

**ALGORITHM ANALYSIS LAB –CSB 3105**

**1.SORTING ALGORITHM - MERGE SORT**

**Merge Sort**

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.Merge sort first divides the array into equal halves and then combines them in a sorted manner.

**Example:**

To understand merge sort, lets take an unsorted array as the following:

Unsorted Array

Merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. Now the array of 8 items is divided into two arrays of size 4.

Merge Sort Division

This does not change the sequence of appearance of items in the original. Now divide these two arrays into halves.

Merge Sort Division

Then further divide these arrays and it achieve atomic value which can no more be divided.

Merge Sort Division

First compare the element for each list and then combine them into another list in a sorted manner.14 and 33 are in sorted positions. Then compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. Then change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

Merge Sort Combine

In the next iteration of the combining phase, compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Merge Sort Combine

After the final merging, the list should look like this –

Merge Sort

**Algorithm**

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

**Step 1**: If it is only one element in the list it is already sorted, return.

**Step 2**: Divide the list recursively into two halves until it can no more be divided.

**Step 3**: Merge the smaller lists into new list in sorted order.

**Pseudo code:**

Merge sort works with recursion

procedure mergesort( var a as array )

if ( n == 1 ) return a

var l1 as array = a[0] ... a[n/2]

var l2 as array = a[n/2+1] ... a[n]

l1 = mergesort( l1 )

l2 = mergesort( l2 )

return merge( l1, l2 )

end procedure

procedure merge( var a as array, var b as array )

var c as array

while ( a and b have elements )

if ( a[0] > b[0] )

add b[0] to the end of c

remove b[0] from b

else

add a[0] to the end of c

remove a[0] from a

end if

end while

while ( a has elements )

add a[0] to the end of c

remove a[0] from a

end while

while ( b has elements )

add b[0] to the end of c

remove b[0] from b

end while

return c

end procedure

**ANALYSIS OF MERGE SORT**

**Algorithm:**

mergesort( int [] a, int left, int right)

{

if (right > left)

{

middle = left + (right - left)/2;

mergesort(a, left, middle);

mergesort(a, middle+1, right);

merge(a, left, middle, right);

}

}

**Assumption**: N is a power of two.

For N = 1: time is a constant (denoted by 1)

Otherwise: time to mergesort N elements = time to mergesort N/2 elements plus  
time to merge two arrays each N/2 elements.

Time to merge two arrays each N/2 elements is linear, i.e. N

Thus

(1) T(1) = 1

(2) T(N) = 2T(N/2) + N

Next solve this recurrence relation. First divide (2) by N:

(3) T(N) / N = T(N/2) / (N/2) + 1

N is a power of two, so

(4) T(N/2) / (N/2) = T(N/4) / (N/4) +1

(5) T(N/4) / (N/4) = T(N/8) / (N/8) +1

(6) T(N/8) / (N/8) = T(N/16) / (N/16) +1

(7) ……  
(8) T(2) / 2 = T(1) / 1 + 1

Then add equations (3) through (8) : the sum of their left-hand sides   
will be equal to the sum of their right-hand sides:

T(N) / N + T(N/2) / (N/2) + T(N/4) / (N/4) + … + T(2)/2 =

T(N/2) / (N/2) + T(N/4) / (N/4) + ….+ T(2) / 2 + T(1) / 1 + LogN

(LogN is the sum of 1s in the right-hand sides)

After crossing the equal term, we get

(9) T(N)/N = T(1)/1 + LogN

T(1) is 1, hence it obtain

(10) T(N) = N + NlogN = O(NlogN)

Hence the complexity of the Merge Sort algorithm is O(NlogN).

**DESCRIPTION ABOUT MERGE SORT**

Merge sort is very predictable. It makes between 0.5lg(n) and lg(n) comparisons per element, and between lg(n) and 1.5lg(n) swaps per element. The minima are achieved for already sorted data; the maxima are achieved, on average, for random data. If using Θ(n) extra space is of no concern, then merge sort is an excellent choice: It is simple to implement, and it is the only stable O(n·lg(n)) sorting algorithm. Note that when sorting linked lists, merge sort requires only Θ(lg(n)) extra space (for recursion).Merge sort is the algorithm of choice for a variety of situations: when stability is required, when sorting linked lists, and when random access is much more expensive than sequential access (for example, external sorting on tape).There do exist linear time in-place merge algorithms for the last step of the algorithm, but they are both expensive and complex. The complexity is justified for applications such as external sorting when Θ(n) extra space is not available.

**PROPERTIES OF MERGE SORT**

* Stable
* Θ(n) extra space for arrays (as shown)
* Θ(lg(n)) extra space for linked lists
* Θ(n·lg(n)) time
* Not adaptive
* Does not require random access to data

#### SIGNIFICANCE OF MERGESORT

Thus the significance of MergeSort is that:

* its worst case time is O(n\*log n).
* it is stable

The only possible down side of MergeSort is that it requires a spare array in order to do the merge operations.

#### EFFICIENT USAGE OF MERGE TARGET

Our first inefficient approach is think of B as always being the merge target.

1. If the size of A is 1, copy A into B.
2. If the size is greater than 1, call mergeSort recursively to sort left half of A, A1, into the left half of B, B1. Same with the right half A2 into B2.
3. Copy B into A. A now consists of two sorted halves.
4. Merge the sorted halves (A1,A2) into B.

At the completion of the algorithm,It is necessary to copy B back to A.

An efficient implementation of this algorithm does fewer data movements by having A and B switch "roles" where one is the merge source and the other the merge target. In this way we completely avoid the copy step (b).

In any case, we want the final merge target to be A. As we go into the recursion, the merge target alternates:

A ← B ← A ← B ← ... ← (base of recursion: A or B)

This requirement means that when we get to the base case with arrays of size 1, we require that

* if B is the merge target: copy (the single element) from A to B.
* if A is the merge target: do nothing, the element is already in A.

**Advantages:**  
1.Conceptuallysimple  
2. It is suited to sorting linked lists of elements because merge traverses each linked list  
3. It is suited to sorting external files; divides data into smaller files until can be stored in arrayinmemory  
4. Stable performance

**Use of merge sort**

1. **Fast no matter the input**

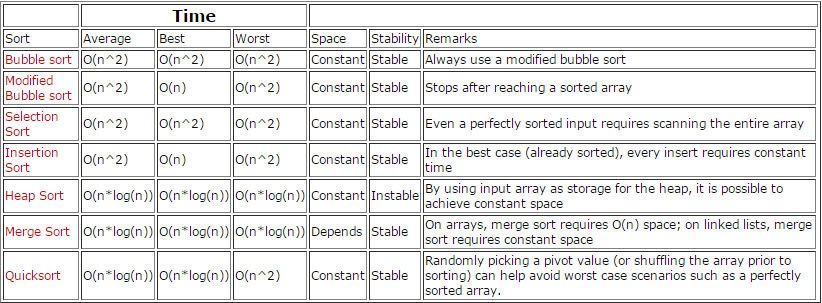
Merge sort is a great sorting algorithm mainly because it’s very fast and stable. It’s complexity is the same even in the worst case and it is O(n\*log(n)). Note that even quicksort’s complexity is O(n2) in the worst case, which for n = 20 is about 4.6 times slower!Merge sort is about 4.6 times faster than quicksort for n = 20!

1. **Easy implementation**

Another cool reason is that merge sort is easy to implement. Indeed most of the developer consider something fast to be difficult to implement, but that’s not the case of merge sort.

It is very important to know the input data. Indeed if the input is nearly sorted the insertion sort or bubble sort can be faster. Note that in the best case insertion and bubble sort complexity is o(n), while merge sort’s best case is o(n\*log(n)).

**Comparison of all sorting algorithms**

**Conclusion**

As a conclusion merge sort is practically one of the best sorting algorithms because it’s easy to implement and fast, so it must be considered by every developer!

