# Answer of 1A black background with a black square Description automatically generated with medium confidenceA black background with red arrows Description automatically generated

A black background with a black square

Description automatically generated with medium confidence

# Answer of 2A black background with red arrows Description automatically generatedA black background with red arrows Description automatically generatedA black background with a black square Description automatically generatedA black background with a black square Description automatically generated with medium confidence

# Answer of 3A black background with red lines Description automatically generatedA black background with red lines Description automatically generatedA black background with red lines Description automatically generatedA black background with red lines Description automatically generatedA red line in the dark Description automatically generatedA black background with red lines Description automatically generatedA black background with red lines Description automatically generated

# Answer of 4

class Solution {

static class Point {

public int x;

public int y;

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public double distanceFromOrigin() {

return Math.sqrt(x \* x + y \* y);

}

public int[] toArray() {

return new int[] { x, y };

}

}

public int[][] kClosest(int[][] points, int k) {

PriorityQueue<Point> pq = new PriorityQueue<>(

(p1, p2) -> Double.compare(p2.distanceFromOrigin(), p1.distanceFromOrigin()));

for (var point : points) {

pq.add(new Point(point[0], point[1]));

if (pq.size() > k) {

pq.poll();

}

}

int[][] result = new int[k][2];

int index = 0;

while (!pq.isEmpty()){

result[index] = pq.poll().toArray();

index++;

}

return result;

}

}

# Answer of 5

class Solution {

static class CharacterFrequency{

public char character;

public int frequency;

public CharacterFrequency(char character, int frequency ){

this.character = character;

this.frequency = frequency;

}

public int getFrequency(){

return this.frequency;

}

}

public String frequencySort(String s) {

Map<Character, Integer> map = new HashMap<>();

// count frequency

for (var c: s.toCharArray()){

map.put(c, map.getOrDefault(c, 0) + 1 );

}

PriorityQueue<CharacterFrequency> pq = new PriorityQueue<>(

Comparator.comparing(CharacterFrequency::getFrequency).reversed()

);

for(Map.Entry<Character, Integer> entry: map.entrySet()){

pq.add( new CharacterFrequency(entry.getKey(), entry.getValue()));

}

StringBuilder sb = new StringBuilder();

while(!pq.isEmpty()){

CharacterFrequency cf = pq.poll();

for(int i = 0; i< cf.frequency; i++){

sb.append(cf.character);

}

}

return sb.toString();

}

}