**PROGRAM 1A**

**Create a Java class called Student with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone Write a Java program to create n Student objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.**

**Description :**

Student Details USN, name, branch and phone number are considered which are initialized through constructors and Details are printed using tostring() function. Student.java accepts the details from User and Student.java accepts these details from Student.java through constructor.

import java.io.\*;

import java.util.\*;

public class Student {

private String USN;

private String Name;

private String Branch;

private String Phone;

public String getUSN()

{

return USN;

}

public String getName()

{

return Name;

}

public String getBranch()

{

return Branch;

}

public String getPhone()

{

return Phone;

}

public Student(String usn,String name,String branch,String phone)

{

super();

USN=usn;

Name=name;

Branch=branch;

Phone=phone;

}

}

**Class Lp1A.java**

import java.io.\*;

import java.util.\*;

public class Lp1A {

public static void main(String[] args) throws IOException {

String usn,name,branch,phone;

Scanner in=new Scanner(System.in);

System.out.println("Enter no. of students");

int n=in.nextInt();

Student st[]=new Student[n];

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

{

System.out.println("\nEnter details "+(i+1));

System.out.println("USN");

usn=in.next();

System.out.println("Name");

name=in.next();

System.out.println("Branch");

branch=in.next();

System.out.println("Phone");

phone=in.next();

st[i]=new Student(usn,name,branch,phone);

}

System.out.println("Details are");

System.out.println("USN\tName\tBranch\tPhone");

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

{

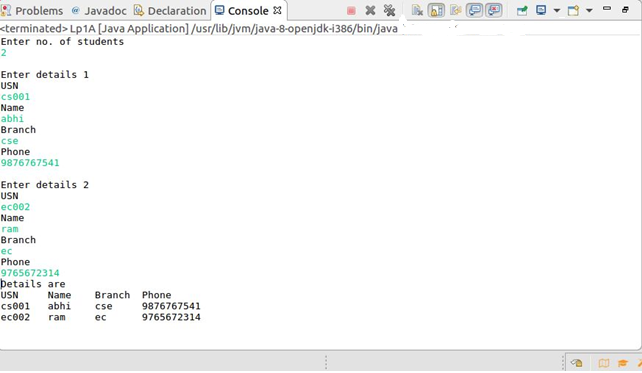
System.out.println(st[i].getUSN()+"\t"+st[i].getName()+"\t"+st[i].getBranch()+"\t"+st[i].getPhone());

}

}

}

OUTPUT:



**PROGRAM 1B**

**Write a Java program to implement the Stack using arrays. Write Push(),Pop(), and Display() methods to demonstrate its working.**

**Description:**

A **stack** is an [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) that serves as a [collection](https://en.wikipedia.org/wiki/Collection_(computing)) of elements, with two principal operations: *push*, which adds an element to the collection, and *pop*, which removes the most recently added element that was not yet removed. The order in which elements come off a stack gives rise to its alternative name, LIFO (for last in, first out). Additionally, a [*peek*](https://en.wikipedia.org/wiki/Peek_(data_type_operation)) operation may give access to the top without modifying the stack.

import java.lang.\*;

import java.util.\*;

public class Lp1B {

Scanner in=new Scanner(System.in);

int top=-1;

int a[]=new int[10];

int SMAX=3;

public void push()

{

int item;

if(top==(SMAX-1))

{

System.out.println("Overflow");

}

else

{

System.out.println("Enter element to be inserted");

item=in.nextInt();

top++;

a[top]=item;

}

}

void pop()

{

int item;

if(top==-1)

{

System.out.println("Underflow");

}

else

{

item=a[top];

top--;

System.out.println("popped element is "+item);

}

}

void display()

{

int i;

if(top==-1)

{

System.out.println("Empty");

}

else

{

System.out.println("Elements are");

for(i=top;i>=0;i--)

{

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

}

}

System.out.println();

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Lp1B s1=new Lp1B();

int ch=0;

Scanner in=new Scanner(System.in);

for(;;)

{

System.out.println("Stack");

System.out.println("1.Push");

System.out.println("2.Pop");

System.out.println("3.Display");

System.out.println("4.Exit");

System.out.println("Enter choice");

ch=in.nextInt();

switch(ch)

{

case 1: s1.push();

break;

case 2: s1.pop();

break;

case 3: s1.display();

break;

case 4: System.exit(0);

break;

default: System.out.println("Invalid choice");

break;