**STUDENT ACTIVITY OPTIMIZATION**

**TOPOLOGICAL SORT**

**TEAM MEMBERS**

SHEREEN FATHIMA. A (140071601068)

STELLA. D (140071601073)

**PROBLEM IDENTIFICATION:**

This scenario is based on an average student’s day to day activities. Every student has his or her own activities to be carried out on a daily basis, right from studying to taking rest and allotting time to have food and other such activities. The day to day activities of the student can be carried out in any order depending on the convenience of the student and also depending on the time allocation, but in order to manage the time efficiently while the activities are carried out, the activities must first be sorted out in a systematic order. The aim of the student would be to both carry out all the activities in his or her schedule as well as make the most efficient use of time and resources. The major problem now is the methodology of ordering the random day to day activities in a systematic order for better time management and maximum efficiency exhibited while carrying out the activities.

**ALGORITHM USED:**

The algorithm used in the systematic arrangement of the day to day student activities is Topological Sort. A topological sort or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge *u v* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering, such ordering is called as Topological ordering. Topological Sorting is mainly used for scheduling jobs from the given dependencies amongst the given jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recompiling formula values in spreadsheets, logic synthesis, determining the order of compilation tasks. So, to order a student’s day to day activities in a systematic and in an efficient manner, the Topological Sort is the best algorithm to be implemented because it gives an accurate outlook on the problem and the solution to the proposed problem can be easily sorted out by implementing Topological Sorting algorithm. Hence, to sort the day to day activities of a student in a systematic order, Topological Sort is an efficient way to carry out the ordering, thus providing an accurate and apt solution to the proposed problem.

**ALGORITHM:**

Topological\_Sort (incoming edges, outgoing edges)

queue Q = 0; // Initialize queue as 0

Topologically\_sorted\_list T = 0; // Topological vertex list initially 0

if inDegree[i] = 0; // if condition to check indegree

then Push[i] = Q; // if true, return value to queue Q

While Q != empty; // while condition to check if queue is not empty

Dequeue Q into u; // Dequeue u into queue Q

Push u in T; // Push u in the Topolocically\_sorted\_list

Remove all outgoing edges from u; // Remove the OutDegree edges from vertex u

Return T; // Returns the value of T

**2. PROBLEM**

**PROBLEM DEFINITION:**

The proposed problem is the methodology of ordering of the random day to day activities of the student to carry out all the tasks in his or her schedule in a systematic order for better time management and maximum efficiency which is to be exhibited while carrying out the activities.

**PROBLEM DESCRIPTION:**

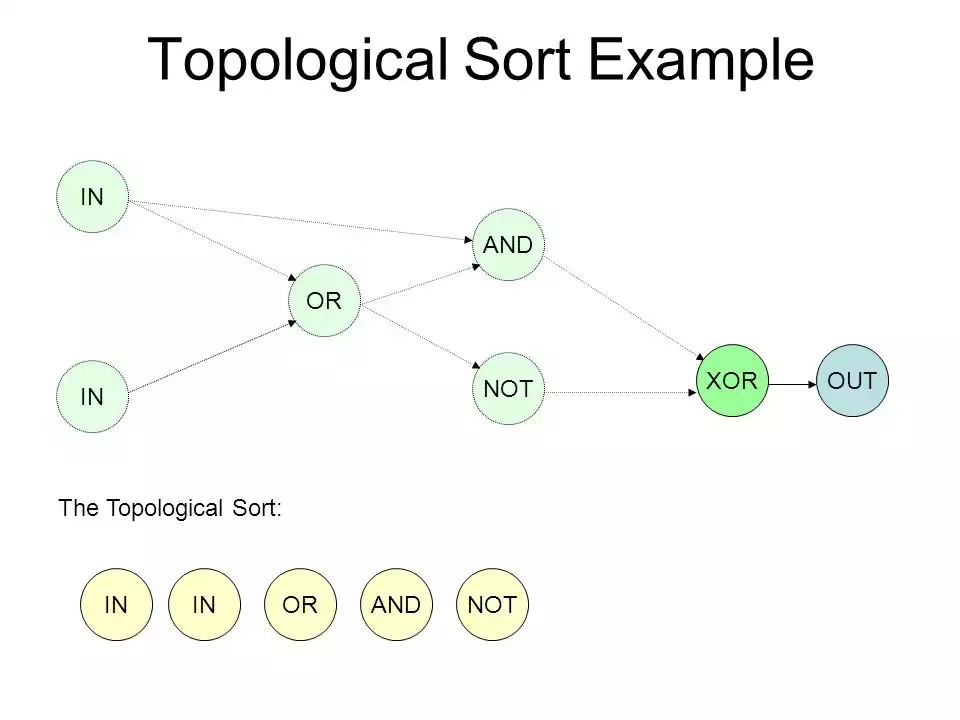
A student has various activities like studying, dining and other such recreational activities which are to be carried out on a daily basis for efficient functioning of the student, both academically and outdoors. Every student has his or her own activities to be carried out on a daily basis, right from studying to taking rest and allotting time to have food and other such activities. The day to day activities of the student can be carried out in any order depending on the convenience of the student and also depending on the time allocation, but in order to manage the time efficiently while the activities are carried out, the activities must first be sorted out in a systematic order. The aim of the student would be to both carry out all the activities in his or her schedule as well as make the most efficient use of time and resources. The major problem now is the methodology of ordering the random day to day activities in a systematic order for better time management and maximum efficiency exhibited while carrying out the activities. To conclude with the description of the problem, the activities are to be sorted out in a systematic, planned order for thee optimized use of time and well planned activities that support it.

**BACKGROUND OF THE PROBLEM:**

These are the various student activities that are carried out on a daily basis that range from the initial activity, that is from waking up, till attending college till sleeping, completing a whole cycle of events that are to be covered in a day. The major aim of the problem is to properly order the set activities of the student. The day to day activities of an average student is represented below in an un orderly manner, which can be implemented in any manner as per the likes and suitability of the student, but for efficient use of the time, following the order of the set activities after the topological ordering is implemented on the activities helps in optimized and planned usage of time.

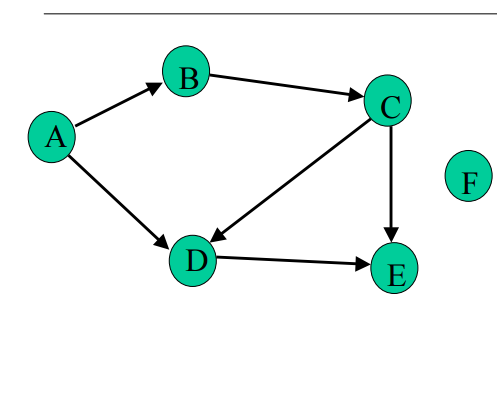
In order to arrange a student’s day to day activities in a systematic and in an efficient manner, the Topological Sort is the best algorithm to be implemented because it gives an accurate outlook on the problem and the solution to the proposed problem can be easily sorted out by implementing Topological Sorting algorithm. Hence, to sort the day to day activities of a student in a systematic order, Topological Sort is an efficient way to carry out the ordering, thus providing an accurate and apt solution to the proposed problem.

**GENERALIZED TOPOLOGICAL SORT EXAMPLE**

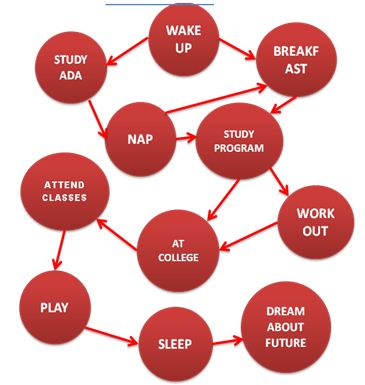


TOPOLOGICAL SORT: STUDENT ACTIVITIES

The above graph represents the methodology to be followed while ordering nodes using Topological Sort. The below graphical representation shows the day to day activities of a student that has to be performed, but in order to perform the daily activities in an order that optimizes the time management, the activities must be ordered first, randomly. Based on the level of indegree and outdegree to a node, the activities of the student can be rated in a manner that optimizes time spent on each and enhances the time spent on each activity.



