## 1. Longest Palindrome Substring

Leetcode 5, longest Palindromic Substring

```
    public String longestPalindromicSubstring(String inputStream){
        if(inputStream.length() <= 1) return inputStream;

        int n= inputStream.length();
        int maxLen = 1, start = 0, end = 0;
        boolean[][] dp = new boolean[n][n];
        for(int i=0; i<n; i++) dp[i][i]=true;

        for(int i=n-2; i>=0; i--){
            for(int j=i+1; j<n; j++){
                if(j==i+1) {
                    dp[i][j] = inputStream.charAt(i) == inputStream.charAt(j);
                    if(dp[i][j] && j-i+1 > maxLen){
                        maxLen = j-i+1;
                        start = i;
                        end=j;
                    }
                }
                else {
                    if(dp[i+1][j-1] && inputStream.charAt(i)==inputStream.charAt(j)){
                        if(j-i+1 > maxLen){
                            maxLen = j-i+1;
                            start = i;
                            end=j;
                        }
                        dp[i][j]=true;
                    }
                }
            }
        }
        return inputStream.substring(start, end+1);
```

## 2. Most Frequent Used Words

Leetcode 819

```
  public String[] mostCommonWord(String paragraph, String[] banned) {
        if(paragraph.length()==0) return new String[0];

        HashMap<String, Integer> map = new HashMap<>();
        paragraph = paragraph + ".";
        paragraph = paragraph.toLowerCase();
        String tmp = "";
        int maxApp = 0;
        HashSet<String> ban = new HashSet<>();
        for(String s : banned){
            ban.add(s.toLowerCase());
        }

        for(int i=0; i<paragraph.length(); i++){
            if(paragraph.charAt(i)>='a' && paragraph.charAt(i)<='z') {
                tmp=tmp+paragraph.charAt(i);

            }   else {
                if(ban.contains(tmp) || tmp.equals("")) { // if "", continue
                    tmp="";
                    continue;
                } else if(map.containsKey(tmp)){
                    map.put(tmp, map.get(tmp)+1);
                } else {
                    map.put(tmp, 1);
                }
                if(map.get(tmp) > maxApp) maxApp = map.get(tmp);
                tmp="";
            }
        }
        String[] res = new String[maxApp];
        int index = 0;
        for(Map.Entry<String, Integer> entry : map.entrySet()){

            if(entry.getValue() == maxApp){
                res[index] = entry.getKey();
                index++;
            }
        }
        return res;

    }
```

## 3. Count Number of substrings with exactly k distinct characters

https://www.geeksforgeeks.org/count-number-of-substrings-with-exactly-k-distinct-characters/

Brute force, check each substring from i to j.

```
    public int countkDist(String s, int k) {
        if(s==null || s.length() < k) return 0;
        int count = 0;
        int n=s.length();
        char[] array = s.toCharArray();
        for(int i=0; i<n; i++){
            int[] label = new int[26];
            int distCount = 0;
            for(int j=i; j<n; j++){
                if(label[array[j]-'a']==0) distCount++;
                label[array[j]-'a']++;
                if(distCount==k) count++;
                else if(distCount>k) break;
            }
        }
        return count;
    }
```

## 4. Maximum Minimum Path

You are gonna climb mountains represented by a matrix. Each integer in the matrix represents the altitude at that point. You are supposed to climb from the top-left corner to the bottom-right corner and only move rightward or downward in each step. You can have many paths to do so. Each path has a minimum altitude. Find the maximum among all the minimum altitudes of all paths. e.g.

```
[[5 4 5],[1 3 6]]
```

Three paths: 5 1 3 6,5 4 3 6,5 4 5 6. Minimums are 1, 3, 4 respectively. Return the maximum in them which is 4. another example:

```
[[8, 4, 7][6, 5, 9]]
```