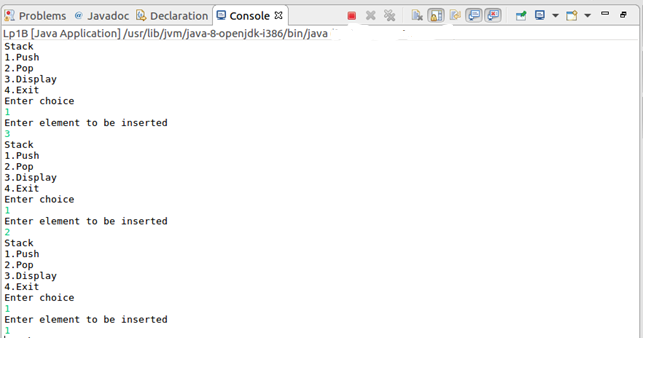
}

}

}

}

OUTPUT:





**PROGRAM 2A**

**Design a superclass called Staff with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely Teaching (domain, publications), Technical (skills), and Contract (period). Write a Java program to read and display at least 3 staff objects of all three categories**

**Description :**

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order. extends is the keyword used to inherit the properties of a class. Staff.java 4 members Staffid, name, salary and mobile number. A staff can be teaching staff or technical staff and Contract staff. Teaching.java staff will have Domain and publication as its members along with inherited members of base class Staff class.Ttechnical.java staff will have skills as its member along with inherited members of base class Staff class. A contract staff will have period as its member along with inherited members of base class Staff class. Whenever Staff details are not entered and tried to be displayed, StaffEmptyException is called.

import java.lang.\*;

import java.util.\*;

public class Lp1B {

Scanner in=new Scanner(System.in);

int top=-1;

int a[]=new int[10];

int SMAX=3;

public void push()

{

int item;

if(top==(SMAX-1))

{

System.out.println("Overflow");

}

else

{

System.out.println("Enter element to be inserted");

item=in.nextInt();

top++;

a[top]=item;

}

}

void pop()

{

int item;

if(top==-1)

{

System.out.println("Underflow");

}

else

{

item=a[top];

top--;

System.out.println("popped element is "+item);

}

}

void display()

{

int i;

if(top==-1)

{

System.out.println("Empty");

}

else

{

System.out.println("Elements are");

for(i=top;i>=0;i--)

{

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

}

}

System.out.println();

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Lp1B s1=new Lp1B();

int ch=0;

Scanner in=new Scanner(System.in);

for(;;)

{

System.out.println("Stack");

System.out.println("1.Push");

System.out.println("2.Pop");

System.out.println("3.Display");

System.out.println("4.Exit");

System.out.println("Enter choice");

ch=in.nextInt();

switch(ch)

{

case 1: s1.push();

break;

case 2: s1.pop();

break;

case 3: s1.display();

break;

case 4: System.exit(0);

break;

default: System.out.println("Invalid choice");

break;

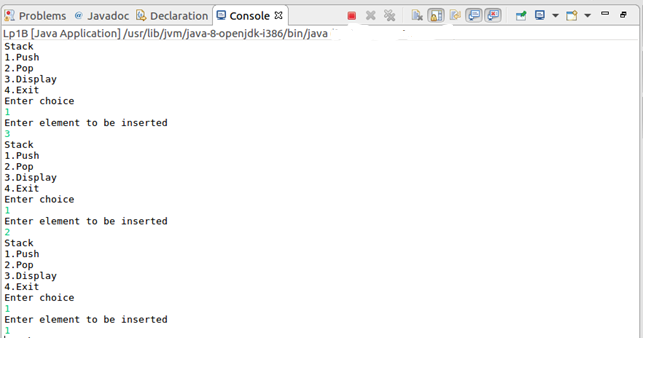
}

}

}

}

OUTPUT:





**PROGRAM 2B**

**Write a Java class called Customer to store their name and date\_of\_birth. The date\_of\_birth format should be dd/mm/yyyy. Write methods to read customer data as and display as using StringTokenizer class considering the delimiter character as “/”.**

**Description :**

Customer name and Date (dd/mm/yyyy) are accepted and displayed as name, date (dd,mm,yyyy).The java.util.stringTokenizer class allows an application to break a string into tokens. StringTokenizer(String str, String delimiter) is a constructor which constructs string tokenizer for the specified string. Date is the string and delimiter / is used in Customer.java to implement the same. String Tokenizer tokenizer= new StringTokenizer (date,"/"); keeps breaking the string date into tokens separated by delimiter / , until it find new tokens using hasMoreElements function.