TOPOLOGICAL SORT: STUDENT ACTIVITIES



*Fig 2.1 Student day to day activities*

**AFTER IMPLEMENTING TOPOLOGICAL SORT:**

The indegrees of all vertices is computed. Then a vertex U with indegree 0 and print it (store it in the ordering), in the queue. If there is no such vertex then there is a cycle and the vertices cannot be ordered. Stop. Revert back. Remove U and all its edges (U,V) from the graph. Update the indegrees of the remaining vertices. Repeat the steps till there are no nodes left and the queue is complete.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***1*** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

*Fig 2.2 Initial queue*

**3. ALGORITHM DESIGN**

**Procedure:**

**Step 1:**Compute the indegrees of all vertices

**Step 2:** Find a vertex U with indegree 0 and print it (store it in the ordering)

**Step 3:**If there is no such vertex then there is a cycle and the vertices cannot be ordered. Stop.

**Step 4:** Remove U and all its edges (U,V) from the graph.

**Step 5:** Update the indegrees of the remaining vertices.

**Step 6:** Repeat steps 2 through 4 while there are vertices to be processed.

**Sample graph:** A graph which has five nodes and now the queue is empty

C

E

D

B

A

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

**Step 1:**According to the algorithm,compute all the indegrees of vertices

Indegree(A)=0

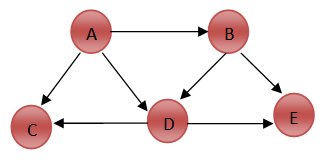
Indegree(B)=1

Indegree(C)=2

Indegree(D)=2

Indegree(E)=2

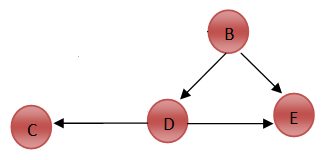
**Step 2:**Find a vertex with input degree(indegree) is 0.The vertex A has the indegree 0.



A

**Step 3:** Remove the vertex a (u) from the graph and insert in to the queue. Then remove all edges (u,v) from the graph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A |  |  |  |  |



**Step 4:**Again compute the degrees of all vertices

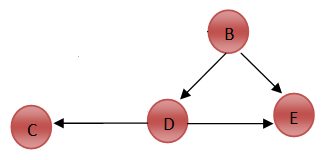
Indegree(B)=0

Indegree(C)=1

Indegree (D) =1

Indegree (E) =2

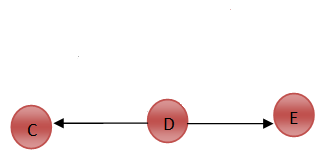
**Step 5:** Find a vertex with input degree (indegree) is 0.The vertex B has the indegree 0.



B

**Step 6:** Remove the vertex B (u) from the graph and insert in to the queue. Then remove all edges (ump) from the graph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B |  |  |  |



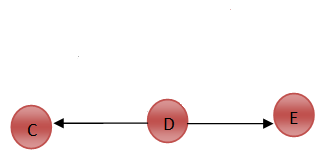
**Step 7**: Again compute the degrees of all vertices

Indegree(C) =1

Indegree (D) =0

Indegree (E) =1

**Step 8**: Find a vertex with input degree (indegree) is 0.The vertex D has the indegree 0.



D

**Step 9:** Remove the vertex D (u) from the graph and insert in to the queue. Then remove all edges (u,v) from the graph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | D |  |  |



**Step 10:**Again compute the degrees of all vertices

Indegree(C) =0

Indegree (E) =0

**Step 11:** Find a vertex with input degree (indegree) is 0.The vertex C and E has the indegree 0.Vetex can be removed in any order. It can be C, E or E, C.Finally; it should be inserted in the same order in queue in which they are removed from the graph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | D | C | E |

**Step 12:** The topological order will be

`

E

C

D

A

B

**Application of topological sort**

1. Scheduling jobs

2. Graph coloring

3. Graph connectivity

4. Network routing

5. Driving directions

6. Cheap flight ticket

**4.ALGORITHM ANAYSIS**

**Complexity of an algorithm**

Complexity of an algorithm is a measure of the amount of time and/or space required by an algorithm for an input of a given size (n).

**Effects of Run time of an algorithm**

1. Computer used, the hardware platform
2. Representation of abstract data types (Adt's)
3. Efficiency of compiler
4. Competence of implementer (programming skills)
5. Complexity of underlying algorithm
6. Size of the input

**Worst Case**

This time is the maximum run time, over all inputs of size n, ignoring effects (a) through (d) above. That is, we only consider the "number of times the principle activity of that algorithm is performed".

**Best Case**

The Run time a sorting algorithm is good only if the input to it is already sorted.

**Average Case**

Average case is the most useful measure. It might be the case that worst case behaviour is pathological and extremely rare, and that we are more concerned about how the algorithm runs in the general case.

**Time complexity**

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input. The time complexity of an algorithm is commonly expressed using a big O notation, which excludes coefficients and low order terms.

The usual algorithms for topological sorting have running time is linear in the number of nodes plus the number of edges, asymptotically, O (V + E).The vertices having indegree 0 should be inserted in to the queue one by one .Therefore it takes running time is O (V + E). The other form of storing the vertices is using indegree array.But its running time is worse time.Its quadratic time O (V2 + E).

**Space complexity** Space complexity is a measure of the amount of working storage an algorithm needs. That means how much memory, in the worstcase is needed at any point in the algorithm.

The memory is required for the following purposes...

* Memory required to store program instructions
* Memory required to store constant values
* Memory required to store variable values
* And for few other things

Generally, when a program is under execution it uses the computer memory for three reasons. They are as follows.

1. **Instruction Space:** It is the amount of memory used to store compiled version of instructions.

2. **Environmental Stack:** It is the amount of memory used to store information of partially executed functions at the time of function call.

3. **Data Space:** It is the amount of memory used to store all the variables and constants.

The algorithm used in the above topological ordering requires space equal to the number of vertices in the graph.

**Space complexity** = O(|v|) where v is the number of vertices

**5. Implementation**

The java program for implementing the topological ordering of vertices is achieved by using a software development platform (i.e.) NetBeans

**NetBeans**

NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans platform,including the NetBeans integrated development environment(IDE), can be extended by third party developers.The NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML5.NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM.

**NetBeans PlatForm**

The NetBeans Platform is a framework for simplifyingthe development of Java Swing desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platformbased applications; no additional SDK is required.Applications can install modules dynamically. Any applicationcan include the Update Center module to allowusers of the application to download digitally signed upgradesand new features directly into the running application.Reinstalling an upgrade or a new release does notforce users to download the entire application again.The platform offers reusable services common to desktopapplications, allowing developers to focus on the logicspecific to their application. Among the features of the platform are:

1. User interface management (e.g. menus and toolbars)
2. User settings management
3. Storage management (saving and loading any kind
4. of data)
5. Window management
6. Wizard framework (supports step-by-step dialogs)
7. NetBeans Visual Library
8. Integrated development tools

**NetBeans IDE**

NetBeans IDE is an open-source integrated development environment. NetBeans IDE supports development of all Java application types (Java SE (including JavaFX), JavaME, web, EJB and mobile applications) out of the box. Among other features are an Ant-based project system, Maven support, refactorings, version control (supporting CVS, Subversion, Git, Mercurial and Clearcase).

**Modularity:** All the functions of the IDE are provided by modules. Each module provides a well-defined function, such as support for the Java language, editing, or support for the CVS versioning system, and SVN. NetBeans contains all the modules needed for Java development in a single download, allowing the user to start working immediately. Modules also allow NetBeans to be extended. New features, such as support for other programming languages, can be added by installing additional modules. For instance, Sun Studio, Sun Java Studio Enterprise, and Sun Java Studio Creator from Sun Microsystems are all based on the NetBeans IDE.

**Java Programming**

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation, dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA),meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture.

