Amazon Question:

How to check if given four points form a square

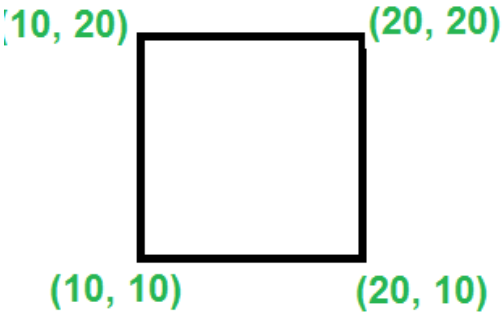
Given coordinates of four points in a plane, find if the four points form a square or not.

To check for square, we need to check for following.

a) All fours sides formed by points are same.

b) The angle between any two sides is 90 degree. (This condition is required as Quadrilateral also has same sides.

c) Check both the diagonals have same distance




```
//How to check if given four points form a square

class Point{
	int x;
	int y;
	Point(int x,int y){
		this.x=x;
		this.y=y;
	}
}
class PointDistanceSquare{
	public static void main(String[] args) {
		Point p1=new Point(0,0);
		Point p2=new Point(4,0);
		Point p3=new Point(0,5);
		Point p4=new Point(5,5);
		boolean flag=validate_square(p1,p2,p3,p4);
		System.out.println(flag);
	}
	public  static boolean validate_square(Point p1,Point p2,Point p3,Point p4){
		double d1=Distance(p1,p2);
		double d2=Distance(p1,p3);
		double d3=Distance(p1,p4);
		System.out.println(d1+" "+d2+" "+d3);
		if(d1==d2 && d3==Math.sqrt(d1*d1+d2*d2)){
```

```
                return true;
            }

            else if(d1==d3 && d2==Math.sqrt(d1*d1+d3*d3)){
                return true;
            }

            else if(d2==d3 && d1==Math.sqrt(d3*d3+d2*d2)){
                return true;
            }
            else{
                return false;
            }
        }
        public static double Distance(Point p1,Point p2){
            int x=Math.abs(p1.x-p2.x);
            int y=Math.abs(p1.y-p2.y);

            double d=Math.sqrt(x*x+y*y);
            System.out.println(d+" distance ");
            return d;
        }

}
```

Check if a string can be obtained by rotating another string 2 places

Given two strings, the task is to find if a string can be obtained by rotating another string two places.

Examples:

```
Input : string1 = "amazon"

            string2 = "azonam"  // rotated anti-clockwise

Output : Yes



Input : string1 = "amazon"

            string2 = "onamaz"  // rotated clockwise

Output : Yes
```

```
//Check if a string can be obtained by rotating another string 2 places
class RotateString{
        public static void main(String[] args) {
            String s1="amazon";
```

```
                String s2="onamaz";
                boolean ans=check_substring(s1,s2);
                System.out.print(ans);


        }
        public static boolean check_substring(String s1,String s2){
                int l=s2.length();
                String clock_string="";
                clock_string+=s2.substring(l-2,l)+s2.substring(0,l-2);
                if(s1.equals(clock_string)){
                        return true;
                }
                String anti_clock_string="";
                anti_clock_string+=s2.substring(2,l)+s2.substring(0,2);
                if(s1.equals(anti_clock_string)){
                        return true;
                }
                return false;


        }

}
```

Find the nearest smaller numbers on left side in an array

Given an array of integers, find the nearest smaller number for every element such that the smaller element is on left side.

Examples:

```
Input:  arr[] = {1, 6, 4, 10, 2, 5}

Output:         {_, 1, 1,  4, 1, 2}

First element ('1') has no element on left side. For 6,

there is only one smaller element on left side '1'.

For 10, there are three smaller elements on left side (1,

6 and 4), nearest among the three elements is 4.



Input: arr[] = {1, 3, 0, 2, 5}

Output:         {_, 1, _, 0, 2}
```

Expected time complexity is O(n).

```
//Check if a string can be obtained by rotating another string 2 places
class RotateString{
	public static void main(String[] args) {
		String s1="amazon";
		String s2="onamaz";
		boolean ans=check_substring(s1,s2);
		System.out.print(ans);


	}
	public static boolean check_substring(String s1,String s2){
		int l=s2.length();
		String clock_string="";
		clock_string+=s2.substring(l-2,l)+s2.substring(0,l-2);
		if(s1.equals(clock_string)){
			return true;
		}
		String anti_clock_string="";
		anti_clock_string+=s2.substring(2,l)+s2.substring(0,2);
		if(s1.equals(anti_clock_string)){
			return true;
		}
		return false;


	}


}
```