import java.util.\*;

public class Customer {

private String Name;

private String DOB;

public Customer(String name,String dob)

{

super();

this.Name=name;

this.DOB=dob;

}

public Customer()

{

}

public void readData(String name,String dob)

{

this.Name=name;

this.DOB=dob;

}

public void displayData()

{

StringTokenizer st=new StringTokenizer(this.DOB,"/");

System.out.print(this.Name+",");

while(st.hasMoreTokens())

{

System.out.print(st.nextToken()+"/");

}

}

}

**Class Lp2B.Java**

import java.util.\*;

public class Lp2B {

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner in=new Scanner(System.in);

System.out.println("Enter Name");

String name=in.nextLine();

System.out.println("Enter DOB");

String date=in.next();

Customer customer=new Customer();

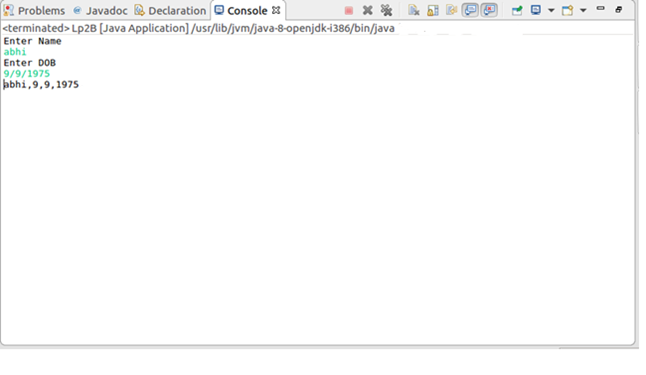
customer.readData(name, date);

customer.displayData();

}

}

OUTPUT:



**PROGRAM 3A**

**Write a Java program to read two integers a and b. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.**

**Description :**

The program accepts two parameters for the computation. The statements which are uncertain to be executed are put under the try block. During the execution, for the normal input, program does not raises any exception. Where as for the input where denominator is zero, exception is raised. Upon this, program searches any catches for the raises exception. Arithmetic Exception is written to serve this exception.

**Class Lp3A.java**

package maverick;

import java.util.\*;

public class Lp3A {

public static void main(String[] args) throws Exception {

// TODO Auto-generated method stub

int a,b;

float Q;

Scanner in=new Scanner(System.in);

System.out.println("Enter a");

a=in.nextInt();

System.out.println("Enter b");

b=in.nextInt();

try

{

if(b!=0)

{

Q=(float)1.0\*(a/b);

System.out.println("result="+Q);

}

else

throw new ArithmeticException();

}

catch(ArithmeticException e)

{

System.out.println("Divide by zero error");

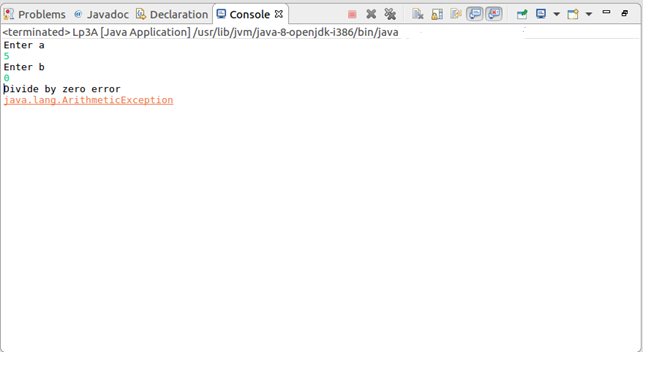
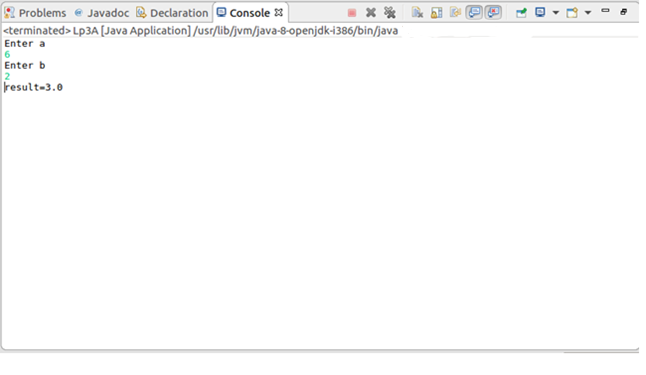
System.out.println(e);

