.

The algorithm looks at a MST for a weighted connected graph as an acyclic subgraph with $|V|-1$

edges for which the sum of edge weights is the smallest.

42) State efficiency of Dijkstra's algorithm.

$O(|V|^2)$ (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)