<table>
<tr><th colspan="2">Name of the Subject: Algorithm Analysis & Design</th></tr>
<tr><th>Assign. No.</th><th>Name of the Assignment along with the description like language to be used if applicable, sample input, expected output, observations or assumptions, references etc.</th></tr>
<tr><td>1.</td><td>
Write a program to implement merge sort.
Your program should take as input as input size (range 1-20, it has to be first command line argument always) and the entire sequence of numbers (only 2-bit integers, range 10-99) which are to be sorted and program should output sorted sequence as output, each number separated by a space character with no newline characters, as shown in Sample Run.

Sample Run:

It assumes that your program is compiled into megeSort.out executable and takes input.txt file as input and output.txt file as output.

./mergeSort.out “5” “45 12 56 98 23” # first command line argument is input size

12 3 45 56 98
</td></tr>
<tr><td>2.</td><td>
Write a program to compare insertion sort and merge sort.
Your program should take as input an integer representing input size (N), a text file which contains N random numbers to be sorted and an output text file that shall eventually contain the sorted sequence. Perform sorting of same set of random numbers through insertion sort and merge sort for different values of input size and fill in the following table. Prepare a graph using the excel plotting tools, take its printout and paste it in practical file.
<table>
<tr><th rowspan="2">Input Size (N)</th><th colspan="2">Running Time (seconds)</th></tr>
<tr><th>Insertion Sort</th><th>Merge Sort</th></tr>
<tr><td>1000</td><td></td><td></td></tr>
<tr><td>10000</td><td></td><td></td></tr>
<tr><td>100000</td><td></td><td></td></tr>
<tr><td>1000000</td><td></td><td></td></tr>
</table>
</td></tr>
</table>

	You may use the following code to compute the running time. *#include<iostream>* *#include<stdlib.h>* *#include<time.h>* *using namespace std;* *int main (int argc, char *argv[])* *{* *long int count = 0;* *long double startTime, endTime;* *long double timeTaken;* *long int numOfIterations = atoi(argv[1]); /* number of iterations as input */* *startTime = clock ();* *while (count++ != numOfIterations) /* time calculation begins */* *{* *.....* *..... /* computations to be measured */* *.....* *} /* time calculation ends */*

```
endTime = clock ();

timeTaken = ( (endTime - startTime) / CLOCKS_PER_SEC ) / numOfIterations;

cout << "Time taken:" << timeTaken << endl;

return 0;

}
```

Sample Run:

It assumes that your program is compiled into timeMeasure.out executable and takes input.txt file as input and output.txt file as output.

./timeMeasure.out 10000 input.txt output.txt

3. Write a program to implement insertion sort and improved insertion sort.

Your program should take as input an integer representing input size (N), a text file which contains N random numbers to be sorted and an output text file that shall eventually contain the sorted sequence. Perform sorting of same set of random numbers through insertion sort and improved insertion sort for different values of input size and fill in the following table by measuring the number of comparisons done during sorting. Prepare a graph using the excel plotting tools, take its printout and paste it in practical file.

Input Size (N)	Number of comparisons	
	Insertion Sort	Merge Sort
1000		
10000		
100000		
1000000		

Sample Run:

It assumes that your program is compiled into compareMeasure.out executable and takes input.txt file as input and output.txt file as output.

./compareMeasure.out 10000 input.txt output.txt

4. Write a program to compare quick sort with randomized quick sort.

Your program should take as input an integer representing input size (N), a text file which contains N random numbers to be sorted and an output text file that shall eventually contain the sorted sequence. Perform sorting of same set of random numbers through quick sort and randomized sort for different values of input size and fill in the following table by measuring the number of *exchanges* done during *partition ( )* procedure calls in each of the sorting methods. Prepare a graph using the excel plotting tools, take its printout and paste it in practical file.

Input Size (N)	Number of exchanges	
	Quick Sort	Randomized Quick Sort
1000		
10000		
100000		
1000000		

5. Write a program to compare conventional matrix multiplication with strassen's matrix multiplication.

Your program should take as input an integer representing input size (N) representing order of the matrix, program should generate two N x N matrices of random integers (integer values are only single digit ranging between 0 - 9) and multiply them based on conventional method and Strassen's matrix multiplication method.