}

}

}

OUTPUT:



**PROGRAM 3B**

**Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.**

**Description :**

The program begins with initiating and starting of the first thread which generates a random number and initiates two more threads for further processing of the generated random number. Square thread provides the square of the generated random number and Cube thread will provide the cube of a generated random number. This will be followed by a 1 second delay in the first thread. To demonstrate this delay, a loop is written to repeat this iteration set for 5 times, where we can observe the random number generation and the delay of 1 second clearly.

import java.util.\*;

public class Square implements Runnable {

public int x;

public Square(int x)

{

this.x=x;

}

public void run()

{

System.out.println("2nd Thread-Square of "+x+" is "+(x\*x));;

}

}

import java.util.\*;

public class Cube implements Runnable {

public int x;

public Cube(int x)

{

this.x=x;

}

public void run()

{

System.out.println("3rd Thread-Cube of "+x+" is "+(x\*x\*x));;

}

}

import java.util.\*;

public class ThreadRandom extends Thread {

public void run()

{

int n=0;

Random r=new Random();

try

{

for(int i=0;i<2;i++)

{

n=r.nextInt(100);

System.out.println("Main Thread Started and Generated "+n);

Thread t2=new Thread(new Square(n));

t2.start();

Thread t3=new Thread(new Cube(n));

t3.start();

Thread.sleep(1000);

System.out.println("----------");

}

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

import java.util.\*;

public class MultiThread {

public static void main(String[] args) {

// TODO Auto-generated method stub

ThreadRandom tr=new ThreadRandom();

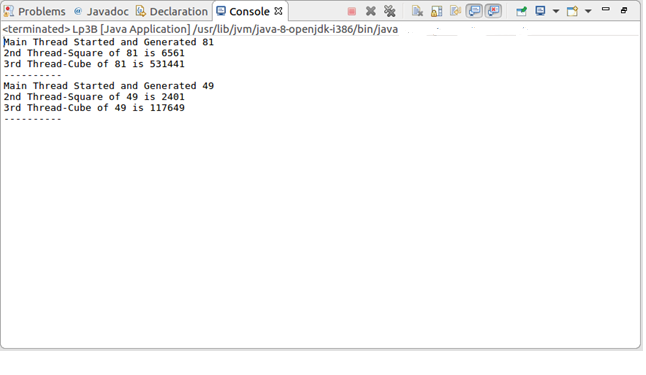
Thread t1=new Thread(tr);

t1.start();

}

}

OUTPUT:



**PROGRAM 4**

**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 non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and-conquer method works along with its time complexity analysis: worst case, average case and best case.**

**Algorithm for Quick Sort**

Algorithm partition(m, p)

// Purpose: Partition the array into two parts such that elements towards left of the key element are less than the key element and the elements towards right of the key element are greater than the key element.

// Input: A[0..n-1] – is a unsorted from the index position low to high.

//Output: A[0..n-1] – is divided into two parts as told above.

{

A[m],A[m+1],..A[p-1])

v A[m] //pivot element

i m

do

{

do {

i i+1

}while(A[i] > = v)

do

{

p p-1

}while(A[p]< = v)

if i < p

swap (A[i],A[p])

else

break

}while(1)

A[m]  A[p]

A[p]  v

return p

}

Algorithm quick\_sort(p,q)

// Purpose: Sort the elements of the array between the lower bound and upper bound.

// Input: A[0..n-1] – the list to be sorted with low and high as lower bound and upper bound

// n – the total number of elements.