**Feautures of java**

1. Java is “simple, object-oriented, and familiar”.

2. Java is “robust and secure”.

3. Java is “architecture-neutral and portable”.

4. Java will be execute with “high performance”.

5. Java is “interpreted, threaded, and dynamic”.

**Java Program**

package topologicalsort;

import java.util.InputMismatchException;

import java.util.Scanner;

import java.util.Stack;

public class TopologicalSort

{

private Stack<Integer> stack;

public TopologicalSort()

{

stack = new Stack<Integer>();

}

public int [] topological(int adjacency\_matrix[][], int source) throws NullPointerException

{

int number\_of\_nodes = adjacency\_matrix[source].length - 1;

int[] topological\_sort = new int [number\_of\_nodes + 1];

int pos = 1;

int j ;

int visited[] = new int[number\_of\_nodes + 1];

int element = source;

int i = source;

visited[source] = 1;

stack.push(source);

while (!stack.isEmpty())

{

element = stack.peek();

while (i <= number\_of\_nodes)

{ if (adjacency\_matrix[element][i] == 1 && visited[i] == 1)

{

if (stack.contains(i))

{

System.out.println("TOPOLOGICAL SORT NOT POSSIBLE");

return null;

}

}

if (adjacency\_matrix[element][i] == 1 && visited[i] == 0)

{

stack.push(i);

visited[i] = 1;

element = i;

i = 1;

continue;

}

i++;

}

j = stack.pop();

topological\_sort[pos++] = j;

i = ++j;

}

return topological\_sort;

}

public static void main(String...arg)

{

int number\_no\_nodes, source;

Scanner scanner = null;

int topological\_sort[] = null;

try

{

System.out.println("Enter the number of nodes in the graph");

scanner = new Scanner(System.in);

number\_no\_nodes = scanner.nextInt();

int adjacency\_matrix[][] = new int[number\_no\_nodes + 1][number\_no\_nodes + 1];

System.out.println("Enter the adjacency matrix");

for (int i = 1; i <= number\_no\_nodes; i++)

for (int j = 1; j <= number\_no\_nodes; j++)

adjacency\_matrix[i][j] = scanner.nextInt();

System.out.println("Enter the source for the graph");

source = scanner.nextInt();

System.out.println("The Topological sort for the graph is given by ");

TopologicalSort toposort = new TopologicalSort();

topological\_sort = toposort.topological(adjacency\_matrix, source);

System.out.println();

for (int i = topological\_sort.length - 1; i > 0; i-- )

{

if (topological\_sort[i] != 0)

System.out.print(topological\_sort[i]+"\t");

}

}catch(InputMismatchException inputMismatch)

{

System.out.println("Wrong Input format");

}catch(NullPointerException nullPointer)

{

}

scanner.close();

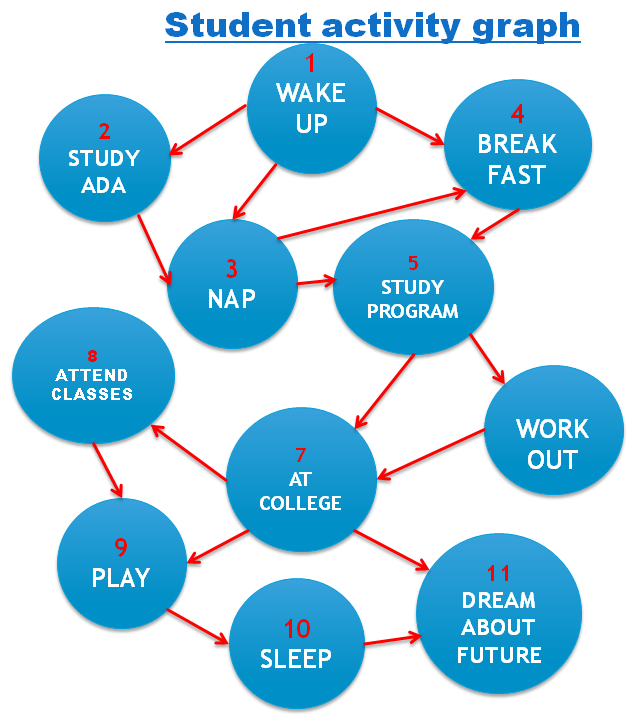
}

}

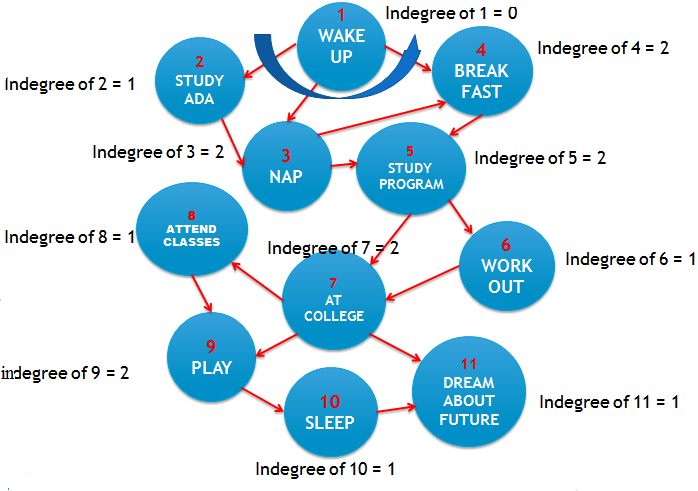
**Implementation of scenario**

****

**Step 1:**First number the vertices

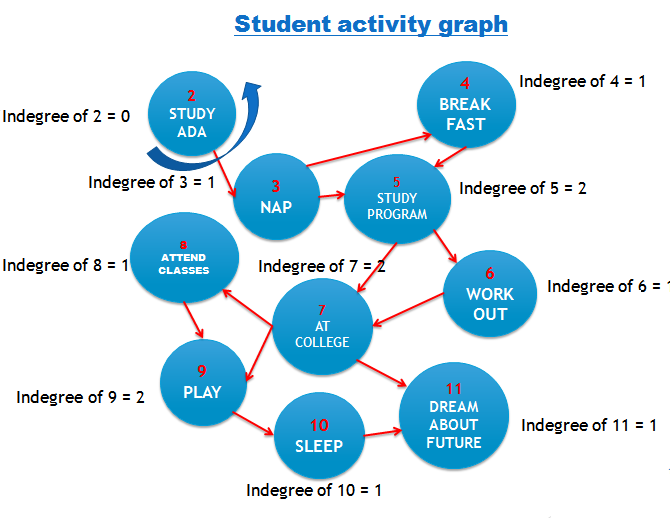
****

**Step 2:**According to the algorithm,compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

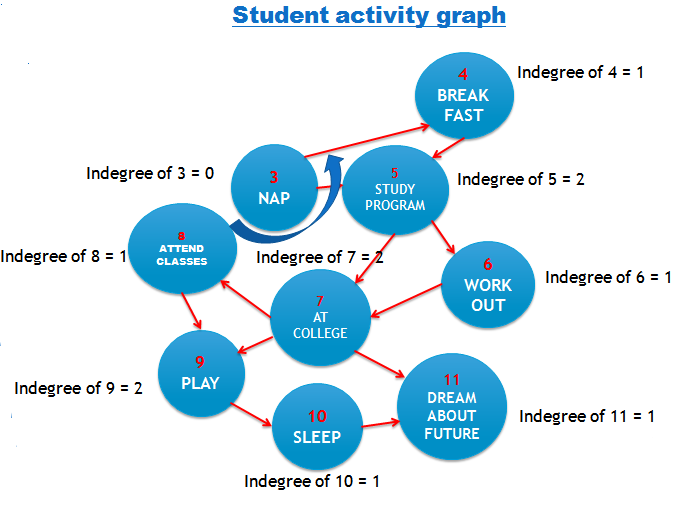
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** |  |  |  |  |  |  |  |  |  |  |