3 paths:

```
8-4-7-9 min: 4
8-4-5-9 min: 4
8-6-5-9 min: 5
return: 5
```

DP solution

```
public int minMaxPath(int[][] matrix)
{
    if(matrix.length==0 || matrix[0].length==0) return Integer.MIN_VALUE;
    int m = matrix.length, n = matrix[0].length;
    int[][] dp = new int[m][n];
    dp[0][0] = matrix[0][0];
    for(int i=1; i<m; i++) dp[i][0] = Math.min(matrix[i][0], dp[i-1][0]);
    for(int j=1; j<n; j++) dp[0][j]= Math.min(matrix[0][j], dp[0][j-1]);

    for(int i=1; i<m; i++){
        for(int j=1; j<n; j++){
            if(matrix[i][j] > dp[i-1][j] && matrix[i][j] > dp[i][j-1]) dp[i][j] =
Math.max(dp[i-1][j], dp[i][j-1]);
            else dp[i][j] = matrix[i][j];
        }
    }
    return dp[m-1][n-1];
}
```

DFS solution

```
public class Main {
    static int minmax = Integer.MIN_VALUE;
    public static int minMaxPath(int[][] matrix)
    {
        dfsHelper(matrix, 0, 0, Integer.MAX_VALUE);
        return minmax;
    }
    public static void dfsHelper(int[][] matrix, int i, int j, int min){
        if(i>=matrix.length || j >=matrix[0].length) return;
        min = Math.min(min, matrix[i][j]);
        if(i == matrix.length-1 && j==matrix[0].length-1) {
            minmax = Math.max(min, minmax);
            return;
        }
        dfsHelper(matrix, i+1,  j, min);
        dfsHelper(matrix, i, j+1, min);
    }
}
```

## 5. Two sum closest

Given two sorted arrays and a number x, find the pair whose sum is closest to x and << x and **the pair has an element from each array**.

We are given two arrays ar1[0…m-1] and ar2[0..n-1] and a number x, we need to find the pair ar1[i] + ar2[j] such that absolute value of (ar1[i] + ar2[j] – x) is minimum.

```
Input:  ar1[] = {1, 4, 5, 7};
        ar2[] = {10, 20, 30, 40};
        x = 32
Output:  1 and 30

Input:  ar1[] = {1, 4, 5, 7};
        ar2[] = {10, 20, 30, 40};
        x = 50
Output:  7 and 40
```

2 pointers in sorted arrays.

```
public static int[] closesTwoSum(int ar1[], int ar2[], int x)
{
    Arrays.sort(ar1);
    Arrays.sort(ar2);
    int n1 = ar1.length, n2=ar2.length;
    int ptr1 = 0, ptr2= n2-1;

    int minDist = Integer.MAX_VALUE;
    int res1=Integer.MAX_VALUE, res2= Integer.MAX_VALUE;

    while(ptr1<n1 && ptr2>=0){
        int tmpDist = x - ar1[ptr1] - ar2[ptr2];
        if(tmpDist>0 && tmpDist<minDist){
            minDist = tmpDist;
            res1 = ar1[ptr1];
            res2 = ar2[ptr2];
        }
        if(tmpDist > 0){
            ptr1++;
        } else {
            ptr2--;
        }
    }
    int[] res = new int[]{res1, res2};
    return res;
}
```

## 6. Subtree Maximum average node

Description Given a binary tree, find the subtree with maximum average. Return the root of the subtree. Example Given a binary tree: 1 / \ -5 11 / \ / \ 1 2 4 -2 return the node 11. Your problem can be different in ways like -- it may not be a binary tree or the average doesn't include the root of the substree. reference code:

```
class ResultType {
    TreeNode node;
    int sum;
    int size;
    public ResultType(TreeNode node, int sum, int size) {
        this.node = node;
        this.sum = sum;
        this.size = size;
    }
}
private ResultType result = null;
public TreeNode findSubtree2(TreeNode root) {
// Write your code here
if (root == null) {
    return null;
}
ResultType rootResult = helper(root);
    return result.node;
}
public ResultType helper(TreeNode root) {
    if (root == null) {
    return new ResultType(null, 0, 0);
}
    ResultType leftResult = helper(root.left);
    ResultType rightResult = helper(root.right);
    ResultType currResult = new ResultType(
    root,
    leftResult.sum + rightResult.sum + root.val,
    leftResult.size + rightResult.size + 1);
```

```
        if (result == null || currResult.sum * result.size > result.sum * currResult.size)
    {
            result = currResult;
        }
        return currResult;
    }
```

## 7. K minimum Distances

A 2D matrix with 0, 1, 9. Get minimum number of steps to achieve 9.