//Output:A[0..n-1] – the sorted list

{

A[n+1]  

if(p < q)

then j q+1

partition (p,j)

quick\_sort(p,j-1)

quick\_sort(j+1,q)

}

**Time complexity of Quick Sort in Best case is (n log2 n)**

**Time complexity of Quick Sort in Worst case is O(n2)**

# Time complexity of Quick Sort in Average case is (n log2 n)

import java.util.\*;

public class Lp4 {

public static void main(String[] args) {

// TODO Auto-generated method stub

Lp4 Lab=new Lp4();

int a[]=new int[100000];

Scanner in=new Scanner(System.in);

long start,end;

System.out.println("QuickSort");

System.out.println("enter no. of elements");

int n=in.nextInt();

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

{

a[i]=Lab.generateRandom(a,10000);

}

System.out.println("elements to be sorted are");

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

{

System.out.print(a[i]+" ");

}

System.out.println();

start=System.nanoTime();

quicksort(a,0,n-1);

end=System.nanoTime();

System.out.println("the sorted elements are");

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

{

System.out.print(a[i]+" ");

}

System.out.println();

System.out.println("time taken is"+"\t"+(end-start)+"ns");

System.out.println("--------");

}

static void quicksort(int a[],int p,int q)

{

int j;

if(p<q)

{

j=partition(a,p,q);

quicksort(a,p,j-1);

quicksort(a,j+1,q);

}

}

static int partition(int a[],int m,int p)

{

int v,i,j;

v=a[m];

i=m;

j=p;

while(i<j)

{

while(a[i]<=v)

i++;

while(a[j]>v)

j--;

if(i<j)

interchange(a,i,j);

}

a[m]=a[j];

a[j]=v;

return j;

}

static void interchange(int a[],int i,int j)

{

int p;

p=a[i];

a[i]=a[j];

a[j]=p;

}

public int generateRandom(int a[],int bound)

{

Random r=new Random();

int offset=r.nextInt(bound);

while(alreadyThere(a,offset))

offset=r.nextInt(bound);

return offset;

}

private boolean alreadyThere(int arr[],int e)

{

for(int i=0;i<arr.length;i++)

{

if(e==arr[i])

return true;

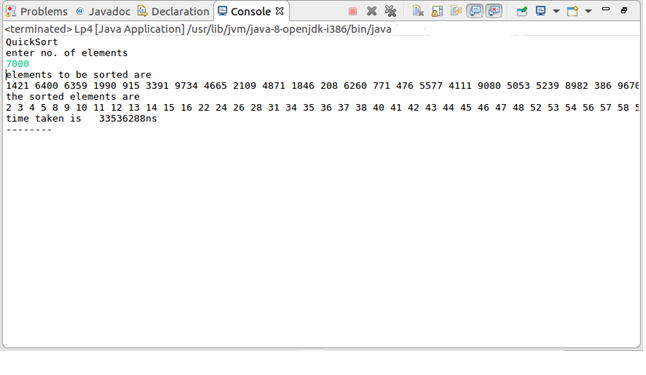
}

return false;

}

}

OUTPUT:



**PROGRAM 5**

**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 non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and-conquer method works along with its time complexity analysis: worst case, average case and best case.**

**Algorithm for Merge Sort**

//This algorithm sorts the element taking high and low.

Algorithm merge\_sort(low, high)

// Purpose: Sort the list in ascending order.

// Input: A[0..n-1] – the list to be sorted

// n – the total number of elements.

//Output:A[0..n-1] – the sorted list

{

if low < high

{

mid (low + high)/2

merge\_sort(low,mid)

merge\_sort(mid+1,high)

merge(low,mid,high)

}

}

Algorithm merge (low, mid, high)

// Purpose: Merge two sorted arrays where the first array starts from low to mid and the seconds starts form mid + 1 to high.

// Input: A[0..n-1] – is sorted from the index position low to mid.

// A[0..n-1] – is sorted from the index position mid + 1 to high.

// n – the total number of elements.

//Output:A[0..n-1] – the sorted from index low to high.

{

h low

i low

j mid+1

while(h < = mid and j < = high)

{

if A[h] < = A[j] then

{

B[i]  A[h]

h h+1

}

else

{

B[i]  A[j]

j j+1

}

ii+1

}

if (h > mid) then

{

for (k j to high) do

B[i]  A[k]

i i+1

}

else

{

for k h to mid do

B[i]  A[k]

i i+1

}

for (k low to high) do

A[k]  B[k]

}

**Time complexity of Merge Sort (n log2 n)**

package maverick;

import java.util.\*;

public class Lp5 {

public static void main(String[] args) {

// TODO Auto-generated method stub

Lp5 Lab=new Lp5();

int a[]=new int[100000];

Scanner in=new Scanner(System.in);

long start,end;

System.out.println("MergeSort");

System.out.println("enter no. of elements");

int n=in.nextInt();

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

{

a[i]=Lab.generateRandom(a,10000);

}

System.out.println("elements to be sorted are");