**Step 3:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

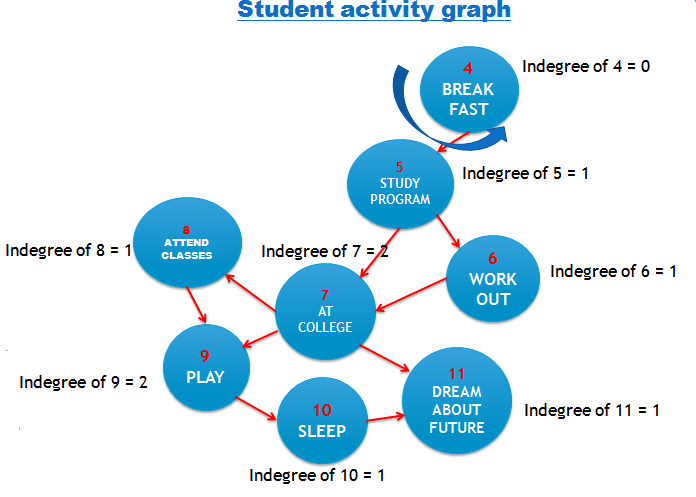
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** |  |  |  |  |  |  |  |  |  |

**Step 4:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

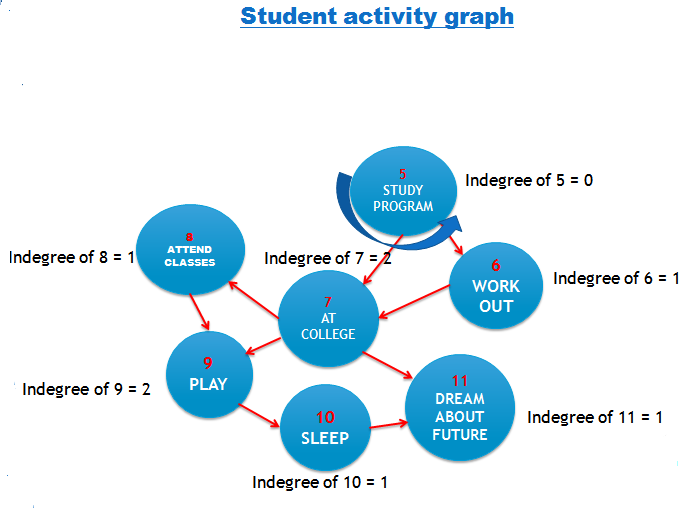
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** |  |  |  |  |  |  |  |  |

**Step 5:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

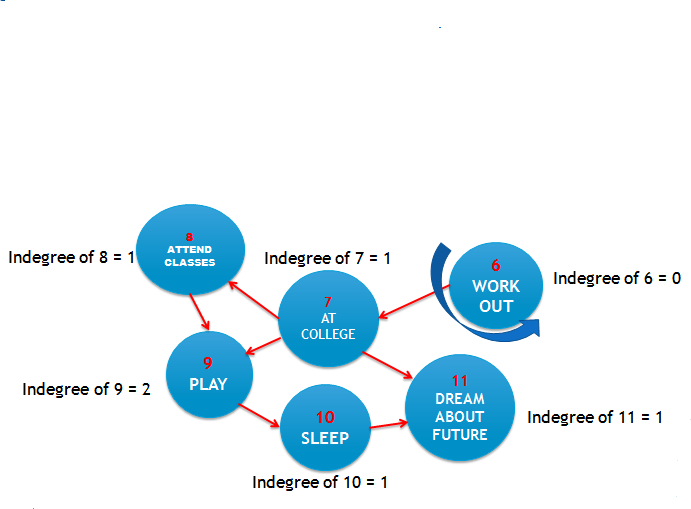
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |  |  |  |  |  |  |  |

**Step 6:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

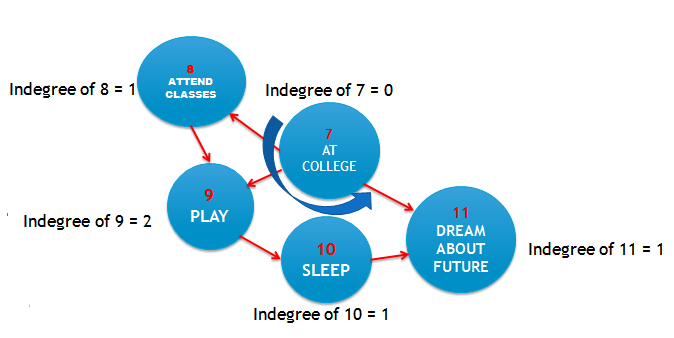
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |  |  |  |  |  |  |

**Step 7:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

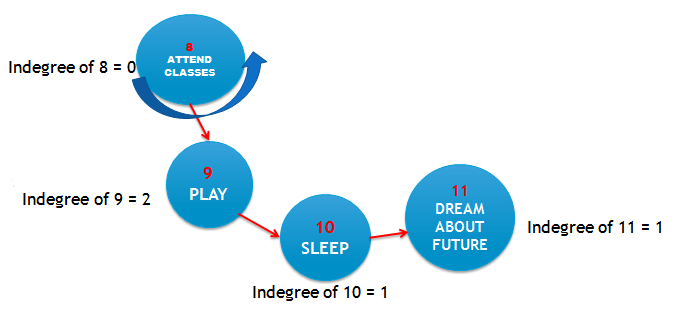
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** |  |  |  |  |  |

**Step 8:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

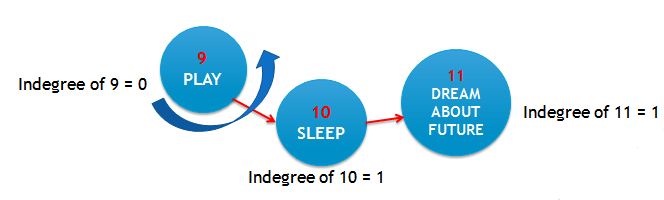
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** |  |  |  |  |

**Step 9:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

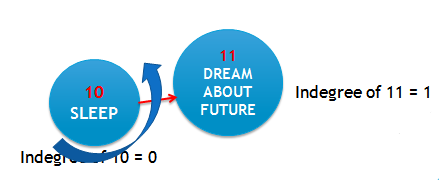
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |  |  |  |

**Step 10:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

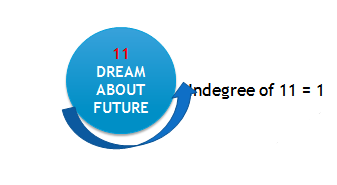
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |  |  |

**Step 11:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |  |

**Step 12:**Again compute the indegree of all vertices.Then remove the vertex and its edges which has indegree 0.Insert the removed vertex in to queue.