0 -- obstable

1--path

9--gold

```
public static int removeObstacle(int m, int n, int[][] lot){
    helper(lot, 0, 0, 0);
    return res;
}

static int res = Integer.MAX_VALUE;

public static void helper(int[][] lot, int i, int j, int step){
    if(i<0 || j<0 || i>=lot.length || j>=lot[0].length || lot[i][j]==0) return;

    if(lot[i][j] == 9) {
        res = Math.min(step, res);
        return;
    }
    lot[i][j]=0;
    helper(lot, i-1, j, step+1);
    helper(lot, i+1, j, step + 1);
    helper(lot, i, j-1, step+1);
    helper(lot, i, j+1, step+1);
    lot[i][j] = 1;

}
```

## 8. K Nearest Points

Load goods into a truck. Truck is at the origin and you are given the coordinates of all goos, find the k nearest goods. Use PriorityQueue. Write your own Comparator.

```
/*
public class Point {
    public int x;
    public int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
*/

public List<Point> findKClosest(Point[] p, int k) {
        PriorityQueue<Point> pq = new PriorityQueue<>(new Comparator<Point>() {
            @Override
            public int compare(Point a, Point b) {
                    return (b.x * b.x + b.y * b.y) - (a.x * a.x + a.y * a.y);
            }
        });

        for (int i = 0; i < p.length; i++) {
            if (i < k)
            pq.offer(p[i]);
            else {
                Point temp = pq.peek();
                if ((p[i].x * p[i].x + p[i].y * p[i].y) - (temp.x * temp.x + temp.y *
temp.y) < 0) {
                    pq.poll();
                    pq.offer(p[i]);
                    }
                }
        }

        List<Point> x = new ArrayList<>();
        while (!pq.isEmpty())
            x.add(pq.poll());

        return x;
}
```

## 9. Flight

Amazon Prime Air is developing a system that divides shipping routes using flight optimization routing stems to a cluster of aircraft that can fulfill these routes. Each shipping route is identified by a unique integer identifier, requires a fixed non-zero amount of travel distance between airports, and is defined to be either a forward shipping route or a return shipping route. Identifiers are guaranteed to be unique within their own route type, but not across route types.

Each aircraft should be assigned two shipping routes at once: one forward route and one return route. Due tot the complex scheduling of flight plans, all aircraft have a fixed maximum operating travel distance, and cannot be scheduled to flying a shipping route that requires more travel distance than the prescribed maximum operating travel distance. The goal of the system is to optimize the total operating travel distance of given aircraft. A forward/return shipping route pair is considered to be optimal if there does not exist another pair that has higher operating travel distance than this pair, and also has a total less than or equal to the maximum operating travel distance of the aircraft. For example, if the aircraft has a maximum operating travel distance of 3000miles, a forward/return shipping route pairs using a total of 2900miles would be optimal if there does not exist a pair that uses a total operating travel distance of 3000miles, but would not be considered optimal if such a pair did exist.

Your task is to write an algorithm to optimize the set of forward/return shipping route pairs that allow the aircraft to be optimally utilized, given a list of forward shipping routes and a list of return shipping routes.

Input

1. maximumOperatingTravelDistance, an integer representing the maximum operating travel distance of the given aircraft.
2. forwardShipping RouteList, a list of pairs of integers where the first integer represents the unique identifier of a forward shipping route and the second integeter represent the amount of travel distance required by shipping route
3. returnShippingRouteList, a list of pairs of integers where the first integer represents the unique identifier of a forward shipping route and the second integeter represent the amount of travel distance required by shipping route

output

Return a list of pairs of integer representing the pairs of IDs of forward and return shipping routes that optimal utilize the given aircraft. If no route is possible, return an empty list.

**Examples:**

*Example 1:*

inputs: maximumOperatingTravelDistance=7200

forwardShippingRouteList = [[1,2000],[2,4000],[3,6000]]

ReturnShippingRouteList=[1,2000]

output[[2, 1]]

*Example 2:*

Inputs: maximumOperatingTravelDistance = 1000

forwardShippingRouteList = [[1, 3000], [2, 2000], [3, 7000], [4, 10000]]

returnShippingRouteList = [[1, 2000], [2,3000], [3, 4000], [4, 5000]]

Ouput:

[[2,4],[3,2]]

## 10 [LintCode] 613 High Five 解题报告

**Description** There are two properties in the node student id and scores, to ensure that each student will have at least 5 points, find the average of 5 highest scores for each person.