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

{

System.out.print(a[i]+" ");

}

System.out.println();

start=System.nanoTime();

mergesort(a,0,n-1);

end=System.nanoTime();

System.out.println("the sorted elements are");

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

{

System.out.print(a[i]+" ");

}

System.out.println();

System.out.println("time taken is"+"\t"+(end-start)+"ns");

System.out.println("--------");

}

static void mergesort(int a[],int l,int h)

{

int m;

if(l<h)

{

m=(l+h)/2;

mergesort(a,l,m);

mergesort(a,m+1,h);

merge(a,l,m,h);

}

}

static void merge(int a[],int low,int mid,int high)

{

int j,i,h,k;

int b[]=new int[100000];

h=low;

i=low;

j=mid+1;

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

{

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

{

b[i]=a[h];

h=h+1;

}

else

{

b[i]=a[j];

j=j+1;

}

i=i+1;

}

if(h>mid)

{

for(k=j;k<=high;k++)

{

b[i]=a[k];

i++;

}

}

else

{

for(k=h;k<=mid;k++)

{

b[i]=a[k];

i++;

}

}

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

{

a[k]=b[k];

}

}

public int generateRandom(int a[],int bound)

{

Random r=new Random();

int offset=r.nextInt(bound);

while(alreadyThere(a,offset))

offset=r.nextInt(bound);

return offset;

}

private boolean alreadyThere(int arr[],int e)

{

for(int i=0;i<arr.length;i++)

{

if(e==arr[i])

return true;

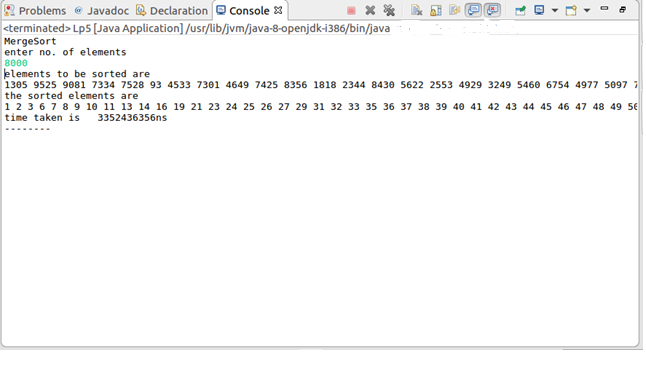
}

return false;

}

}

**OUTPUT:**



**PROGRAM 6A**

**Implement in Java, the 0/1 Knapsack problem using (a) Dynamic Programming method**

**Algorithm for Knapsack Problem**

Algorithm Knapsack / ele,rem\_cap, reach[ ])

//Purpose: to solve knapsack problem using dynamic programming.

//Input: n – the no. of objects to be selected.

m- the capacity of the knapsack

w- weights of all the objects.

p – profits of all the objects.

//Output: the optimal solution for the no. of objects selected with specified remaining capacity.

if ( ele num-1)

if (amount[ele] <= rem\_cap)

reach [ele]  1

return prof[ele]

return 0

if (amount [ele] > rem\_cap )

return (rec(ele +1, rem+cap,reach))

reach1[10]

for i 0 to num

reach1[i]  reach[i]

reach1[ele]  1

a rec(ele+1,rem\_cap-amount[ele],reacb1) + prof[ele]

b rec(ele+1,rem\_cap,reach)

if max(a,b)

for i 0 to num

reach[i]  reach1[i]

return a

return b

**Time complexity of Knapsack Problem is (mn)**

import java.util.\*;

public class Lp6A {

public static void main(String[ ] args) {

// TODO Auto-generated method stub

int v[ ][ ]=new int[10][10];

int w[ ]=new int[10];

int p[ ]=new int[10];

Scanner in=new Scanner(System.in);

int i,j;

System.out.println("0/1 Knapsack by Dynamic Programming");

System.out.println("enter total no. of items");

int n=in.nextInt();

System.out.println("enter weight of each item");

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

{

w[i]=in.nextInt();

}

System.out.println("enter profit of each item");

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

{

p[i]=in.nextInt();

}

System.out.println("enter Knapsack capacity");

int m=in.nextInt();

DisplayInfo(m,n,w,p);

Knapsack(m,n,w,p,v);

System.out.println("contents of Knapsack table are");

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

{

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

{

System.out.print(v[i][j]+" ");

}

System.out.println();