****

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |

**Hence the Topological order will be**

11111ggh

1**0**ponnjnn00000

9

8

4

3

2

1

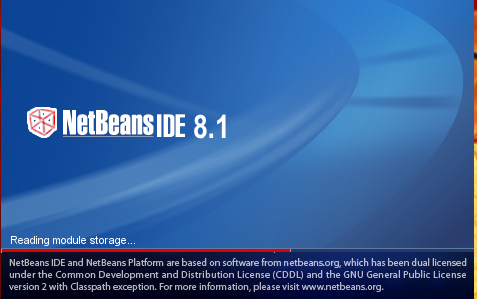
7

6

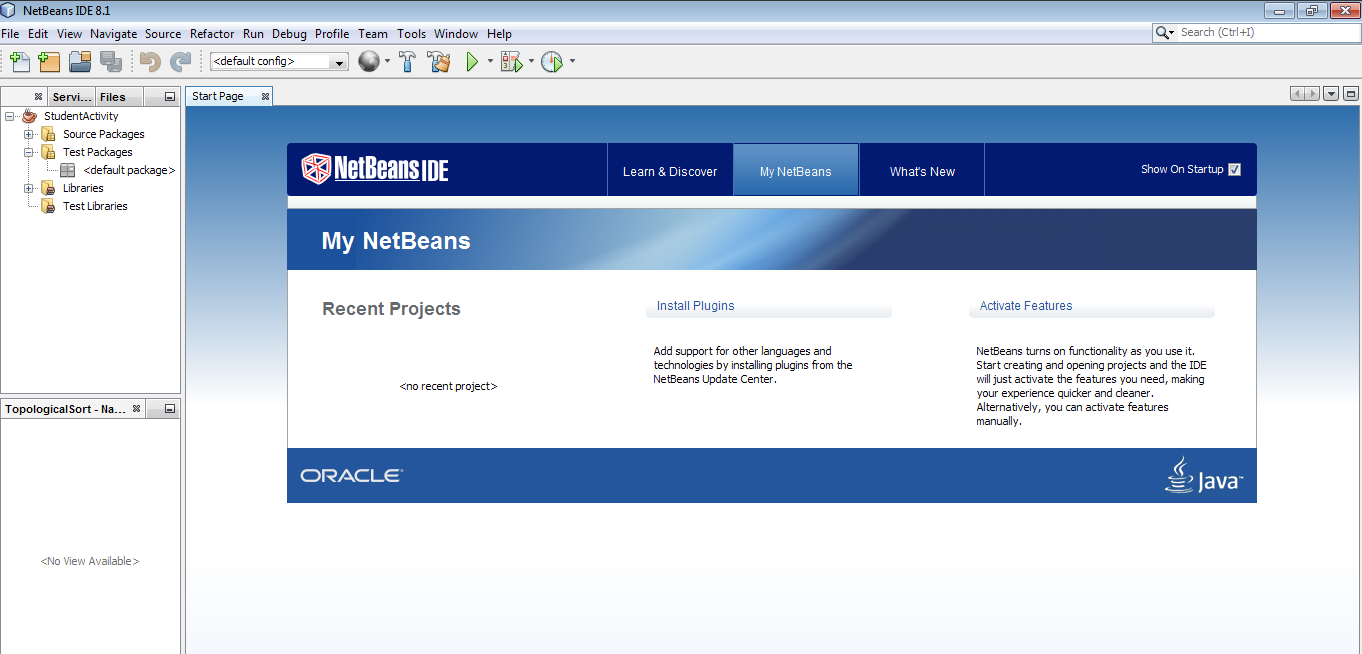
5

**6.OUTPUT SCREENSHOTS**

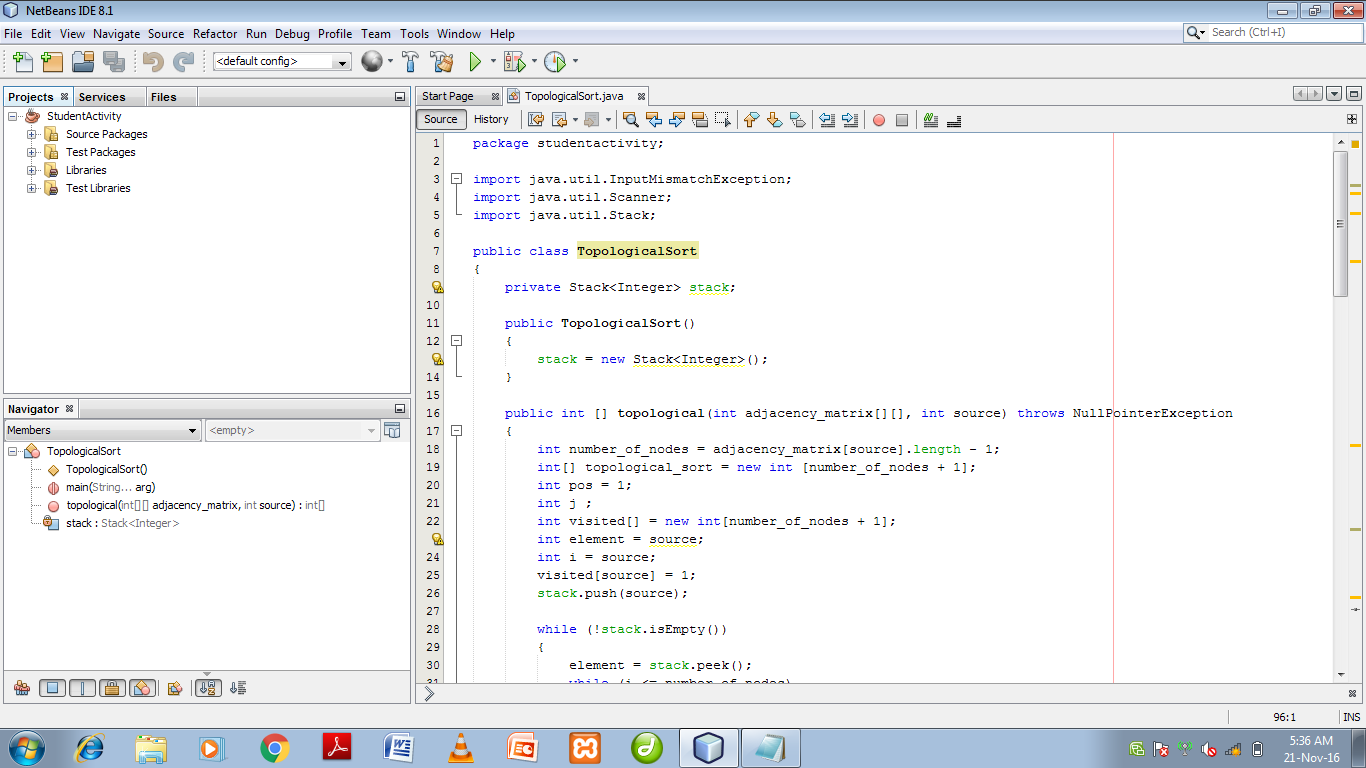
**Step 1**:Open the NetBeans IDE to run the java program



