11 Week DSA Workshop by GeeksforGeeks

==========

Week 0 - Day 1

- Why DSA Important
- Types of Structures
- Terminology

==========

Week 1 - Day 1

==========

Rain Water Trapping –

```
// Java implementation of the approach
class GFG{

// Function to return the maximum
// water that can be stored
public static int maxWater(int[] arr, int n)
{

        // To store the maximum water
        // that can be stored
        int res = 0;

        // For every element of the array
        // except first and last element
        for(int i = 1; i < n - 1; i++)
        {

                // Find maximum element on its left
```

```
                int left = arr[i];
                for(int j = 0; j < i; j++)
                {
                        left = Math.max(left, arr[j]);
                }

                // Find maximum element on its right
                int right = arr[i];
                for(int j = i + 1; j < n; j++)
                {
                        right = Math.max(right, arr[j]);
                }

                // Update maximum water value
                res += Math.min(left, right) - arr[i];
        }
        return res;
}

// Driver code
public static void main(String[] args)
{
        int[] arr = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
        int n = arr.length;

        System.out.print(maxWater(arr,n));
}
}
```

Sieve of Eratosthenes –

```
// Java program to print all
// primes smaller than or equal to
// n using Sieve of Eratosthenes

class SieveOfEratosthenes {
        void sieveOfEratosthenes(int n)
        {
                // Create a boolean array
                // "prime[0..n]" and
                // initialize all entries
                // it as true. A value in
                // prime[i] will finally be
                // false if i is Not a
                // prime, else true.
```

```
		boolean prime[] = new boolean[n + 1];
		for (int i = 0; i <= n; i++)
			prime[i] = true;

		for (int p = 2; p * p <= n; p++)
		{
			// If prime[p] is not changed, then it is a
			// prime
			if (prime[p] == true)
			{
				// Update all multiples of p
				for (int i = p * p; i <= n; i += p)
					prime[i] = false;
			}
		}

		// Print all prime numbers
		for (int i = 2; i <= n; i++)
		{
			if (prime[i] == true)
				System.out.print(i + " ");
		}
	}

	// Driver Code
	public static void main(String args[])
	{
		int n = 30;
		System.out.print(
			"Following are the prime numbers ");
		System.out.println("smaller than or equal to " + n);
		SieveOfEratosthenes g = new SieveOfEratosthenes();
		g.sieveOfEratosthenes(n);
	}
}
```

Print all substrings –

```
//Java program to print all possible
// substrings of a given string

class GFG {

// Function to print all sub strings
	static void subString(char str[], int n) {
```

```
                // Pick starting point
                for (int len = 1; len <= n; len++) {
                        // Pick ending point
                        for (int i = 0; i <= n - len; i++) {
                                // Print characters from current
                                // starting point to current ending
                                // point.
                                int j = i + len - 1;
                                for (int k = i; k <= j; k++) {
                                        System.out.print(str[k]);
                                }

                                System.out.println();
                        }
                }
        }

// Driver program to test above function
        public static void main(String[] args) {
                char str[] = {'a', 'b', 'c'};
                subString(str, str.length);

        }
}
```

==========

Week 1- Day 2

==========

Array Rotation –

```
// Java program to rotate an array by
// d elements

class RotateArray {
        /*Function to left rotate arr[] of size n by d*/
        void leftRotate(int arr[], int d, int n)
        {   int[] temp = new int[d];
                for (int i = 0; i < d; i++)
                        temp[i] = arr[i];
                for (int i = 0; i < n-d; i++)
                        arr[i] = arr[i+d];
```

```
            for (int i = 0; i < d; i++)
                arr[i+n-d] = temp[i];
        }


        /* utility function to print an array */
        void printArray(int arr[], int n)
        {
            for (int i = 0; i < n; i++)
                System.out.print(arr[i] + " ");
        }

        // Driver program to test above functions
        public static void main(String[] args)
        {
            RotateArray rotate = new RotateArray();
            int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
            rotate.leftRotate(arr, 2, 7);
            rotate.printArray(arr, 7);
        }
}
```

Method 2

```
// Java program to rotate an array by
// d elements

class RotateArray {
        /*Function to left rotate arr[] of size n by d*/
        void leftRotate(int arr[], int d, int n)
        {
            for (int i = 0; i < d; i++)
                leftRotatebyOne(arr, n);
        }

        void leftRotatebyOne(int arr[], int n)
        {
            int i, temp;
            temp = arr[0];
            for (i = 0; i < n - 1; i++)
                arr[i] = arr[i + 1];
            arr[n-1] = temp;
        }

        /* utility function to print an array */
        void printArray(int arr[], int n)
```

```
	{
		for (int i = 0; i < n; i++)
			System.out.print(arr[i] + " ");
	}

	// Driver program to test above functions
	public static void main(String[] args)
	{
		RotateArray rotate = new RotateArray();
		int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
		rotate.leftRotate(arr, 2, 7);
		rotate.printArray(arr, 7);
	}
}
```

Program to reverse an array

```
/* Basic Java program that reverses an array*/
public class reverseArray {

	/* function that reverses array and stores it
	in another array*/
	static void reverse(int a[], int n)
	{
		int[] b = new int[n];
		int j = n;
		for (int i = 0; i < n; i++) {
			b[j - 1] = a[i];
			j = j - 1;
		}

		/*printing the reversed array*/
		System.out.println("Reversed array is: \n");
		for (int k = 0; k < n; k++) {
			System.out.println(b[k]);
		}
	}

	public static void main(String[] args)
	{
		int [] arr = {10, 20, 30, 40, 50};
		reverse(arr, arr.length);
	}
}
```

Linear Search –

```
// Java code for linearly searching x in arr[]. If x
// is present then return its location, otherwise
// return -1

class GFG
{
	public static int search(int arr[], int x)
	{
		int n = arr.length;
		for (int i = 0; i < n; i++)
		{
			if (arr[i] == x)
				return i;
		}
		return -1;
	}

	// Driver code
	public static void main(String args[])
	{
		int arr[] = { 2, 3, 4, 10, 40 };
		int x = 10;

		// Function call
		int result = search(arr, x);
		if (result == -1)
			System.out.print(
				"Element is not present in array");
		else
			System.out.print("Element is present at index "
											+ result);
	}
}
```