Time taken for various input sizes are the be measured as per table below.

Input Size (N)	Time Taken	
	Conventional method	Strassen's method
8		
32		
256		
512		
1024		

6.

Write a program to compute optimal binary search tree.

Your program should take as input an input file which contains the keys (and dummy keys) along with their probabilities as input. Program should output the resulting optimal binary search tree as per the format of output file.

eg.

Input file (input.txt)

```
k1 0.15
k2 0.10
k3 0.05
k4 0.10
k5 0.20
d0 0.05
d1 0.10
d2 0.05
d3 0.05
d4 0.05
d5 0.10
```

Output file (output.txt)

```
k2 is the root
k1 is the left child of k2
d0 is the left child of k1
d1 is the right child of k1
k5 is the right child of k2
k4 is the left child of k5
k3 is the left child of k4
d2 is the left child of k3
d3 is the right child of k3
d4 is the right child of k4
d5 is the right child of k5
```

	Sample Run: It assumes that your program is compiled into obst.out executable and takes input.txt file as input and output.txt file as output. ./obst.out input.txt output.txt
7.	Write a program to solve activity selection problem. Your program should take as input a text file which contains three columns (each column separated with a single space), with each row specifying activity ID, start time of activity and finish time of activity, respectively. Output of the problem should be maximal size subset of mutually compatible activities in order of their scheduling, each separated by comma as shown below. eg. Input file (input.txt) It contains five activities with ID's 1 ... 5. Activity 1 begins at time 1 and ends at time 5, and so on ... 1 1 5 2 3 4 3 2 3 4 4 7 5 5 8 Output file (output.txt)

	It contains the ID's of activities, separated by comma, in the order in which they can be scheduled. 3,2,4 Sample Run: It assumes that your program is compiled into activitySelector.out executable and takes input.txt file as input and output.txt file as output. ./activitySelector.out input.txt output.txt
8.	Write a program to solve Huffman coding problem. Your program should take as input a text file which contains two columns (each column separated with a single space), with each row specifying alphabet and frequency of alphabet, respectively. Output of the problem should be Huffman codes for each alphabet in another two column file (each column separated with a single space), with each row specifying alphabet and Huffman code for that alphabet. eg. Input file (input.txt)

It contains six alphabets a .... f.

Alphabet ‘a’ occurs 45 times, and so on ...

```
a 45
b 13
c 12
d 16
e 9
f 5
```

Output file (output.txt)

It contains the Huffman codes of the alphabets.

```
a 0
b 101
c 100
d 111
e 1101
f 1100
```

Sample Run:

It assumes that your program is compiled into huffmanCoding.out executable and takes input.txt file as input and

	output.txt file as output. ./huffmanCoding.out input.txt output.txt
9.	Write a program to solve task scheduling problem. Your program should take as input an input file which contains the task ID, deadline and penalty incurred for that task, respectively, each separate by a space. Each task is of unit time. Program should output the final optimal schedule containing the task ID, separated by comma. eg. Input file (input.txt) 1 4 70 2 2 60 3 4 50 4 3 40 5 1 30 6 4 20 7 6 10 Output file (output.txt)

	2,4,1,3,7,5,6 Sample Run: It assumes that your program is compiled into taskScheduling.out executable and takes input.txt file as input and output.txt file as output. ./taskScheduling.out input.txt output.txt
10.	Write a program to find degrees of vertices for a given undirected graph. Your program should take as input a text file which contains the input graph with the format shown as below. Output of the program should be degrees of all the vertices of the input graph as shown below. eg. Input file (input.txt) It represents an **u**ndirected graph with six (**6**) vertices specified in first line. The ID's of vertices are **a,b,...e,f** mentioned in second line. Subsequently, each line specifies an edge with starting and ending vertices, respectively. Say, **a,b** on third line means an edge (a,b) in the graph and so on ... (total seven edges in the graph).

```
6,u
a,b,c,d,e,f
a,b
d,e
f,b
b,e
a,c
c,d
b,d
```

Output file (output.txt)

It contains the ID's of vertices, along with their respective degree in each line separated with a single space.

```
a 2
b 4
c 2
d 3
e 2
f 1
```

Sample Run:

It assumes that your program is compiled into vertexDegree.out executable and takes input.txt file as input and output.txt file as output.