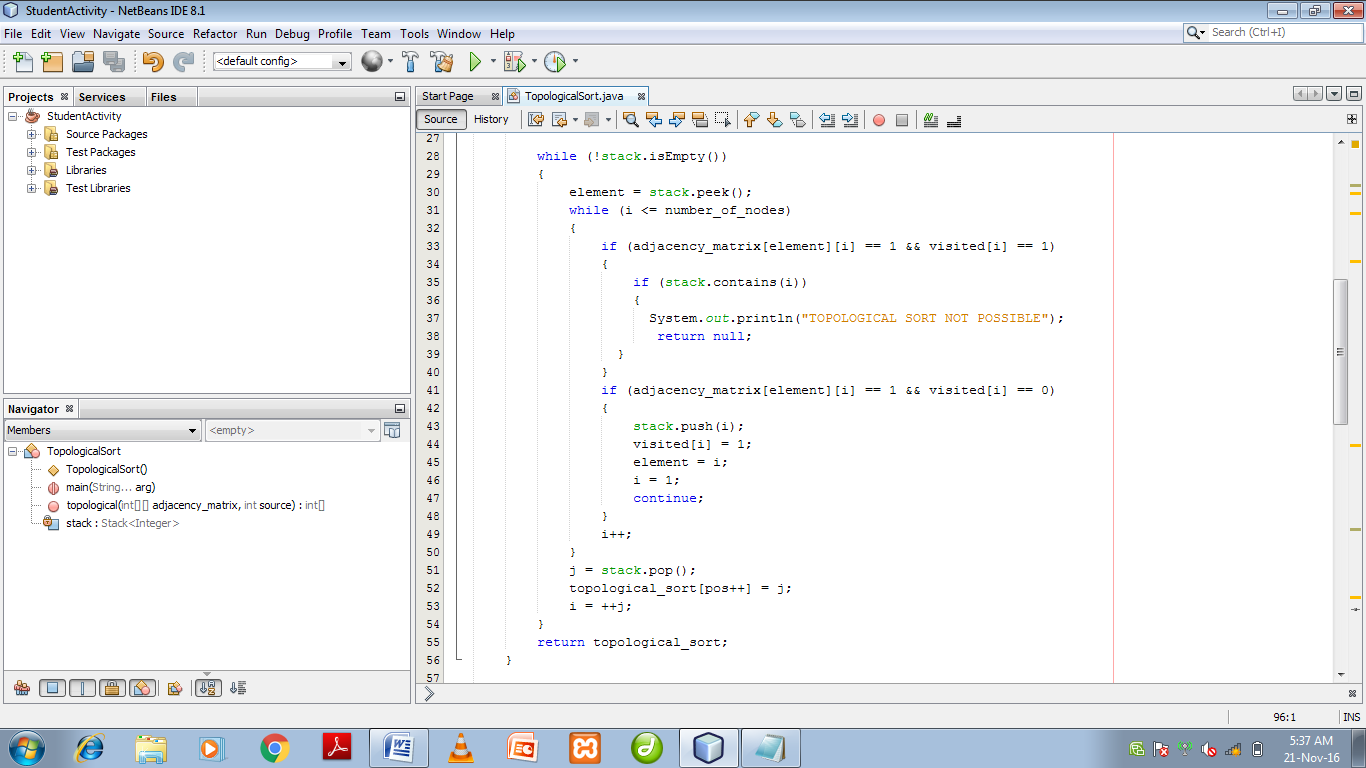
**Step 2:**Create a new java project

****

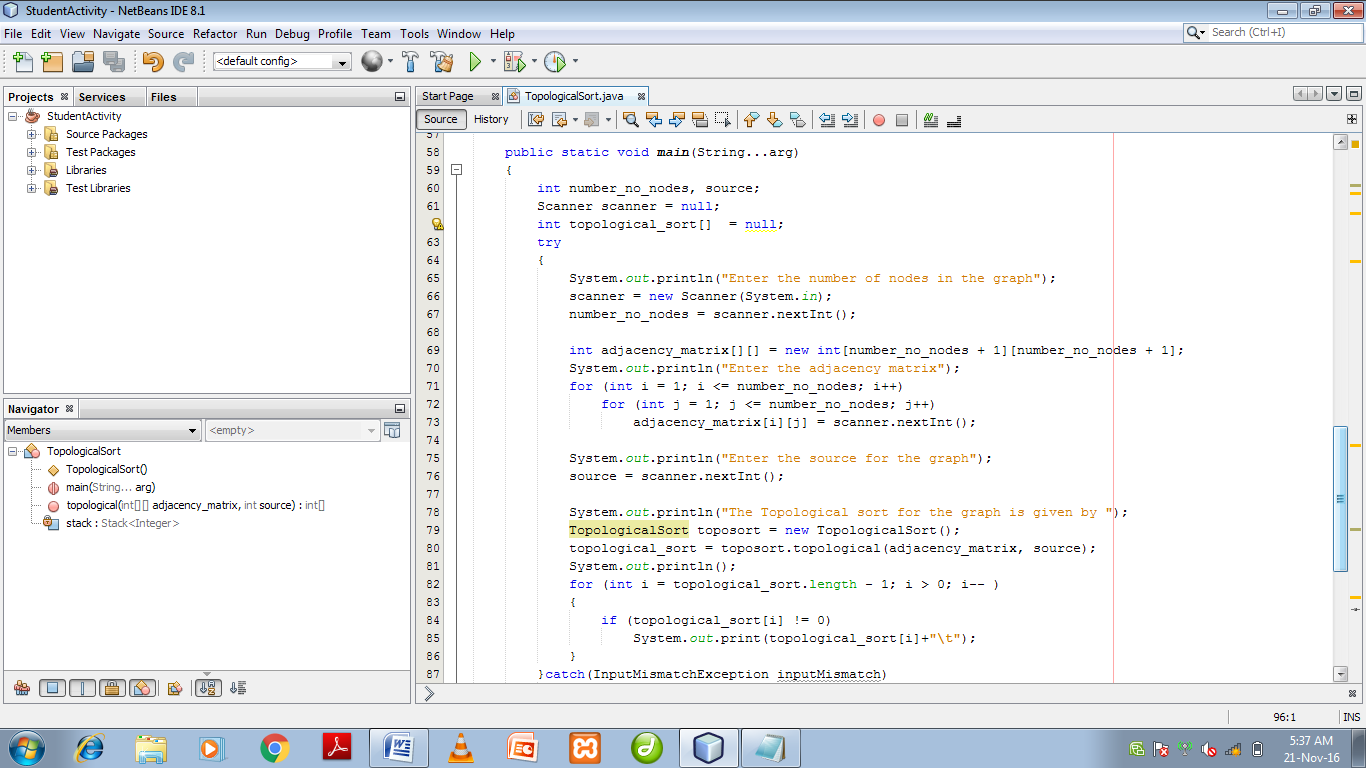
**Step 3:**Type the program for topological ordering of vertices.Import necessary packages for the java program.First intialize the adjacency\_matrix and also create a new array to store the number of vertices.



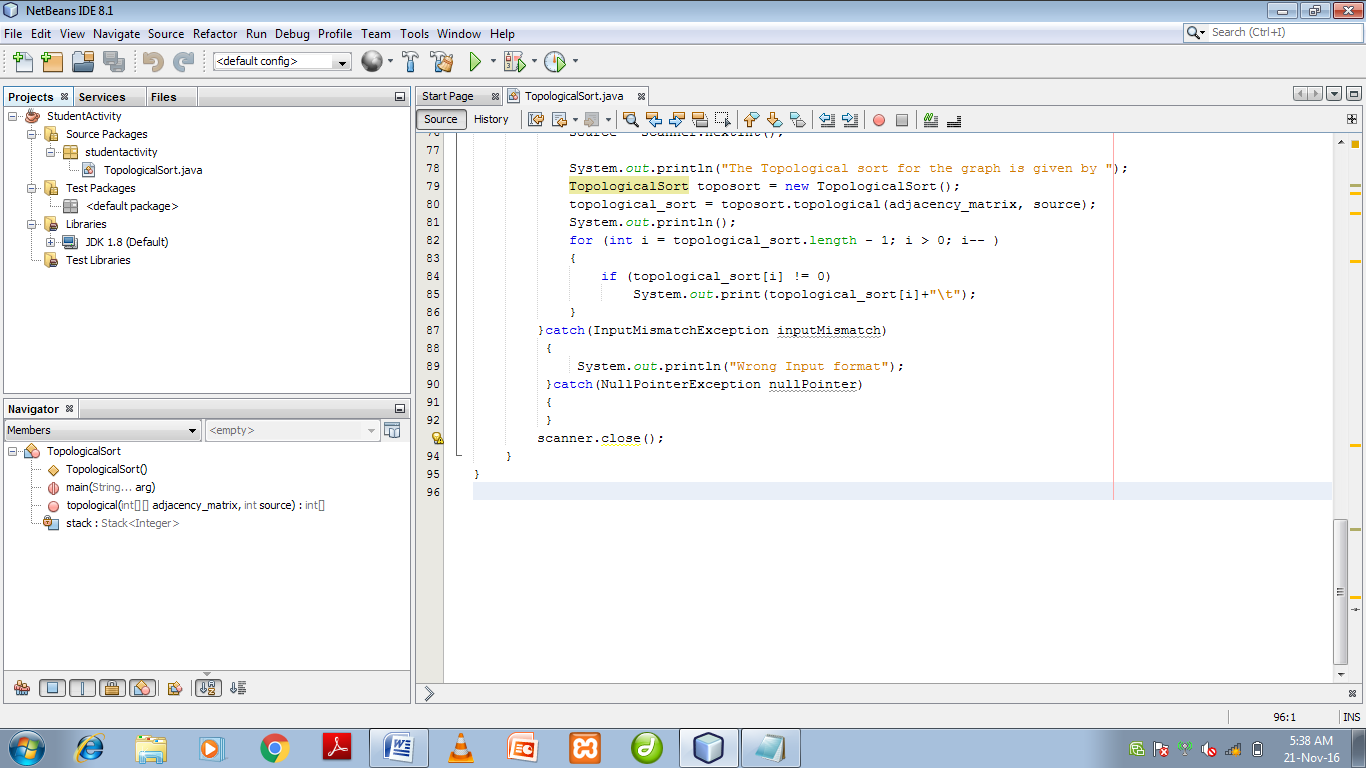
**Step 4:**Create a while loop to check whether the graph forms a cycle. If it forms a cycle then it shows the message “Topological sort is not possible”