Pair with given product | Set 1 (Find if any pair exists)

```
Input : arr[] = {10, 20, 9, 40};
		int x = 400;

Output : Yes



Input : arr[] = {10, 20, 9, 40};
		int x = 190;

Output : No
```

```
import java.util.*;
class ProductofTwo{

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

```
                int[] a={10, 20, 9, 40};
                int x=400;
                HashSet<Integer> hs=new HashSet<Integer>();
                for(int i=0;i<a.length;i++){
                        if(x%a[i]==0){
                                if(hs.contains(x/a[i])){
                                        System.out.println("Pair Exist:"+a[i]+"*"+x/a[i]+"="+x);
                                        break;
                                }
                                hs.add(a[i]);
                        }
                }
        }

}
```

Position of rightmost set bit

Write a one line C function to return position of first 1 from right to left, in binary representation of an Integer.

```
I/P   18,   Binary Representation 010010

O/P   2

I/P   19,   Binary Representation 010011

O/P   1
```

```
class GFG {

        // function to find
        // the rightmost set bit
        static int PositionRightmostSetbit(int n)
        {
                // Position variable initialize
                // with 1 m variable is used to
                // check the set bit
                int position = 1;
                int m = 1;

                while ((n & m) == 0) {

                        // left shift
                        m = m << 1;
                        position++;
                }
                return position;
        }

        // Driver Code
```

```
	public static void main(String[] args)
	{
		int n = 16;

		// function call
		System.out.println(PositionRightmostSetbit(n));
	}
}
```

Replace all ‘0’ with ‘5’ in an input Integer

Given a integer as a input and replace all the ‘0’ with ‘5’ in the integer.

Examples:

```
	102 - 152

	1020 - 1525
```

Use of array to store all digits is not allowed.

```
class Conver0to5digit{
	public static void main(String[] args) {
		int n=0000000;
		if(n==0){
			System.out.println("5");
		}
		else{
			System.out.println(Conversion(n));
		}
	}
	public static int Conversion(int n){
		if(n==0){
			return 0;
		}
		int digit=n%10;
		if(digit==0){
			digit=5;
		}
		return Conversion(n/10)*10+digit;
		//return Integer.parseInt(Integer.toString(n).replace('0','5'));
	}
}
```

Program for array rotation

Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.

Rotation of the above array by 2 will make array


```
static int[] rotLeft(int[] a, int d) {
        int n=a.length;
        int[] b=new int[n];
        for(int i=0;i<n;i++){
            b[i]=a[i];
        }
        for(int i=0;i<n;i++){
            a[(n+i-d)%n]=b[i];
        }
        return a;
    }
```

Calculate the angle between hour hand and minute hand

This problem is know as Clock angle problem where we need to find angle between hands of an analog clock at a given time.

Examples:

Input: h = 12:00, m = 30.00

Output: 165 degree

Input: h = 3.00, m = 30.00

Output: 75 degree

The idea is to take 12:00 (h = 12, m = 0) as a reference. Following are detailed steps.

1) Calculate the angle made by hour hand with respect to 12:00 in h hours and m minutes.

2) Calculate the angle made by minute hand with respect to 12:00 in h hours and m minutes.

3) The difference between two angles is the angle between two hands.

How to calculate the two angles with respect to 12:00? The minute hand moves 360 degree in 60 minute(or 6 degree in one minute) and hour hand moves 360 degree in 12 hours(or 0.5 degree in 1 minute). In h hours and m minutes, the minute hand would move (h*60 + m)*6 and hour hand would move (h*60 + m)*0.5.