**Example** Given results = [[1,91],[1,92],[2,93],[2,99],[2,98],[2,97],[1,60],[1,58],[2,100],[1,61]]

Return

思路 根据题意，对于每一个id，我们维护一个大小为K的min-heap。一个一个把Record放进去，如果容量超了，就把最小的踢掉。这样heap里永远是最大的K个。 全部放完以后，对于每一个id，我们把heap里的Record拿出来算一下平均数

```
/**
 * Definition for a Record
 * class Record {
 *     public int id, score;
 *     public Record(int id, int score){
 *         this.id = id;
 *         this.score = score;
 *     }
 * }
 */
public class Solution{
    /**
     * @param results a list of <student_id, score>
     * @return find the average of 5 highest scores for each person
     * Map<Integer, Double> (student_id, average_score)
     */

    public Map<Integer, Double> highFive(Record[] results) {
        // Write your code here

        Map<Integer, Double> result = new HashMap<Integer, Double>();
        HashMap<Integer, PriorityQueue<Record>> map = new HashMap<Integer,
PriorityQueue<Record>>();

        int k = 5;
        for (Record r : results) {
            if (!map.containsKey(r.id)) {
                PriorityQueue<Record> pq = new PriorityQueue<Record>(k, new
Comparator<Record>() {
                    @Override
                    public int compare(Record a, Record b) {
                        return a.score - b.score; // min-heap
                    }
                });
                map.put(r.id, pq);
            }

            map.get(r.id).add(r);
            if (map.get(r.id).size() > k) {
                map.get(r.id).poll();
            }
        }

        for (Map.Entry<Integer, PriorityQueue<Record>> entry : map.entrySet()) {
            int id = entry.getKey();
            PriorityQueue<Record> pq = entry.getValue();
            double average = 0;
            int num = pq.size();
            while (!pq.isEmpty()) {
                average += pq.poll().score;
            }
            average /= num;
            result.put(id, average);
        }

        return result;
    }
}
```

## 11 Robot

Leetcode 505, the Maze II

There is a **ball** in a maze with empty spaces and walls. The ball can go through empty spaces by rolling **up**, **down**, **left** or **right**, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the ball's **start position**, the **destination** and the **maze**, find the shortest distance for the ball to stop at the destination. The distance is defined by the number of **empty spaces** traveled by the ball from the start position (excluded) to the destination (included). If the ball cannot stop at the destination, return -1.

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The start and destination coordinates are represented by row and column indexes.

**Example 1:**

```
Input 1: a maze represented by a 2D array

0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (4, 4)

Output: 12

Explanation: One shortest way is : left -> down -> left -> down -> right -> down ->
right.
             The total distance is 1 + 1 + 3 + 1 + 2 + 2 + 2 = 12.
```

**Example 2:**

```
Input 1: a maze represented by a 2D array

0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
```

```
1 1 0 1 1
0 0 0 0 0

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (3, 2)

Output: -1

Explanation: There is no way for the ball to stop at the destination.
```

```java
public int shortestDistance(int[][] maze, int[] start, int[] destination) {
    if(maze==null || maze.length==0 || maze[0].length==0) return -1;

    int[][] distance = new int[maze.length][maze[0].length];

    for(int[] row : distance)
        Arrays.fill(row, Integer.MAX_VALUE);
    distance[start[0]][start[1]] = 0;
    helper(maze, start, distance);
    if (distance[destination[0]][destination[1]]==Integer.MAX_VALUE) return -1;
    return distance[destination[0]][destination[1]];
}

public void helper(int[][] maze, int[] start, int[][] distance){
    int[][] dirs = new int[][]{{0, 1}, {0, -1}, {1, 0}, {-1, 0}};

    for(int[] dir : dirs){
        int x = start[0] + dir[0];
        int y = start[1] + dir[1];
        int count = 0;
        while(x>=0 && x<maze.length && y>=0 && y<maze[0].length && maze[x][y]==0){
            x += dir[0];
            y += dir[1];
            count++;
        }
        if(distance[start[0]][start[1]] + count < distance[x-dir[0]][y-dir[1]]){
            distance[x-dir[0]][y-dir[1]] = distance[start[0]][start[1]] + count;
            helper(maze, new int[]{x-dir[0], y-dir[1]}, distance);
        }

    }
}
```