Binary Search –

```
// Java implementation of recursive Binary Search
class BinarySearch {
	// Returns index of x if it is present in arr[l..
	// r], else return -1
	int binarySearch(int arr[], int l, int r, int x)
	{
		if (r >= l) {
```

```
                int mid = l + (r - l) / 2;

                // If the element is present at the
                // middle itself
                if (arr[mid] == x)
                        return mid;

                // If element is smaller than mid, then
                // it can only be present in left subarray
                if (arr[mid] > x)
                        return binarySearch(arr, l, mid - 1, x);

                // Else the element can only be present
                // in right subarray
                return binarySearch(arr, mid + 1, r, x);
        }

        // We reach here when element is not present
        // in array
        return -1;
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr, 0, n - 1, x);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index " + result);
    }
}
```

==========

Week 2 - Day 1

==========

Maximize Toys –

```
// Java Program to maximize the
// number of toys with K amount
import java.io.*;
import java .util.*;

class GFG
{
// This functions returns
// the required number of toys
static int maximum_toys(int cost[],
                                                int N, int K)
{
        int count = 0, sum = 0;

        // sort the cost array
        Arrays.sort(cost);
        for (int i = 0; i < N; i++)
        {

                // Check if we can buy ith toy or not
                if (sum +cost[i] <= K)
                {
                        sum = sum + cost[i];
                        // Increment count
                        count++;
                }
        }
        return count;
}

// Driver Code
public static void main (String[] args)
{
int K = 50;
int cost[] = {1, 12, 5, 111, 200,
                        1000, 10, 9, 12, 15};
int N = cost.length;

System.out.print( maximum_toys(cost, N, K));
}
```

}

First and last occurrence of an element -

```
// Java program to find first and last occurrence of
// an elements in given sorted array
import java.io.*;

class GFG {
	/* if x is present in arr[] then returns the index of
	FIRST occurrence of x in arr[0..n-1], otherwise
	returns -1 */
	public static int first(int arr[], int low, int high, int x, int n)
	{
		if (high >= low) {
			int mid = low + (high - low) / 2;
			if ((mid == 0 || x > arr[mid - 1]) && arr[mid] == x)
				return mid;
			else if (x > arr[mid])
				return first(arr, (mid + 1), high, x, n);
			else
				return first(arr, low, (mid - 1), x, n);
		}
		return -1;
	}

	/* if x is present in arr[] then returns the index of
	LAST occurrence of x in arr[0..n-1], otherwise
	returns -1 */
	public static int last(int arr[], int low, int high, int x, int n)
	{
		if (high >= low) {
			int mid = low + (high - low) / 2;
			if ((mid == n - 1 || x < arr[mid + 1]) && arr[mid] == x)
				return mid;
			else if (x < arr[mid])
				return last(arr, low, (mid - 1), x, n);
			else
				return last(arr, (mid + 1), high, x, n);
		}
		return -1;
	}

	public static void main(String[] args)
	{
```

```
                int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };
                int n = arr.length;
                int x = 8;
                System.out.println("First Occurrence = " + first(arr, 0, n - 1, x, n));
                System.out.println("Last Occurrence = " + last(arr, 0, n - 1, x, n));
        }
}
```

Find the row with maximum number of 1's.

==========

Week 2 - Day 2

==========

Hashing in C++

```
// C++ program to illustrate hashing using
// set in CPP STL

#include <iostream>
#include <set>
#include <iterator>

using namespace std;

int main()
{
    // empty set container
    set <int> s;

    // List of elements
    int arr[] = {40, 20, 60, 30, 50, 50, 10};

    // Insert the elements of the List
    // to the set
    for(int i = 0; i < 7; i++)
        s.insert(arr[i]);

    // Print the content of the set
    // The elements of the set will be sorted
    // without any duplicates
```

```
    cout<<"The elements in the set are: \n";
    for(auto itr=s.begin(); itr!=s.end(); itr++)
    {
        cout<<*itr<<" ";
    }

    // Check if 50 is present in the set
    if(s.find(50)!=s.end())
    {
        cout<<"\n\n50 is present";
    }
    else
    {
        cout<<"\n\n50 is not present";
    }

    return 0;
}
```

Hashing in C++

Hashing in Java

```
// Java program to demonstrate working of HashTable
import java.util.*;

class GFG {
    public static void main(String args[])
    {

        // Create a HashTable to store
        // String values corresponding to integer keys
        Hashtable<Integer, String>
            hm = new Hashtable<Integer, String>();

        // Input the values
        hm.put(1, "Geeks");
        hm.put(12, "forGeeks");
        hm.put(15, "A computer");
        hm.put(3, "Portal");

        // Printing the Hashtable
        System.out.println(hm);
    }
}
```

Hashing in java

Check whether array contains duplicates or not

```
function checkIfArrayIsUnique(arr) {
    var map = {}, i, size;

    for (i = 0, size = arr.length; i < size; i++){
        if (map[arr[i]]){
            return false;
        }

        map[arr[i]] = true;
    }

    return true;
}
```

Check if two arrays are equal or not

Print all pair with given sum

Find Winner of election

==========

Week 3 - Day 1

==========

A Boolean Matrix question

Sum of two large Numbers

```
// Java program to find sum of two large numbers.

import java.util.*;

class GFG
```

```
{
// Function for finding sum of larger numbers
static String findSum(String str1, String str2)
{
    // Before proceeding further, make sure length
    // of str2 is larger.
    if (str1.length() > str2.length()){
        String t = str1;
        str1 = str2;
        str2 = t;
    }

    // Take an empty String for storing result
    String str = "";

    // Calculate length of both String
    int n1 = str1.length(), n2 = str2.length();

    // Reverse both of Strings
```

```
str1=new StringBuilder(str1).reverse().toString();

str2=new StringBuilder(str2).reverse().toString();


int carry = 0;

for (int i = 0; i < n1; i++)

{

        // Do school mathematics, compute sum of

        // current digits and carry

        int sum = ((int)(str1.charAt(i) - '0') +

                                (int)(str2.charAt(i) - '0') + carry);

        str += (char)(sum % 10 + '0');


        // Calculate carry for next step

        carry = sum / 10;

}


// Add remaining digits of larger number

for (int i = n1; i < n2; i++)

{
```

```
            int sum = ((int)(str2.charAt(i) - '0') + carry);
            str += (char)(sum % 10 + '0');
            carry = sum / 10;
        }

        // Add remaining carry
        if (carry > 0)
            str += (char)(carry + '0');

        // reverse resultant String
        str = new StringBuilder(str).reverse().toString();

        return str;
}

// Driver code
public static void main(String[] args)
{
        String str1 = "12";
```

```
        String str2 = "198111";

        System.out.println(findSum(str1, str2));

}

}
```

==========

Week 3 - Day 2

==========

Check the number, power of 2.