}

Optimal(m,n,w,v); }

static void DisplayInfo(int m,int n,int w[],int p[])

{

System.out.println("ITEM\tWEIGHT\tPROFIT");

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

{

System.out.println(i+"\t"+w[i]+"\t"+p[i]);

}

System.out.println("capacity="+"\t"+m);

}

static void Knapsack(int m,int n,int w[ ],int p[ ],int v[ ][ ])

{

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

{

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

{

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

v[i][j]=0;

else if(j<w[i])

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

else

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

}

}

}

private static int max(int i,int j)

{

if(i>j)

return i;

else

return j;

}

static void Optimal(int m,int n,int w[],int v[][])

{

int i=n,j=m,item=0;

int x[]=new int[10];

while(i!=0&&j!=0)

{

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

{

x[i]=1;

j=j-w[i];

}

i=i-1;

}

System.out.println("Optimal solution is"+"\t"+v[n][m]);

System.out.println("selected items are");

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

{

if(x[i]==1)

{

System.out.println(i+" ");

item=i;

}

if(item==0)

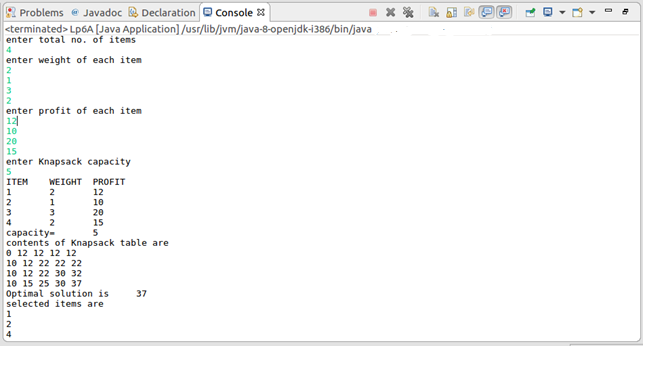
System.out.println("Knapsack is full");

}

}

}

OUTPUT:



**PROGRAM 6B**

**Greedy Technique:**

import java.util.\*;

public class Lp6B {

public static void main(String[] args) {

// TODO Auto-generated method stub

float w[ ]=new float[10],p[ ]=new float[10];

float ratio[ ]=new float[10];

Scanner in=new Scanner(System.in);

int i;

System.out.println("0/1 Knapsack problem by Greedy Method");

System.out.println("enter total no. of items");

int n=in.nextInt();System.out.println("enter weight of each item");

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

{

w[i]=in.nextFloat();

}

System.out.println("enter profit of each item");

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

{

p[i]=in.nextFloat();

}

System.out.println("enter Knapsack capacity");

int m=in.nextInt();

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

{

ratio[i]=p[i]/w[i];

}

System.out.println("Information about the problem is");

DisplayInfo(n,w,p,ratio);

System.out.println("capacity is"+"\t"+m);

SortArray(n,ratio,w,p);

System.out.println("Details after sorting items on p/w ratio in descending order");

DisplayInfo(n,w,p,ratio);

GreKnapsack(m,n,w,p);

}

static void SortArray(int n,float ratio[ ],float w[ ],float p[ ])

{

int i,j;

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

{

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

{

if(ratio[j]<ratio[j+1])

{

float temp=ratio[j];

ratio[j]=ratio[j+1];

ratio[j+1]=temp;

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

}

}

}

}

static void DisplayInfo(int n,float w[ ],float p[ ],float ratio[ ])

{

System.out.println("ITEM\tWEIGHT\tPROFIT\tRATIO");

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

{

System.out.println(i+"\t"+w[i]+"\t"+p[i]+"\t"+ratio[i]);

}

}

static void GreKnapsack(int u,int n,float w[ ],float p[ ])

{

float x[ ]=new float[10],tp=0;

int i;

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

x[i]=0;

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

{

if(w[i]>u)

break;

else

{

x[i]=1;

tp=tp+p[i];

u=(int)(u-w[i]);

}

}

if(i<n)

x[i]=u/w[i];

tp=tp+(x[i]\*p[i]);

System.out.println("Result is");