```
import java.util.*;
import java.io.*;
class ClockAngle{
	public static void main(String[] args) {
		Scanner in = new Scanner(new BufferedReader(new
InputStreamReader(System.in)));
		int h=in.nextInt();
		int m=in.nextInt();
		System.out.println(Calculate_angle(h,m));
	}
	public static double Calculate_angle(int h,int m){

		double a1=(h*60+m)*0.5;
		double a2=6*m;
		double angle=Math.abs(a1-a2);
		return Math.min(360-angle,angle);
}
}
```

Check if all bits of a number are set

Given a number **n**. The problem is to check whether every bit in the binary representation of the given number is set or not. Here 0 <= **n**.

Examples :

Input : 7

Output : Yes

(7)10 = $(111)_2$

```
Input : 14

Output : No
```

```
import java.io.*;
import java.util.*;
class CheckBits{
	public static void main(String[] args) {
		Scanner in=new Scanner(new BufferedReader(new InputStreamReader(System.in)));

		System.out.println("ENter the number");
		int n=in.nextInt();
		System.out.println(check_validate(n));

	}
	public static String check_validate(int n){
		if(n==0){
			return "No";
		}
		while(n>0){
			if((n&1)==0){
				return "No";
			}
			n=n>>1;
		}
		return "YES";
	}
}
```

Check if a given Binary Tree is SumTree

Write a function that returns true if the given Binary Tree is SumTree else false. A SumTree is a Binary Tree where the value of a node is equal to sum of the nodes present in its left subtree and right subtree. An empty tree is SumTree and sum of an empty tree can be considered as 0. A leaf node is also considered as SumTree.

Following is an example of SumTree.

```
          26

        /   \

      10     3

    /   \     \
```

```
   4    6    3
```

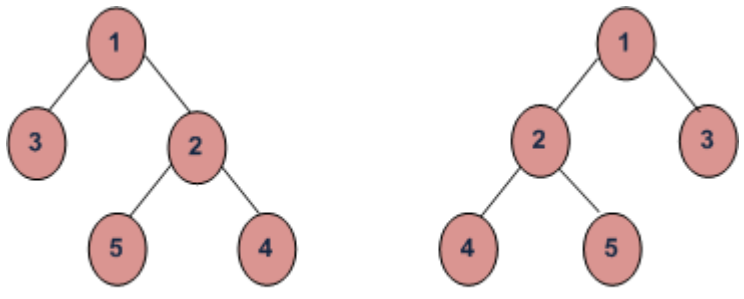
```
import java.io.*;
import java.util.*;

class TreeNode{
        int data;
        TreeNode left;
        TreeNode right;
        TreeNode(int d){
                data=d;
                left=null;
                right=null;
        }
}

class SumTree{
        TreeNode root;
                public static void main(String[] args) {
                Scanner in=new Scanner(new BufferedReader(new InputStreamReader(System.in)));
                SumTree st=new SumTree();
                st.root=new TreeNode(26);
                st.root.left = new TreeNode(10);
      st.root.right = new TreeNode(3);
      st.root.left.left = new TreeNode(4);
      st.root.left.right = new TreeNode(6);
      st.root.right.right = new TreeNode(3);
      boolean flag=st.isSumTree(st.root);
      if(flag){
         System.out.println("The given tree is a sum tree");
      }
      else{
         System.out.println("The given tree is not a sum tree");
      }
   }
   public boolean isSumTree(TreeNode node)
        {

                int ls,rs;
                ls=rs=0;
                if(node==null || isLeaf(node)){
                        return true;
                }
                if(!isSumTree(node.left) && !isSumTree(node.right))
                {
                        if(node.left==null){
                                ls=0;
                        }
                        else if(isLeaf(node.left)){
                                ls=node.left.data;
```

```
                }
                else{
                        ls=2*node.left.data;
                }

                if(node.right==null){
                        rs=0;
                }
                else if(isLeaf(node.right)){
                        rs=node.right.data;
                }
                else{
                        rs=2*node.right.data;
                }


        }
        if(node.data==ls+rs){
                        return true;
                }
                return false;

    }



    public boolean isLeaf(TreeNode node)
    {
        if(node == null)
        {
            return false;
        }
        if(node.left==null && node.right==null)
        {
            return true;
        }
        return false;
    }
}
```