Find optimal weights which can be used to weigh all the weights in the range

==========

Week 4 - Day 1

==========

Sum of numbers 1 to N. (Iterative)

```
public class SumNatural {

    public static void main(String[] args) {

        int num = 100, sum = 0;

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

        {

            // sum = sum + i;
```

```
            sum += i;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Sum of numbers 1 to N. (Recursive)

```
// Java program to find the
// sum of natural numbers up
// to n using recursion
import java.util.*;
import java.lang.*;
class GFG
{
        // Returns sum of first
        // n natural numbers
        public static int recurSum(int n)
        {
                if (n <= 1)
                        return n;
```

```
        return n + recurSum(n - 1);
    }
    // Driver code
    public static void main(String args[])
    {       int n = 5;
        System.out.println(recurSum(n));
    }
}
```

Program to count occurrence of a given character in a string(Iterative)

```
// JAVA program to count occurrences
// of a character
class GFG
{
    // Method that return count of the given
    // character in the string
    public static int count(String s, char c)
    {
        int res = 0;
```

```
        for (int i=0; i<s.length(); i++)
        {
            // checking character in string
            if (s.charAt(i) == c)
            res++;
        }
        return res;
    }
    // Driver method
    public static void main(String args[])
    {    String str= "geeksforgeeks";
        char c = 'e';
        System.out.println(count(str, c));
    }
}
```

Program to count occurrence of a given character in a string(Recursive)

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

	static int countinString(char ch, String s)

	{	//base case;

		if(s.length()==0)

			return 0;

		int count = 0;

		//checking if the first character of

		//the given string is that character

		//or not

		if(s.charAt(0)==ch)

		count++;

		//this will count the occurrence of

		//given character in the string

		//from index 1 to the last

		//index of the string

		count+=countinString(ch,s.substring(1));

		return count;
```

```
        }

        public static void main (String[] args) {

                String str= "geeksforgeeks";

                char c = 'e';

                System.out.println(countinString(c,str));

        }

}
```

Generate all the binary strings of N bits.

Count ways to reach the n’th stair.

==========

Week 4 - Day 2

==========

Decimal to Binary Conversion
Check whether K-th bit is set or not

LinkedList Introduction

Linked List | (Introduction)
Detect and Remove Loop in a Linked List
LRU Cache Implementation
Reverse a Linked List

==========

Week 5 - Day 1

==========

Stack Implementation --

Stack using Array--

```
/* Java program to implement basic stack
operations */
class Stack {
	static final int MAX = 1000;
	int top;
	int a[] = new int[MAX]; // Maximum size of Stack

	boolean isEmpty()
	{
		return (top < 0);
	}
	Stack()
	{
		top = -1;
	}

	boolean push(int x)
	{
		if (top >= (MAX - 1)) {
			System.out.println("Stack Overflow");
			return false;
		}
		else {
			a[++top] = x;
			System.out.println(x + " pushed into stack");
			return true;
		}
	}

	int pop()
	{
		if (top < 0) {
			System.out.println("Stack Underflow");
			return 0;
		}
		else {
			int x = a[top--];;
			return x;
		}
```

```
        }

        int peek()
        {
                if (top < 0) {
                        System.out.println("Stack Underflow");
                        return 0;
                }
                else {
                        int x = a[top];
                        return x;
                }
        }
}

// Driver code
class Main {
        public static void main(String args[])
        {
                Stack s = new Stack();
                s.push(10);
                s.push(20);
                s.push(30);
                System.out.println(s.pop() + " Popped from stack");
        }
}
```

Stack using LinkedList--

```
class Node<T> {
        T data;
        Node<T> next;

        public Node(T data) {
                this.data = data;
        }
}

public class Stack<T> {
```

```
Node<T> head;
int size;

    public Stack() {
  head=null;
  size=0;

    }

    public int size() {
  return size;

    }

    public void push(T data) {
  Node<T> a=new Node<>(data);
  a.next=head;
  head=a;
  size++;

    }

    public boolean isEmpty() {
  return size==0;

    }

    public T pop() throws StackEmptyException {
  Node<T> a=head;
  if(head==null){
     throw new StackEmptyException();
  }else{

     head=head.next;
     size--;
  }
  return a.data;

    }

    public T top() throws StackEmptyException {
  if(head==null){
     throw new StackEmptyException();
  }else{
     return head.data;
  }
```

```
        }
}
class StackEmptyException extends Exception {

}
```

Q. Design a Data Structure for LRU Cache.

==========

Week 5- Day 2

==========

Queue Implementation --

Queue using LinkedList--

```
class Node<T> {
        T data;
        Node<T> next;

        public Node(T data) {
                this.data = data;
        }
}


 **********/
public class Queue<T> {
    Node<T> front;
    Node<T> rear;
    int size;
    // LinkedList<T> m=new LinkedList<>();
```

```
   public Queue() {
front=null;
rear=null;
size=0;

   }

   public void enqueue(T data) {
Node<T> a=new Node<>(data);
if(size==0){
    front=a;
    rear=a;
    size++;
}else{
    rear.next=a;
    rear=a;
    size++;
}

   }

   public int size() {
return size;

   }

   public boolean isEmpty() {
return size==0;

   }

   public T dequeue() throws QueueEmptyException {
T b=front.data;
if(size==0){
    throw new QueueEmptyException();
}else{
    Node<T> a=front.next;
    front.next=null;
    front=a;
    size--;
}
return b;

   }
```

```
    public T front() throws QueueEmptyException {
    if(size==0){
        throw new QueueEmptyException();
    }else{
        return front.data;
    }

    }
}
class QueueEmptyException extends Exception{

}
```