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

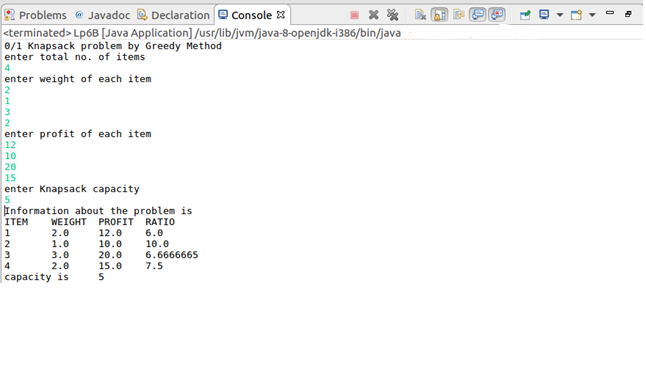
System.out.println("\t"+x[i]);

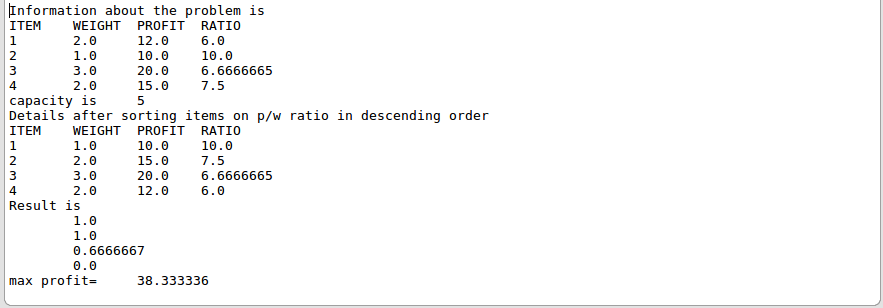
System.out.println("max profit="+"\t"+tp);

}

}

**OUTPUT:**





**PROGRAM 7**

**From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. Write the program in Java.**

# Dijkstra’s Algorithm

Algorithm Dijkstra (G)

//Purpose: To find the shortest path from the starting vertex to other vertices.

//Input: A weighted connected graph G=<V,E>

//Output: The length of shortest path from starting vertex to other vertices.

{

for i 1 to n

s[i]  0

dist[i]  cost[v][i]

s[v]  1

dist[v]  0

num 2

while ( num < = n)

u choose( )

s[u]  1

num++

for i w to n

if (((dist[u] + cost[u][w]) < dist[w] ) && (!s[w]))

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

}

//Algorithm to choose the nearest non selected vertex from a particular vertex

Algorithm choose ( )

{

j 1

min 1000

for (dist[w] < min) &&(!s[w]))

min  dist[w]

j w

return ( j )

}

# Time complexity of Dijkstra’s Algorithm is (E log V)

**Code:**

import java.util.\*;

public class Lp7 {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i,j;

int dist[]=new int[10],visited[]=new int[10];

int cost[][]=new int[10][10],path[]=new int[10];

Scanner in=new Scanner(System.in);

System.out.println("DIJKSTRA");

System.out.println("enter no. of nodes");

int n=in.nextInt();

System.out.println("enter cost matrix");

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

{

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

{

cost[i][j]=in.nextInt();

}

}

System.out.println("the entered cost matrix is");

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

{

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

{

System.out.print(cost[i][j]+"\t");

}

System.out.println();

}

System.out.println("enter source ");

int sv=in.nextInt();

Dij(cost,dist,sv,n,path,visited);

PrintPath(sv,n,dist,path,visited);

System.out.println("-----------");

}

static void Dij(int cost[][],int dist[],int sv,int n,int path[],int visited[])

{

int count=2,min,v=0;

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

{

visited[i]=0;

dist[i]=cost[sv][i];

if(cost[sv][i]==999)

path[i]=0;

else

path[i]=sv;

}

visited[sv]=1;

while(count<=n)

{

min=999;

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

if((dist[w]<min)&&(visited[w]==0))

{

min=dist[w];

v=w;

}

visited[v]=1;

count++;

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

{

if((dist[w]>dist[v]+cost[v][w]))

{

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

path[w]=v;

}

}

}

}

static void PrintPath(int sv,int n,int dist[],int path[],int visited[])

{

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

{

if(visited[w]==1&&w!=sv)

{

System.out.print("shortest distance between ");

System.out.println(sv+"->"+w+" is "+dist[w]);

int t=path[w];

System.out.print("the path is");

System.out.print(" "+w);

while(t!=sv)

{

System.out.print("<->"+t);

t=path[t];

}

System.out.println("<->"+sv);

}

}

} }

**OUTPUT:**