## 13 Movies in flight

(closet two sum)

```java
public static int twoSumClosest(int[] nums, int target) {
    int diff = Integer.MAX_VALUE;
    if(nums.length==0) return diff;

    Arrays.sort(nums);
    int lo = 0, hi = nums.length-1;
    while(lo < hi){

        if(nums[lo] + nums[hi] > target) {

            hi--;
        } else {
            diff = Math.min(diff, target - nums[lo] - nums[hi]);
            lo++;
        }
    }
    return diff;
}
```

## 14. Reorder Log Files

Give a log, consisting of List< String >, each element representing one line of logs. Each line of log information is separated by a space. The first is the ID of the log, followed by the log content. There are two ways to make content:

1. All consist of letters and spaces.
2. All consist of numbers and spaces. Now that the logs are sorted, it is required that component 1 be sorted in order of content lexicography and placed at the top, and component 2 should be placed at the bottom and output in the order of input. (Note that the space also belongs to the content, and when the lexicographic order of the composition method 1 is equal, sort according to the dictionary order of log ID., and the guarantee ID is not repeated)

```
public String[] logSort(String[] logs) {
    Arrays.sort(logs, (str1, str2) -> {
        int contentStart1 = str1.indexOf(" ");
        int contentStart2 = str2.indexOf(" ");
        boolean isDigit1 = Character.isDigit(str1.charAt(contentStart1 + 1));
        boolean isDigit2 = Character.isDigit(str2.charAt(contentStart2 + 1));
        if (isDigit1 && isDigit2) {
            return 0;
        }
        if (isDigit1) {
            return 1;
        }
        if (isDigit2) {
            return -1;
        }
        int compare = str1.substring(contentStart1 +
1).compareTo(str2.substring(contentStart2 + 1));
        if (compare != 0) {
            return compare;
        }
        return str1.substring(0, contentStart1).compareTo(str2.substring(0,
contentStart2));
```

```
        });
        return logs;
    }
```

## 15. Given an array of letters and a window size k, return subarrays of size k with no duplicates.

e.g. [a, d, f, g, k ,g] and window size k=4

return [[a,d,f,g], [d,f,g,k]]

check #3

https://www.geeksforgeeks.org/count-number-of-substrings-with-exactly-k-distinct-characters/

## 16. MST (Minimum spanning tree)

Coonnect all cities with least cost.