Queue using Array

```
// Java program to implement a queue using an array
class Queue {
    private static int front, rear, capacity;
    private static int queue[];

    Queue(int c)
    {
        front = rear = 0;
        capacity = c;
        queue = new int[capacity];
    }

    // function to insert an element
    // at the rear of the queue
    static void queueEnqueue(int data)
    {
        // check queue is full or not
        if (capacity == rear) {
            System.out.printf("\nQueue is full\n");
            return;
        }

        // insert element at the rear
        else {
            queue[rear] = data;
            rear++;
        }
        return;
    }

    // function to delete an element
```

```
// from the front of the queue
static void queueDequeue()
{
	// if queue is empty
	if (front == rear) {
		System.out.printf("\nQueue is empty\n");
		return;
	}

	// shift all the elements from index 2 till rear
	// to the right by one
	else {
		for (int i = 0; i < rear - 1; i++) {
			queue[i] = queue[i + 1];
		}

		// store 0 at rear indicating there's no element
		if (rear < capacity)
			queue[rear] = 0;

		// decrement rear
		rear--;
	}
	return;
}

// print queue elements
static void queueDisplay()
{
	int i;
	if (front == rear) {
		System.out.printf("\nQueue is Empty\n");
		return;
	}

	// traverse front to rear and print elements
	for (i = front; i < rear; i++) {
		System.out.printf(" %d <-- ", queue[i]);
	}
	return;
}

// print front of queue
static void queueFront()
{
	if (front == rear) {
```

```
                System.out.printf("\nQueue is Empty\n");
                return;
            }
            System.out.printf("\nFront Element is: %d", queue[front]);
            return;
        }
}

public class StaticQueueinjava {

        // Driver code
        public static void main(String[] args)
        {
            // Create a queue of capacity 4
            Queue q = new Queue(4);
             // print Queue elements
            q.queueDisplay();

            // inserting elements in the queue
            q.queueEnqueue(20);
            q.queueEnqueue(30);
            q.queueEnqueue(40);
            q.queueEnqueue(50);

            // print Queue elements
            q.queueDisplay();

            // insert element in the queue
            q.queueEnqueue(60);

            // print Queue elements
            q.queueDisplay();

            q.queueDequeue();
            q.queueDequeue();
            System.out.printf("\n\nafter two node deletion\n\n");

            // print Queue elements
            q.queueDisplay();

            // print front of the queue
            q.queueFront();
        }
}
```

==========

Week 6

==========

STACK QUESTIONS

https://www.geeksforgeeks.org/largest-rectangle-under-histogram/

https://www.geeksforgeeks.org/next-greater-element/

https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/

https://www.geeksforgeeks.org/reverse-stack-without-using-extra-space/

==========

Week 7-Day 1

==========

RECURSION

https://www.geeksforgeeks.org/program-for-factorial-of-a-number/

https://www.geeksforgeeks.org/different-ways-to-print-fibonacci-series-in-java/

https://www.geeksforgeeks.org/program-check-array-sorted-not-iterative-recursive/

https://www.geeksforgeeks.org/f.gind-possible-words-phone-digits/

https://www.geeksforgeeks.org/generating-all-possible-subsequences-using-recursion/

==========

Week 7-Day 2,Week 8-Day 1

==========

BINARY TREES

```
package gfg;
import java.util.*;

//Making node for a tree

class TreeNode{
    int data;
    TreeNode left;
    TreeNode right;
    public TreeNode(int data){
        this.data=data;
    }
}

//Class for question-check if a tree is balanced(optimized)

class CheckBalanced {
        int height;
        boolean isBal;
        CheckBalanced(int height,boolean isBal){
                this.height=height;
                this.isBal=isBal;
        }
}

//Class for question- finding diameter of binary tree(optimized)

class DiameterCheck{
        int height;
        int diameter;
        DiameterCheck(int height,int diameter){
                this.height=height;
```

```
                this.diameter=diameter;
        }
}
```

//Function to take input from the user

```
public class Tree {
   public static TreeNode takeInput(boolean isRoot,int parentData,boolean isLeft){
      if(isRoot){
         System.out.print("Enter root's data : ");
      }else{
         if(isLeft){
            System.out.print("Enter left child of "+parentData+ " : ");
         }else{
            System.out.print("Enter right child of "+parentData+ " : ");
         }
      }
      Scanner s=new Scanner(System.in);

      int data=s.nextInt();
      if(data==-1){
         return null;
      }
      TreeNode root=new TreeNode(data);
      TreeNode rootLeft=takeInput(false,data,true);
      TreeNode rootRight=takeInput(false,data,false);
      root.left=rootLeft;
      root.right=rootRight;
      return root;

   }
```

//Function to print binary tree

```
public static void print(TreeNode root){
            if(root==null){
               return;
            }
            System.out.print(root.data);
            if(root.left!=null){
               System.out.print(": L :"+root.left.data);
            }
            if(root.right!=null){
               System.out.print(": R :"+root.right.data);
            }
            System.out.println();
```

```
        print(root.left);
        print(root.right);
    }
```

//Main function

```
public static void main (String[] args) {

    TreeNode root =takeInput(true,-1,false);
            print(root);
            System.out.println(diameter(root));
            System.out.println(diameterNew(root).diameter);
            System.out.println(countNodes(root));
            System.out.println(isBalanced(root));
            System.out.println(isBalance(root).isBal);
            System.out.print(kDistance(root,5,3));
            LevelOrder(root);
    }
```

//function to calculate number of nodes in a binary tree

```
public static int countNodes(TreeNode root) {
    if(root==null) {
            return 0;
    }
    int leftNodes=countNodes(root.left);
    int rightNodes=countNodes(root.right);
    return 1+leftNodes+rightNodes;
}
```

//function to remove leaf nodes from a binary tree

```
public static TreeNode removeleaves(TreeNode root) {
    if(root==null) {
            return null;
    }
    if(root.left==null && root.right==null) {
            return null;
    }
    root.left=removeleaves(root.left);
    root.right=removeleaves(root.right);
    return root;
}
```

//function to calculate height of a binary tree

```
    public static int height(TreeNode root) {
        if(root==null) {
                return 0;
        }
        int leftNodes=height(root.left);
        int rightNodes=height(root.right);
        return 1+Math.max(leftNodes, rightNodes);
    }
```