	./vertexDegree.out input.txt output.txt
11.	Write a program to solve path existence problem. Your program should take as input a text file which contains the **d**irected graph as input in the format as specified in the above question except that in this case the last line of input file contains the path whose existence is to be ascertained. Your program should ascertain whether the path specified in the last line of input file exist in the input graph or not. eg. Input file (input.txt) It represents a **d**irected graph with six (**4**) vertices specified in first line. The ID's of vertices are **1,2,3,4** mentioned in second line. Subsequently, each line specifies an edge with starting and ending vertices, respectively. Say, **1,2** on third line means a directed edge (1,2) in the graph and so on ... (total four edges in the graph). It specifies the path **1,4,3** on the last line. 4,d 1,2,3,4 1,2 2,3 1,4 4,3 1,4,3 Output file (output.txt)

	It contains 1 if path exists or 0 if it doesn't exist. 1 Sample Run: It assumes that your program is compiled into pathExists.out executable and takes input.txt file as input and output.txt file as output. ./pathExists.out input.txt output.txt
12.	Write a program that takes two inputs namely a graph and a source vertex. It computes the following. Your program should take as input an input file which contains the input undirected graph and a source vertex as another input on the last line. Program should output the vertices at a distance X from the source vertex on separate lines as shown below. <u>Note:</u> The vertices in adjacency list should be sorted with respect to their vertex ID. For example if vertices b, d and c are adjacent to vertex a, then Adj[a] = {b,c,d} and not {b,d,c}. eg.

Input file (input.txt)

Last line specifies the source vertex which is vertex **a** in the input file below.

```
5,u
a,b,c,d,e
a,b
c,a
b,d
c,e
b,e
c,d
a
```

Output file (output.txt)

Vertex a is at a distance 0 from a

Vertices b and c are at distance 1 from a

Vertices d and e are at distance 2 from a

```
a
b,c
d,e
```

Sample Run:

It assumes that your program is compiled into vertexDistance.out executable and takes input.txt file as input and output.txt file as output.

./vertexDistance.out input.txt output.txt

13. Write a program that takes a graph as input and outputs the vertices in the order in which they are discovered for the first time (discovery time when vertex becomes grey from white) when DFS is applied.

Note: The vertices in adjacency list should be sorted with respect to their vertex ID. For example if vertices b, d and c are adjacent to vertex a, then Adj[a] = {b,c,d} and not {b,d,c}.

eg.

Input file (input.txt)

```
6,u
a,b,c,d,e,f
a,b
a,c
c,f
b,d
d,e
e,f
b,e
```

	Output file (output.txt) a,b,d,e,f,c Sample Run: It assumes that your program is compiled into dfs.out executable and takes input.txt file as input and output.txt file as output. ./dfs.out input.txt output.txt
14.	Write a program to find topological sorting for the elements whose partial ordering is specified. Your program should take as input a text file which contains the partial ordering among the elements to be topologically sorted. Output of the program should be topological sorting of elements. Note: Your program shall be tested for the partial orderings such that only **one** topological sorting exists as the solution. eg. Input file (input.txt) It represents partial ordering among the given elements.

	4,d a,b,c,d b,a b,d a,c a,d c,d Output file (output.txt) It contains the topological sorting of the elements. b,a,c,d Sample Run: It assumes that your program is compiled into topoSort.out executable and takes input.txt file as input and output.txt file as output. ./topoSort.out input.txt output.txt
15.	Write a program to find strongly connected components in a graph. Your program should take as input a text file which contains the **d**irected graph as input. Your program should find the strongly connected

components in the given graph.

eg.

Input file (input.txt)

```
8,d
a,b,c,d,e,f,g,h
a,b
b,e
e,a
e,f
b,f
b,c
c,g
f,g
g,f
c,d
d,c
g,h
d,h
h,h
```

Output file (output.txt)

It contains strongly connected components, each on a separate line.

	a,b,e c,d f,g h Sample Run: It assumes that your program is compiled into stronglyCC.out executable and takes input.txt file as input and output.txt file as output. ./stronglyCC.out input.txt output.txt

Format for the long assignment: Please provide the assignment in the following template.

Signature of the Faculty