Union found

https://www.ctolib.com/topics-80931.html

```
//给好的connection class，两个城市名，和一个cost。
import java.util.*; //这句话极度重要
class Connection{
    String node1;
    String node2;
    int cost;
    public Connection(String a, String b, int c){
        node1 = a;
        node2 = b;
        cost = c;
    }
}
//下面进入正题
public class City_Connections {
private static int unionNum;//这里开个全局变量，不丢人。
//这个static是题目要求的，我自己一般不写，累。
public static ArrayList<Connection> getLowCost(ArrayList<Connection> connections){
    //先把空的情形挡掉
    if (connections == null || connections.size() == 0){
        return new ArrayList<>();
    }

    ArrayList<Connection> result = new ArrayList<>();
    Map<String, Integer> map = new HashMap<>();
    //这里把cost小的往前排。
    Collections.sort(connections, new Comparator<Connection>() {
        @Override
        public int compare(Connection o1, Connection o2) {
            return o1.cost - o2.cost;
        }
    });
```

```
        unionNum = 0;
        for (Connection c : connections){
            String a = c.node1;
            String b = c.node2;
            //看城市是不是连过了，要是可以连，那么就在result里面加上
            if (union(map, a, b)){
                result.add(c);
            }
        }
        //这里要检查一下,是不是所有的城市都属于同一个union
        String find = connections.get(0).node1;
        int union = map.get(find);
        for (String str : map.keySet()){
            // 如果我们中出了一个叛徒，返回空表
            if (map.get(str) != union){
                return new ArrayList<>();
            }
        }
        //这里最后要求按照城市的名字排序
        Collections.sort(result, new Comparator<Connection>() {
            @Override
            public int compare(Connection o1, Connection o2) {
                if(o1.node1.equals(o2.node1)){
                    return o1.node2.compareTo(o2.node2);
                }
                return o1.node1.compareTo(o2.node1);
            }
        });
        return result;
    }
    private static boolean union(Map<String, Integer> map, String a, String b){
        if (!map.containsKey(a) && !map.containsKey(b)){
            map.put(a, unionNum);
            map.put(b, unionNum);
            //这里用了一个新的没用过的数字
            unionNum++;
            return true;
        }
        //只有一方落单,那就加入有组织的一方
        if (map.containsKey(a) && !map.containsKey(b)){
            int aU = map.get(a);
            map.put(b, aU);
            return true;
        }
        if (!map.containsKey(a) && map.containsKey(b)){
            int bU = map.get(b);
            map.put(a, bU);
            return true;
        }
        //两个人都有团伙的情况。
        int aU = map.get(a);
        int bU = map.get(b);
        //如果是自己人,那肯定要踢掉,否则成环了
        if(aU == bU) return false;
        //把所有对方的团伙都吃进来
        for (String s : map.keySet()) {
            if (map.get(s) == bU) map.put(s, aU);
        }
        return true;
    }
    public static void main(String[] args) {
        ArrayList<Connection> connections = new ArrayList<>();
        //下面的图是个苯环，中间加上了几根，如果想验证空表，去掉几根线就行。
        connections.add(new Connection("A","B",6));
        connections.add(new Connection("B","C",4));
        connections.add(new Connection("C","D",5));
        connections.add(new Connection("D","E",8));
        connections.add(new Connection("E","F",2));
        connections.add(new Connection("B","F",10));
        connections.add(new Connection("E","C",9));
        connections.add(new Connection("F","C",7));
        connections.add(new Connection("B","E",3));
        connections.add(new Connection("A","F",16));
        //这里就输出一下看看结果。
        ArrayList<Connection> res = getLowCost(connections);
        for (Connection c : res){
            System.out.println(c.node1 + " -> " + c.node2 + " " + c.cost);
        }
    }
```

## 17 length K substring with k-1 distinct characters

Solution 1:

```
public static List<String> subStringLessKDist(String inputString, int num){
    List<String> res = new ArrayList<>();
    if(inputString==null || inputString.length()==0 || num<=1 || inputString.length()
<=num) return res;
    HashSet<String> resSet = new HashSet<>();
    for(int i=0; i<inputString.length() - num + 1; i++) {
        String tmp = inputString.substring(i,  i+num);
        char[] arr = tmp.toCharArray();
        HashSet<Character> tmpSet = new HashSet<>();
        for(char c : arr) tmpSet.add(c);
        if(tmpSet.size()==num-1) resSet.add(tmp);
    }
    Iterator itr = resSet.iterator();
    while(itr.hasNext()) {
        res.add((String)itr.next());
    }
    return res;
}
```

Solution 2:

```
public static List<String> findKMinusOneDistinct(String inputString, int k) {

        Map<Character, Integer> occurrenceMap = new HashMap<>();
        List<String> resultList = new ArrayList<>();

        for (int i = 0; i + k <= inputString.length(); i++) {

            String str = inputString.substring(i, i + k);
            boolean isRepeat = false;

            for (char c : str.toCharArray()) {
                if (occurrenceMap.containsKey(c)) {
                    if (!isRepeat)
                        occurrenceMap.put(c, occurrenceMap.get(c) + 1);
                    else
                        break;

                    isRepeat = true;
                } else
                    occurrenceMap.put(c, 1);
            }
            //if it makes it through and has precisely 1 repeat character
            if (isRepeat)
                resultList.add(str);
            //empty the map
            occurrenceMap.clear();
        }
        return resultList;
    }
```

## 18 Closet Two Sum (single array)

Largest one that smaller than target.

```
Class Solution{
    public int[] twoSum(int[] numbers, int target){
        Arrays.sort(numbers);
        int low = 0, high = numbers.length-1;
        int min = Integer.MAX_VALUE;
        while(low < high){
            int cur = numbers[low] + numbers[high];
            int diff = target - cur;
            if(diff < d && diff >=0){
                min = diff;
                res[0] = numbers[low];
                res[1] = numbers[high];
            }
            if(cur < target){
                low++;
            } else {
                high--;
            }
        }
```

```
            return res;
        }
    }
```