//function to print nodes at level h in a binary tree

```
    public static void level(TreeNode root,int h) {
        if(root==null) {
                return;
        }
        if(h==0) {
                System.out.println(root.data);
                return;
        }
        level(root.left,h-1);
        level(root.right,h-1);
    }
```

//function to calculate number of leaf nodes

```
    public static int leafNodes(TreeNode root) {
        if(root==null) {
                return 0;
        }
        if(root.left==null && root.right==null) {
                return 1;
        }
        return leafNodes(root.left)+leafNodes(root.right);
    }
```

//function to mirror a binary tree

```
    public static void mirror(TreeNode root) {
        if(root==null) {
                return;
        }
        TreeNode temp=root.left;
        root.left=root.right;
        root.right=temp;
        mirror(root.left);
```

```
        mirror(root.right);
    }
```

//Function to check if binary tree is balanced or not(Time complexity --O(n^2))

```
        public static boolean isBalanced(TreeNode root) {
                if(root==null) {
                        return true;
                }
                if(Math.abs(height(root.left)-height(root.right))>1) {
                        return false;
                }
                boolean leftStatus=isBalanced(root.left);
                boolean rightStatus=isBalanced(root.right);
                return leftStatus && rightStatus;

        }
```

//Function to check if binary tree is balanced or not(Time complexity --O(n))

```
        public static CheckBalanced isBalance(TreeNode root) {
                if(root==null) {
                        return new CheckBalanced(0,true);
                }
                CheckBalanced left=isBalance(root.left);
                CheckBalanced right=isBalance(root.right);
                CheckBalanced ans=new CheckBalanced(1+left.height+right.height,true);
                if(Math.abs(left.height-right.height)>1) {
                        ans.isBal=false;
                        return ans;
                }
                if(!left.isBal || !right.isBal) {
                        ans.isBal=false;
                        return ans;
                }
                return ans;

        }
```

//Function to calculate diameter of a binary tree(Time complexity --O(n^2))

```
        public static int diameter(TreeNode root) {
                if(root==null) {
                        return 0;
                }
                int left=height(root.left);
```

```
		int right=height(root.right);
		int d=left+right+1;
		int leftmax=diameter(root.left);
		int rightmax=diameter(root.right);
		return Math.max(Math.max(leftmax, rightmax), d);

	}
```

//Function to calculate diameter of a binary tree(Time complexity --O(n))

```
	public static DiameterCheck diameterNew(TreeNode root) {
		if(root==null) {
			return new DiameterCheck(0,0);
		}
		DiameterCheck leftmax=diameterNew(root.left);
		DiameterCheck rightmax=diameterNew(root.right);
		int h=leftmax.height+rightmax.height+1;
		int d=Math.max(h,Math.max(leftmax.diameter, rightmax.diameter));
		DiameterCheck ans=new DiameterCheck(Math.max(leftmax.height,
rightmax.height)+1,d);
		return ans;

	}
```

//Function to print all nodes at k distance from a given node in a binary tree

```
	public static int kDistance(TreeNode root,int data,int k) {
		if(root==null) {
			return -1;
		}
		if(root.data==data) {
			kDistanceDown(root,k);
			return 0;
		}
		int leftD=kDistance(root.left,data,k);
		if(leftD!=-1) {
			if(leftD+1==k) {
				System.out.print(root.data);
			}else {
				kDistanceDown(root.right,k-leftD-2);
			}
			return leftD+1;
		}
		int rightD=kDistance(root.left,data,k);
		if(rightD!=-1) {
			if(rightD+1==k) {
```

```
                System.out.print(root.data);
            }else {
                kDistanceDown(root.left,k-rightD-2);
            }
            return rightD+1;
        }
        return -1;
    }
```

//Function to print all nodes at distance k from root

```
    public static void kDistanceDown(TreeNode root,int k) {
        if(root==null) {
            return;
        }
        if(k==0) {
            System.out.println(root.data);
            return;
        }
        kDistanceDown(root.left,k-1);
        kDistanceDown(root.right,k-1);

    }
```

// Level order traversal

```
    public static void levelOrder(TreeNode root) {
        Queue<TreeNode> q=new LinkedList<>();
        q.add(root);
        while(!q.isEmpty()) {
            TreeNode p=q.poll();
            System.out.print(p.data);
            if(p.left!=null) {
                q.add(p.left);
            }
            if(p.right!=null) {
                q.add(p.right);
            }
        }
    }
```

==========

Week 8-Day 2

==========

BINARY SEARCH TREES

```
package gfg;
import java.util.*;

// Node of a binary tree

class Node{
    int data;
    Node left;
    Node right;
    public Node(int data){
        this.data=data;
    }
}

// Class for optimized solution of question-if a tree is Bst or not

class BstSet{
        int min;
        int max;
        boolean isBST;
        BstSet(int min,int max,boolean isBST){
                this.min=min;
                this.max=max;
                this.isBST=isBST;
        }
}




public class BSTree {

// Function to take input of a binary tree

    public static Node takeInput(boolean isRoot,int parentData,boolean isLeft){
        if(isRoot){
            System.out.print("Enter root's data : ");
        }else{
            if(isLeft){
                System.out.print("Enter left child of "+parentData+ " : ");
```

```
        }else{
            System.out.print("Enter right child of "+parentData+ " : ");
        }
    }
    Scanner s=new Scanner(System.in);

    int data=s.nextInt();
    if(data==-1){
        return null;
    }
    Node root=new Node(data);
    Node rootLeft=takeInput(false,data,true);
    Node rootRight=takeInput(false,data,false);
    root.left=rootLeft;
    root.right=rootRight;
    return root;

}
```

// Function to print binary tree

```
    public static void print(Node root){
            if(root==null){
                return;
            }
            System.out.print(root.data);
            if(root.left!=null){
                System.out.print(": L :"+root.left.data);
            }
            if(root.right!=null){
                System.out.print(": R :"+root.right.data);
            }
            System.out.println();
            print(root.left);
            print(root.right);
        }
```

// Main function

```
    public static void main (String[] args) {
        int[] arr= {1,2,3,4,5,6,7,8};
        Node root=arrayToBinary(arr,0,arr.length-1);
        print(root);
//      Node root =takeInput(true,-1,false);
//              print(root);
//              System.out.print(search(root,3));
```

```
//                System.out.print(searchBST(5,root));
        inorder(root);
                System.out.println(isBST(root));
                System.out.println(isBstNew(root).isBST);
                System.out.println(lca(root,5,8).data);

        }
```