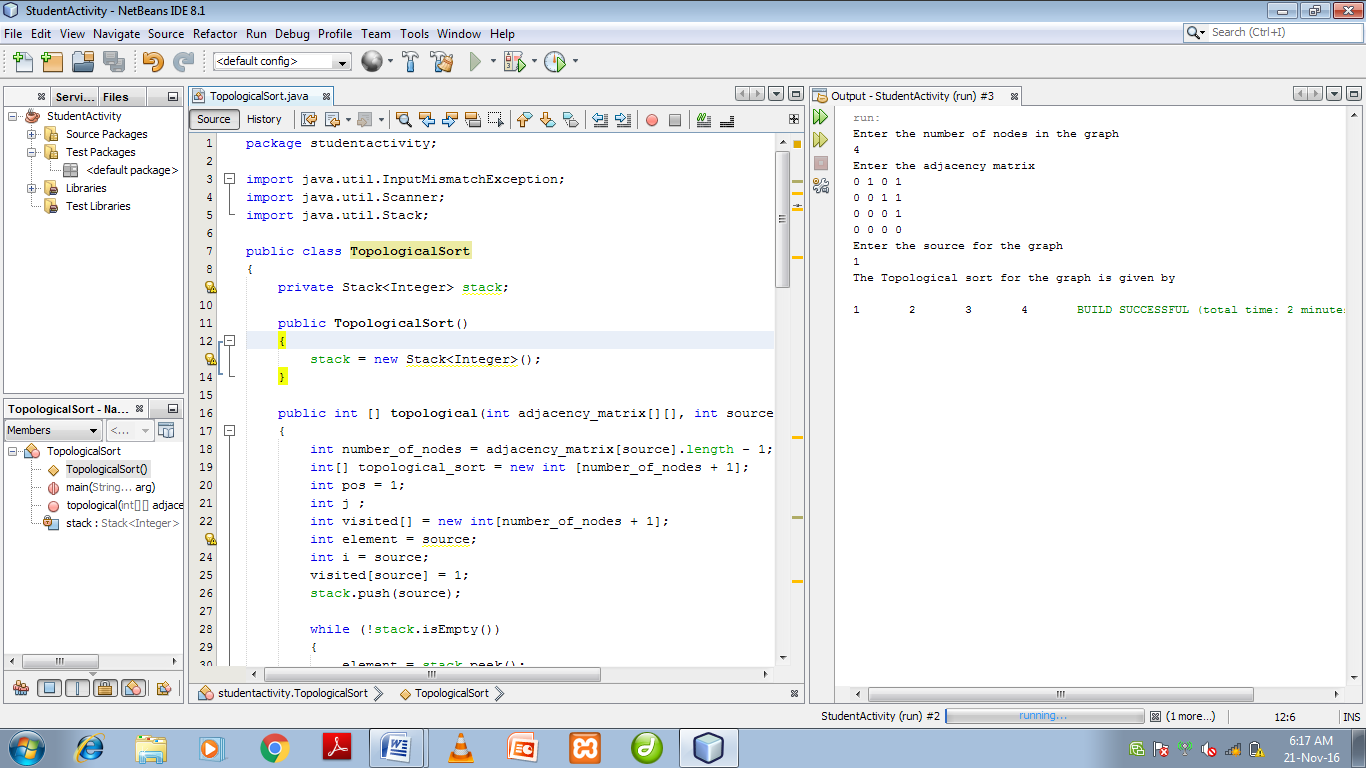
**Step 5:** Get input from the user



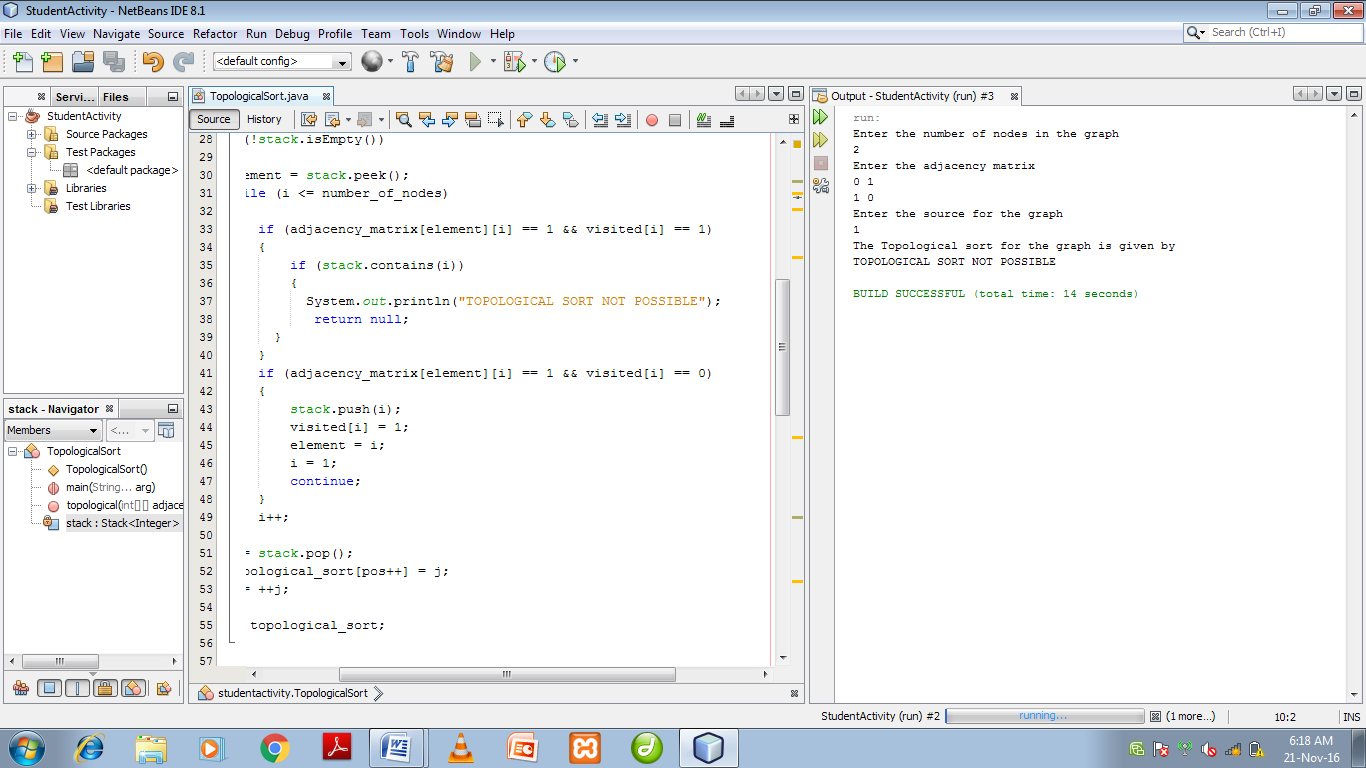
**Step 6**:To print the topological ordering or to display some error message.



**Step 7**: Run the file to see the correct output by giving the correct inputs.



**Step 8**: Run the file to see the incorrect output by giving the incorrect inputs.



**CHAPTER 7 – CONCLUSION**

The major problem faced while managing the time for the student activities was the exact arrangement of activities in an optimized and in an efficient order. This problem was overcome using Topological Sort. By making use of the topological ordering algorithm, the student activities were managed accordingly and optimized, which was beneficent to the student. Hence, using the topological sorting algorithm, the aim of the project was achieved successfully, the student activities were organized in an efficient manner.

The aim of the project has been successfully achieved. The objective of this project was to take in the day to day activities of a student and arrange and order it in a manner which is academically helpful to the student, which would help the student in optimizing the time allotted to manage his activities and optimize it. Every student has his or her own activities to be carried out on a daily basis, right from studying to taking rest and allotting time to have food and other such activities. The day to day activities of the student can be carried out in any order depending on the convenience of the student and also depending on the time allocation, but in order to manage the time efficiently while the activities are carried out, the activities must first be sorted out in a systematic order. The aim of the student would be to both carry out all the activities in his or her schedule as well as make the most efficient use of time and resources, which has been successfully done. Hence, it’s concluded that, by making use of the topological ordering algorithm, the aim of the project was achieved successfully, using topological ordering algorithm.