```
class PowerofNum{
	public static void main(String[] args) {
		int n=128;
		System.out.println(power_of(n));
	}
	public static boolean power_of(int num){
		if(num==1){
			return true;
		}
		int m=2;
		while(m<=Math.sqrt(num)){
			int n=2;
			while(n<=Math.sqrt(num)+1){
				int x=(int)Math.pow(m,n);
				if(x==num){
					return true;
				}
				n++;
			}
			m++;
		}
		return false;
	}
}
```

Check if two trees are Mirror

Given two Binary Trees, write a function that returns true if two trees are mirror of each other, else false. For example, the function should return true for following input trees.



Mirror Trees

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

This problem is different from the problem discussed here.

For two trees ‘a’ and ‘b’ to be mirror images, the following three conditions must be true:

1.Their root node’s key must be same

2.Left subtree of root of ‘a’ and right subtree root of ‘b’ are mirror.

3.Right subtree of ‘a’ and left subtree of ‘b’ are mirror.

```
boolean areMirror(Node a, Node b)
    {
        /* Base case : Both empty */
        if (a == null && b == null)
            return true;

        // If only one is empty
        if (a == null || b == null)
            return false;

        /* Both non-empty, compare them recursively
           Note that in recursive calls, we pass left
           of one tree and right of other tree */
        return a.data == b.data
                    && areMirror(a.left, b.right)
                    && areMirror(a.right, b.left);
    }
```

Count number of bits to be flipped to convert A to B

Given two numbers ‘a’ and b’. Write a program to count number of bits needed to be flipped to convert ‘a’ to ‘b’.

Example :

Input : a = 10, b = 20

Output : 4

Binary representation of a is 00001010
Binary representation of b is 00010100

We need to flip highlighted four bits in a

to make it b.

Input : a = 7, b = 10

Output : 3

```
Binary representation of a is 00000111
Binary representation of b is 00001010

We need to flip highlighted three bits in a

to make it b.
```

```
import java.io.*;
import java.util.*;

class FlipBit{
	public static void main(String[] args) {
		Scanner in=new Scanner(new BufferedReader(new InputStreamReader(System.in)));
		while(true){
			int a=in.nextInt();
		int b=in.nextInt();

		System.out.println(Count_bit_to_change(a^b));
		}
	}
	public static int Count_bit_to_change(int n){
		System.out.println(n);

		int flag=0;
		while(n!=0){
			flag+=n&1;
			n=n>>1;
		}
		return flag;
	}
}
```

Count number of occurrences (or frequency) in a sorted array

Given a sorted array arr[] and a number x, write a function that counts the occurrences of x in arr[]. Expected time complexity is O(Logn)

Examples:

```
Input: arr[] = {1, 1, 2, 2, 2, 2, 3,},   x = 2

Output: 4 // x (or 2) occurs 4 times in arr[]
```

Input: arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 3

Output: 1

Input: arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 1

Output: 2

Input: arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 4

Output: -1 // 4 doesn't occur in arr[]

```
class CountOccurance{
	public static void main(String[] args) {
			int[] a={1,3,7,9,9,9,9,14,19,22};
			System.out.println(Count_Occurance(a,9));
		}

		public static int Count_Occurance(int[] a,int x){
			int m=a.length;
			int ind=Find_index(a,x,0,a.length-1);
			if(ind==-1){
				return 0;
			}
			int flag=1;
			int left=ind-1;
			while(left>=0 && a[left]==x){
				flag++;
				left--;
			}
			int right=ind+1;
			while(right<m && a[right]==x){
				flag++;
				right++;
			}
			return flag;
		}
		public static int Find_index(int[] a,int x,int start,int end){
			if(start>end){
				return -1;
			}
```

```
                int mid=start+(end-start)/2;
                if(a[mid]==x){
                        return mid;
                }
                if(a[mid]<x){
                        return Find_index(a,x,start,mid-1);
                }
                return Find_index(a,x,mid+1,end);
        }
}
```

Count all possible paths from top left to bottom right of a mXn matrix

Note the count can also be calculated using the formula (m-1 + n-1)!/(m-1)!(n-1)!.

Equilibrium index of an array

Equilibrium index of an array is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. For example, in an array A:

Example :

```
Input : A[] = {-7, 1, 5, 2, -4, 3, 0}

Output : 3

3 is an equilibrium index, because:

A[0] + A[1] + A[2]  =  A[4] + A[5] + A[6]
```

Write a function int equilibrium(int[] arr, int n); that given a sequence arr[] of size n, returns an equilibrium index (if any) or -1 if no equilibrium indexes exist.

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

                int[] a={-7, 1, 5, 2, -4, 3, 0};
                int ind=Find_index(a);
                System.out.println(ind);

        }
        public static int Find_index(int[] a){
                int sum=0;
                int left_sum=0;
                for(int i=0;i<a.length;i++){
                        sum+=a[i];
```

```
            }

            for(int i=0;i<a.length;i++){
                sum-=a[i];

                if(sum==left_sum){
                    return i+1;
                }
                left_sum+=a[i];
            }
            return -1;
        }

}
```

Find first and last positions of an element in a sorted array

Given a sorted array with possibly duplicate elements, the task is to find indexes of first and last occurrences of an element x in the given array.

Examples:

```
Input : arr[] = {1, 3, 5, 5, 5, 5 ,67, 123, 125}

            x = 5

Output : First Occurrence = 2

              Last Occurrence = 5



Input : arr[] = {1, 3, 5, 5, 5, 5 ,7, 123 ,125 }

            x = 7

Output : First Occurrence = 6

              Last Occurrence = 6
```

```
class FirstAndLast{
        public static void main(String[] args) {
                int a[] = {1, 2, 2, 2, 2, 3, 4, 7, 8, 8};
                int x=2;
                int first=FindFirst(a,x,0,a.length-1,a.length);
                int last=FindLast(a,x,0,a.length-1,a.length);
                System.out.println(first+" "+last);
        }
```

```
	public static int FindFirst(int[] a,int x,int start,int end,int n){
		if(end>=start){
			int mid=start+(end-start)/2;
			if((mid==0 || x>a[mid-1]) && a[mid]==x){
				return mid;
			}
			else if(x>a[mid]){

				return FindFirst(a,x,mid+1,end,n);
			}
			else{
				return FindFirst(a,x,start,mid-1,n);
			}
		}
		return -1;
	}
	public static int FindLast(int[] a,int x,int start,int end,int n){
		if(end>=start){
			int mid=start+(end-start)/2;
			if((mid==n-1 || x<a[mid+1]) && a[mid]==x){
				return mid;
			}
			else if(x<a[mid]){

				return FindFirst(a,x,mid+1,end,n);
			}
			else{
				return FindFirst(a,x,mid+1,end,n);
			}
		}
		return -1;
	}

}
```

Find four elements that sum to a given value | Set 1 (n^3 solution)

Given an array of integers, find all combination of four elements in the array whose sum is equal to a given value X.
For example, if the given array is {10, 2, 3, 4, 5, 9, 7, 8} and X = 23, then your function should print “3 5 7 8” (3 + 5 + 7 + 8 = 23).

```
import java.util.*;
class FourthSum{
	public static void main(String[] args) {
		int[] a = {1, 4, 45, 6, 10, 12};
		int x=21;
		Find_El(a,x);

	}
	public static void Find_El(int[] a,int x){
```

```
		Arrays.sort(a);
		int n=a.length;
		for(int i=0;i<n-3;i++){
			for(int j=i+1;j<n-2;j++){
				int l=j+1;
				int r=n-1;
				while(l<=r){
					//System.out.println(a[i]+" "+a[j]+" "+a[l]+" "+a[r]);
					if(a[i]+a[j]+a[l]+a[r]==x){
						System.out.println(a[i]+" "+a[j]+" "+a[l]+"
"+a[r]);

						l++;
						r--;
						break;
					}
					else if(a[i]+a[j]+a[l]+a[r]>x){
						r--;
					}
					else{
						l++;
					}
				}
			}
		}
	}
}
```

k largest(or smallest) elements in an array | added Min Heap method

Question: Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

```
//k largest(or smallest) elements in an array | added Min Heap method
class KthLargest{
	public static void main(String[] args) {
		int[] a={1, 23, 12, 9, 30, 2, 50};
		int k=0;
		int j=0;
		int n=a.length;
		for(int i=0;i<n-1;i++){
			for(j=0;j<n-i-1;j++){
				if(a[j]>a[j+1]){
					int temp=a[j];
					a[j]=a[j+1];
					a[j+1]=temp;
				}

			}
```

```
                if(k<3){
                        System.out.println(a[j]+" ");
                        k++;
                }
        }

    }
}
```

Find the index of first 1 in a sorted array of 0’s and 1’s

Given a sorted array consisting 0’s and 1’s. The problem is to find the index of first ‘1’ in the sorted array. It could be possible that the array consists of only 0’s or only 1’s. If 1’s are not present in the array then print “-1”.