// Inorder traversal (Inorder traversal of Bst gives sorted order of elements of a tree)

```
    public static void inorder(Node root) {
        if(root==null) {
                return;
        }
        inorder(root.left);
        System.out.print(root.data+" ");
        inorder(root.right);
    }
```

// Function to print all nodes in range k1 and k2

```
    public static void searchInRange(Node root,int k1,int k2) {
        if(root==null) {
                return;
        }
        if(root.data<k1 && root.data<k2) {
                searchInRange(root.right,k1,k2);
        }
        else if(root.data>k1 && root.data>k2) {
                searchInRange(root.left,k1,k2);
        }else {
                System.out.print(root.data+" ");
                searchInRange(root.left,k1,k2);
                searchInRange(root.right,k1,k2);
        }
    }
```

// Function to convert sorted array to BST

```
    public static Node arrayToBinary(int[] arr,int start,int end) {
        if(start>end) {
                return null;
        }
        int mid=(start+end)/2;
```

```
        Node root=new Node(arr[mid]);
        root.left=arrayToBinary(arr,start,mid-1);
        root.right=arrayToBinary(arr,mid+1,end);
        return root;
    }
```

// Function to search in a binary tree

```
    public static boolean search(Node root,int x) {
        if(root==null) {
                return false;
        }
        if(root.data== x) {
                return true;
        }
        if(root.data>x) {
                return search(root.left,x);
        }else {
                return search(root.right,x);
        }
    }
```

// Function to check if binary tree is BST or not(Time complexity --O(n^2))

```
    public static boolean isBST(Node root) {
        if(root==null) {
                return true;
        }
        int max=findMAX(root.left);
        int min=findMIN(root.right);
        if(max>=root.data) {
                return false;
        }
        if(min<root.data) {
                return false;
        }
        boolean left=isBST(root.left);
        boolean right=isBST(root.right);
        return left && right;

    }
```

// Function to calculate maximum node in BT

```
    public static int findMAX(Node root) {
        if(root==null) {
```

```
            return Integer.MIN_VALUE;
        }
        return Math.max(Math.max(findMAX(root.left), findMAX(root.right)),root.data);
    }
```

// Function to calculate minimum node in BT

```
    public static int findMIN(Node root) {
        if(root==null) {
            return Integer.MAX_VALUE;
        }
        return Math.min(Math.min(findMIN(root.left), findMIN(root.right)),root.data);
    }
```

// Function to check if binary tree is BST or not(Time complexity --O(n))

```
    public static BstSet isBstNew(Node root) {
        if(root==null) {
            return new BstSet(Integer.MAX_VALUE,Integer.MIN_VALUE,true);
        }
        BstSet left=isBstNew(root.left);
        BstSet right=isBstNew(root.right);
        int min=Math.min(Math.min(left.min, right.min), root.data);
        int max=Math.max(Math.max(left.max, right.max), root.data);
        BstSet ans=new BstSet(min,max,true);

        if(!left.isBST) {
            ans.isBST=false;
        }
        if(!right.isBST) {
            ans.isBST=false;
        }
        if(left.max>=root.data) {
            ans.isBST=false;
        }
        if(right.min<root.data) {
            ans.isBST=false;
        }
        return ans;
    }
```

// Function to check if binary tree is BST or not(Time complexity --O(n))

```
    public static boolean isBST3(Node root,int leftRange,int rightRange) {
        if(root==null) {
            return true;
```

```
        }
        if(root.data<leftRange || root.data>rightRange ) {
                return false;
        }
        boolean left=isBST3(root.left,root.data-1,Integer.MIN_VALUE);
        boolean right=isBST3(root.right,Integer.MAX_VALUE,root.data);
        return left && right;
    }
```

// Function to calculate lowest common ancestor of 2 nodes

```
    public static Node lca(Node root,int x,int y) {
        if(root==null || root.data==x || root.data==y) {
                return root;
        }
        if(root.data>x && root.data >y) {
                return lca(root.left,x,y);
        }else if(root.data<x && root.data<y) {
                return lca(root.right,x,y);
        }else {
                Node left=lca(root.left,x,y);
                Node right=lca(root.right,x,y);
                if(left!=null && right!=null) {
                        return root;
                }
                if(left==null) {
                        return right;
                }
                return left;
        }

    }
```

// Function to calculate path from root to target node

```
    public static ArrayList<Integer> path(Node root,int target){
        if(root==null) {
                return null;
        }
        if(root.data==target) {
                ArrayList<Integer> arr=new ArrayList<>();
                arr.add(root.data);
                return arr;
        }
```

```
        if(target>root.data) {
                ArrayList<Integer> right=path(root.right,target);
                if(right!=null) {
                        right.add(root.data);
                        return right;
                }
                return null;
        }
        else {
                ArrayList<Integer> left=path(root.left,target);
                if(left!=null) {
                        left.add(root.data);
                        return left;
                }
                return null;
        }
   }
}
```

==========

Week 9-Day 1

==========

Priority Queue

```
package gfg;
import java.util.*;
class Element{
        int data;
        int priority;
        public Element(int data,int priority) {
                this.data=data;
                this.priority=priority;
        }
}
public class Heap {
        ArrayList<Element> heap;
        public Heap() {
                heap = new ArrayList<>();
        }
        public boolean isEmpty() {
                return heap.size()==0;
```

```
	}
	public int size() {
		return heap.size();
	}
	public int removeMin() throws PriorityQueueEmptyException {
		if(isEmpty()) {
			throw new PriorityQueueEmptyException();
		}
		int removed=heap.get(0).data;
		heap.set(0,heap.get(heap.size()-1));
		heap.remove(heap.size()-1);
		int parentIndex=0;
		int leftChild=2*parentIndex+1;
		int rightChild=2*parentIndex+2;
		int minIndex=parentIndex;
		while(leftChild<heap.size()) {
			if(heap.get(minIndex).priority>heap.get(leftChild).priority) {
				minIndex=leftChild;
			}
			if(rightChild<heap.size() &&
heap.get(minIndex).priority>heap.get(rightChild).priority) {
				minIndex=rightChild;
			}
			if(parentIndex==minIndex) {
				break;
			}
			Element temp=heap.get(parentIndex);
			heap.set(parentIndex, heap.get(minIndex));
			heap.set(minIndex, temp);
			parentIndex=minIndex;
			leftChild=2*parentIndex+1;
			rightChild=2*parentIndex+2;
		}
		return removed;

	}
	public int getMin() throws PriorityQueueEmptyException {
		if(isEmpty()) {
			throw new PriorityQueueEmptyException();
		}
		return heap.get(0).data;
	}
	public void insert(int data,int priority){
		Element ele=new Element(data,priority);
		heap.add(ele);
		int childIndex=heap.size()-1;
```

```
                int parentIndex=(childIndex-1)/2;
                while(childIndex>0) {
                        if(heap.get(parentIndex).priority>heap.get(childIndex).priority) {
                                Element temp=heap.get(parentIndex);
                                heap.set(parentIndex, heap.get(childIndex));
                                heap.set(childIndex, temp);
                                childIndex=parentIndex;
                                parentIndex=(childIndex-1)/2;
                        }else {
                                return;
                        }
                }


        }



}
```

==========

Week 9-Day 2

==========

Graphs

```
package gfg;
import java.util.*;
public class Graphs {

        public static void main(String[] args) {
                Scanner s=new Scanner(System.in);
                int n=s.nextInt();
                int e=s.nextInt();
                int[][] adj=new int[n][n];
                for(int i=0;i<e;i++) {
```

```
                int v1=s.nextInt();
                int v2=s.nextInt();
                adj[v1][v2]=1;
                adj[v2][v1]=1;
            }
            boolean visited[]=new boolean[n];
            dfs(adj,0,visited);
//          for(int i=0;i<visited.length;i++) {
//              if(!visited[i]) {
//                  dfs(adj,i,visited);
//              }
//          }

//          bfs(adj,visited,0);
            System.out.println();
            for(int i=0;i<n;i++) {
                for(int j=0;j<n;j++) {
                    System.out.print(adj[i][j]+" ");
                }
                System.out.println();
            }

        }
        public static void dfs(int[][] adj,int current,boolean[] visited) {
            System.out.print(current+" ");
            visited[current]=true;
            for(int i=0;i<adj.length;i++) {
                if(adj[current][i]==1 && !visited[i]) {
                    dfs(adj,i,visited);
                }
            }
        }
        public static void bfs(int[][] adj,boolean[] visited,int current) {
            Queue<Integer> q=new LinkedList<>();
            q.add(current);
            visited[current]=true;
            while(!q.isEmpty()) {
                int ele=q.remove();
                System.out.print(ele+" ");
                for(int i=0;i<adj.length;i++) {
                    if(!visited[i] && adj[ele][i]==1) {
                        q.add(i);
                        visited[i]=true;
                    }
                }
```

```
		}
	}

}
```

==========

Week 10-Day 1

==========

Graphs

Get Path -DFS

```
public static ArrayList<Integer> getpathdfs(int[][] adj,int source,int dest,boolean[] visited)
{
		if(source==dest) {
			ArrayList<Integer> ans=new ArrayList<>();
			ans.add(dest);
			return ans;
		}
		visited[source]=true;
		for(int i=0;i<adj.length;i++) {
			if(adj[source][i]==1 && !visited[i]) {
				ArrayList<Integer> ans=getpathdfs(adj,i,dest,visited);
				if(ans!=null) {
					ans.add(i);
				}
				return ans;
			}
		}
		return null;
	}
```

Get Path -BFS

```
public static ArrayList<Integer> getPathbfs(int[][] adj,boolean[] visited,int source,int dest)
{
		Queue<Integer> q=new LinkedList<>();
		HashMap<Integer,Integer> hm=new HashMap<>();
		q.add(source);
		hm.put(source, -1);
		visited[source]=true;
		while(!q.isEmpty()) {
			int ele=q.remove();
			for(int i=0;i<adj.length;i++) {
				if(!visited[i] && adj[ele][i]==1) {
					q.add(i);
					visited[i]=true;
					hm.put(i, ele);
					if(i==dest) {
						ArrayList<Integer> ans=new ArrayList<>();
						ans.add(dest);
						int parent=hm.get(i);
						while(parent!=-1) {
							ans.add(parent);
							parent=hm.get(parent);
						}
						return ans;
					}
				}
			}

		}
		return null;
	}
```

Has Path

```
public static boolean haspath(int a,int b,int adj[][],boolean[] vis) {
		if(a==b) {
			return true;
		}
		for(int i=0;i<adj.length;i++) {
			if(!vis[i] && adj[a][i]==1) {
```

```
                        return haspath(i,b,adj,vis);
                    }
                }
                return false;
        }
```

Kruskal's Algorithm

```
package gfg;

import java.util.Arrays;
import java.util.Scanner;
class Edge implements Comparable<Edge>{
        int v1;
        int v2;
        int w;
        public Edge(int v1,int v2,int w) {
                this.v1=v1;
                this.v2=v2;
                this.w=w;
        }
        @Override
        public int compareTo(Edge o) {
                return this.w-o.w;
        }
}

public class graphss {
        public static void main(String[] args) {
                Scanner s=new Scanner(System.in);
                int v=s.nextInt();
                int e=s.nextInt();
                Edge ed[]=new Edge[e];
                for(int i=0;i<e;i++) {
                        int v1=s.nextInt();
                        int v2=s.nextInt();
                        int w=s.nextInt();
                        Edge edge=new Edge(v1,v2,w);
                        ed[i]=edge;
                }
                Edge[] ans=kruskal(ed,v);
                for(int i=0;i<e;i++) {
                        System.out.print(ans[i].v1 +" "+ans[i].v2+" "+ans[i].w);
                }
        }
        public static int findParent(int v,int[] parent) {
```

```
		if(v==parent[v]) {
			return v;
		}
		return findParent(parent[v],parent);

	}
	public static Edge[] kruskal(Edge[] ed,int v) {
		Arrays.sort(ed);
		int parent[]=new int[v];
		for(int i=0;i<v;i++) {
			parent[i]=i;
		}
		int count=0;
		Edge[] ans=new Edge[v-1];
		int j=0;
		while(count!=v-1) {
			int v1Parent=findParent(ed[j].v1,parent);
			int v2Parent=findParent(ed[j].v2,parent);
			if(v1Parent!=v2Parent) {
				parent[v1Parent]=v2Parent;
				ans[count]=ed[j];
				count++;
			}
			j++;

		}
		return ans;
	}


}
```