Examples :

Input : arr[] = {0, 0, 0, 0, 0, 0, 1, 1, 1, 1}

Output : 6

The index of first **1** in the array is 6.

Input : arr[] = {0, 0, 0, 0}

Output : -1

1's are not present in the array.

```
class FirstOne{
    public static void main(String[] args) {
        int[] a={0, 0, 0, 0, 0, 0, 1, 1, 1, 1};
        int x=help(a);
        System.out.println(x);
    }
    public static int help(int[] a){
        int low=0;
        int high=a.length-1;
        while(low<high){
            int mid=low+(high-low)/2;
            if((a[mid]==1) && (mid==0 || a[mid-1]==1)){
                return (int)mid;
            }
            else if(a[mid]==0){
                low=mid+1;
            }
            else{
                high=mid-1;
```

```
                }
            }
            return -1;
        }
}
```

Find minimum difference between any two elements

Given an unsorted array, find the minimum difference between any pair in given array.

Examples :

```
Input  : {1, 5, 3, 19, 18, 25};

Output : 1

Minimum difference is between 18 and 19



Input  : {30, 5, 20, 9};

Output : 4

Minimum difference is between 5 and 9



Input  : {1, 19, -4, 31, 38, 25, 100};

Output : 5

Minimum difference is between 1 and -4
```

```
//Find minimum difference between any two elements
import java.util.*;
class MinDifference{
        public static void main(String[] args) {
                int a[]={1, 5, 3, 19, 18, 25};
                int n=a.length;
                Arrays.sort(a);
                int min=Integer.MAX_VALUE;
                for(int i=0;i<n-1;i++){
                        min=Math.min(Math.abs(a[i]-a[i+1]),min);
                }
                System.out.println(min);
        }
}
```

Count the number of possible triangles

Given an unsorted array of positive integers. Find the number of triangles that can be formed with three different array elements as three sides of triangles. For a triangle to be possible from 3 values, the sum of any two values (or sides) must be greater than the third value (or third side).
For example, if the input array is {4, 6, 3, 7}, the output should be 3. There are three triangles possible {3, 4, 6}, {4, 6, 7} and {3, 6, 7}. Note that {3, 4, 7} is not a possible triangle.
As another example, consider the array {10, 21, 22, 100, 101, 200, 300}. There can be 6 possible triangles: {10, 21, 22}, {21, 100, 101}, {22, 100, 101}, {10, 100, 101}, {100, 101, 200} and {101, 200, 300}

Method 2 (Tricky and Efficient)
Let a, b and c be three sides. The below condition must hold for a triangle (Sum of two sides is greater than the third side)
i) a + b > c
ii) b + c > a
iii) a + c > b

Following are steps to count triangle.

1. Sort the array in non-decreasing order.

2. Initialize two pointers ‘i’ and ‘j’ to first and second elements respectively, and initialize count of triangles as 0.

3. Fix ‘i’ and ‘j’ and find the rightmost index ‘k’ (or largest ‘arr[k]’) such that ‘arr[i] + arr[j] > arr[k]’. The number of triangles that can be formed with ‘arr[i]’ and ‘arr[j]’ as two sides is ‘k – j’. Add ‘k – j’ to count of triangles.
Let us consider ‘arr[i]’ as ‘a’, ‘arr[j]’ as b and all elements between ‘arr[j+1]’ and ‘arr[k]’ as ‘c’. The above mentioned conditions (ii) and (iii) are satisfied because ‘arr[i] < arr[j] < arr[k]'. And we check for condition (i) when we pick 'k'

4. Increment ‘j’ to fix the second element again.

Note that in step 3, we can use the previous value of ‘k’. The reason is simple, if we know that the value of ‘arr[i] + arr[j-1]’ is greater than ‘arr[k]’, then we can say ‘arr[i] + arr[j]’ will also be greater than ‘arr[k]’, because the array is sorted in increasing order.

5. If ‘j’ has reached end, then increment ‘i’. Initialize ‘j’ as ‘i + 1’, ‘k’ as ‘i+2’ and repeat the steps 3 and 4.