==========

Week 10-Day 2

==========

Activity Selection

```
import java.util.*;
class Pair{
	int start;
	int end;
```

```
        Pair(int start,int end){
        this.start=start;7
        this.end=end;
        }
}
class PairComparator implements Comparator<Pair>{
        public int compare(Pair a,Pair b){
        return a.end-b.end;
        }
}

public class Main {


    public static void main(String[] args) {
         Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        Pair a[]=new Pair[n];
        for(int i=0;i<n;i++){
        int k=sc.nextInt();
        int l=sc.nextInt();
        a[i]=new Pair(k,l);
        }
        Arrays.sort(a,new PairComparator());
        int count=1;
        int c=a[0].end;
        for(int i=0;i<n-1;i++){
        if(c<=a[i+1].start){
                count++;
                c=a[i+1].end;
        }
        }
        System.out.print(count);
    }

}
```

Fractional Knapsack

```
import java.util.*;
class triplet{
        int time;
        int cost;
        int speed;
        double spc;
        triplet(int time,int cost,int speed){
```

```
        this.time=time;
        this.speed=speed;
        this.cost=cost;
        this.spc=(double)(this.speed)/(double)(this.cost);
        }
}
class tripletComparator implements Comparator<triplet>{

        public int compare(triplet t,triplet t1){
        int timeCompare=t.time-t1.time;
        double spcCompare=t1.spc-t.spc;
        if(timeCompare==0){
        return spcCompare==0.0 ? timeCompare : (int)spcCompare;
        }else{
        return timeCompare;
        }
        }
}


public class Main {


    public static void main(String[] args) {
         Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        long tar=sc.nextLong();
        triplet[] arr=new triplet[n];
        for(int i=0;i<n;i++){
        int time=sc.nextInt();
        int cost=sc.nextInt();
        int speed=sc.nextInt();
        arr[i]= new triplet(time,cost,speed);
        }
        Arrays.sort(arr,new tripletComparator());
        // for(int i=0;i<n;i++){
        //        System.out.println(arr[i].time+" "+arr[i].speed+" "+arr[i].spc);
        // }
        double spc=    Integer.MIN_VALUE;
        int time=0;
        long cost=0;
        int speed=0;
        long count=0;
        for(int i=0;i<n;i++){
        if(arr[i].time>time && arr[i].spc>=spc){
                cost+=arr[i].cost;
                speed=arr[i].speed;
```

```
                count+=(speed)*(arr[i].time-time);
                spc=arr[i].spc;
                time=arr[i].time;
        }
        if(count>=tar){
                System.out.print(cost);
                return;
        }
        }
        System.out.print(cost);

    }


}
```

Job Scheduling

https://www.geeksforgeeks.org/job-sequencing-problem/

==========

Week 11-Day 1

==========

Dynamic Programming

Fibonacci Number using recursion /dp

https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/

Staircase Problem

https://www.geeksforgeeks.org/count-ways-reach-nth-stair/

Minimum steps to 1

https://www.geeksforgeeks.org/minimum-steps-minimize-n-per-given-condition/

==========

Week 11-Day 2

==========

BackTracking

Rat in a maze--

```
public class Solution {



    public static void ratInAMaze(int maze[][]){
         int[][] path=new int[maze.length][maze.length];
        m(maze,path,0,0);
    }
        public static void m(int[][] maze,int[][] path,int i,int j){

        if(i<0||i>maze.length-1||j<0||j>maze.length-
1||maze[i][j]==0||path[i][j]==1){
        return;
        }
        path[i][j]=1;
        if(i==maze.length-1 && j==maze.length-1){
        for(int r=0;r<maze.length;r++){
                for(int c=0;c<maze.length;c++){
                System.out.print(path[r][c]+" ");
                }
        }
        System.out.println();
        path[i][j]=0;
        return;
        }
        m(maze,path,i-1,j);
        m(maze,path,i,j-1);
        m(maze,path,i+1,j);
        m(maze,path,i,j+1);
        path[i][j]=0;
        }
```

```
}
```

0/1 Knapsack--

Recursive--

```
public class Solution {

    public static int knapsack(int[] weight,int value[],int maxWeight, int n){
        return knapsack(weight,value,maxWeight,n,0);

    }
        public static int knapsack(int[] weight,int value[],int maxWeight, int n,int i){
        if(i==weight.length){
        return 0;
        }
         if(weight[i]<=maxWeight){
        int ans1=value[i]+knapsack(weight,value,maxWeight-weight[i],n,i+1);
        int ans2=knapsack(weight,value,maxWeight,n,i+1);
        return Math.max(ans1,ans2);
        }else{
        return knapsack(weight,value,maxWeight,n,i+1);
        }
    }
}
```

DP Solution--

https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

```
public class Solution {

    public static int knapsack(int[] weight,int value[],int maxWeight, int n){
        int dp[]=new int[n+1];
        for(int i=0;i<=n;i++){
        dp[i]=-1;
        }
        return knapsack(weight,value,maxWeight,n,0,dp);

    }
```

```
	public static int knapsack(int[] weight,int value[],int maxWeight, int n,int i,int
dp[][]){
	if(i==weight.length){
	return 0;
	}
	 if(weight[i]<=maxWeight){
	int ans1,ans2;
	if(dp[i+1][w]!=-1){
		ans1=dp[i+1];
	}else{
		ans1=value[i]+knapsack(weight,value,maxWeight-weight[i],n,i+1,dp);
		dp[i+1]=ans1;
	}
	if(dp[i+1]!=-1){
		ans2=dp[i+1];
	}else{
		ans2=knapsack(weight,value,maxWeight,n,i+1,dp);
		dp[i+1]=ans2;
	}
	return Math.max(ans1,ans2);
	}else{
	if(dp[i+1]!=-1){
		return dp[i+1];
	}else{
		return knapsack(weight,value,maxWeight,n,i+1,dp);
	}

	}
  }
```