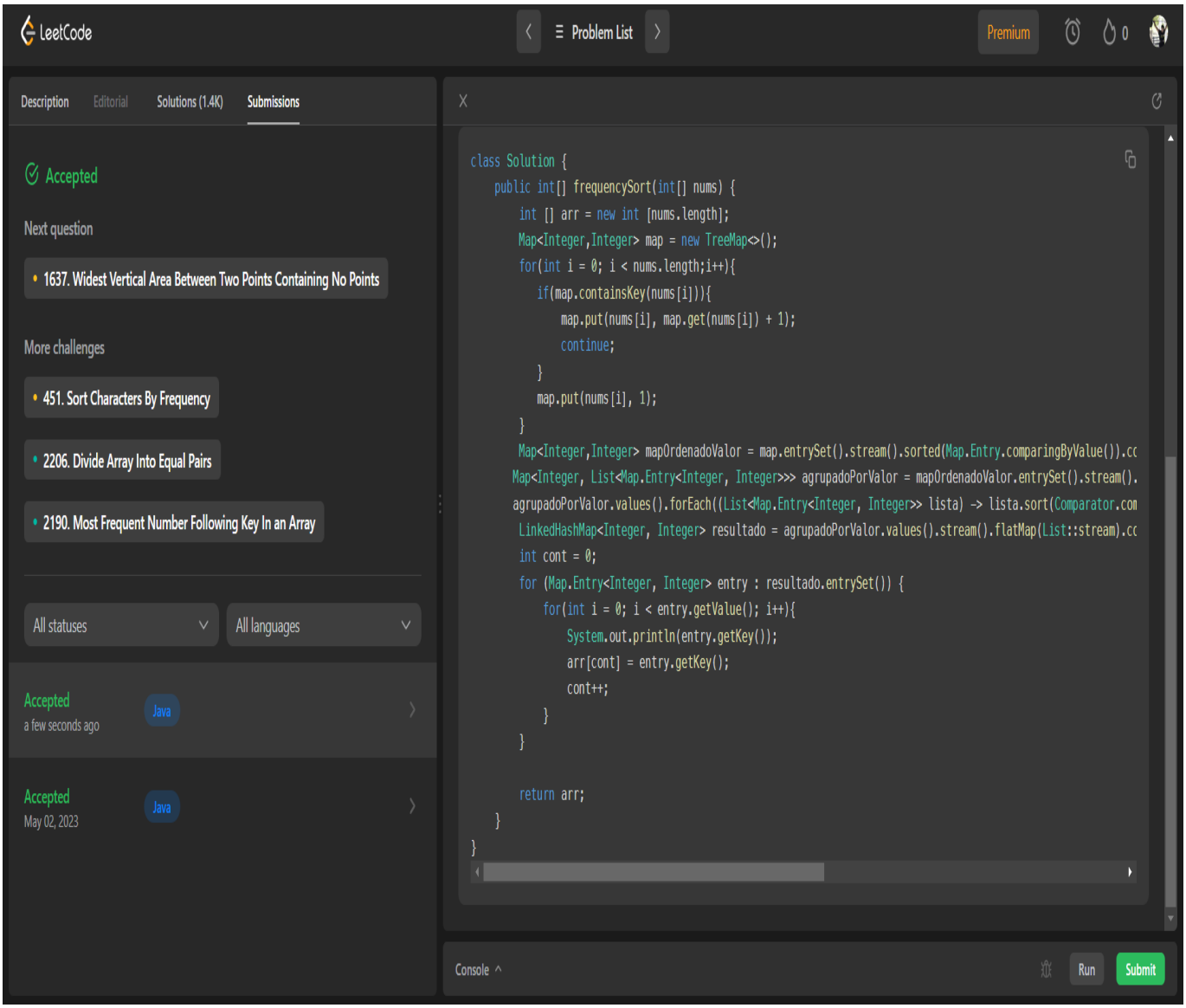
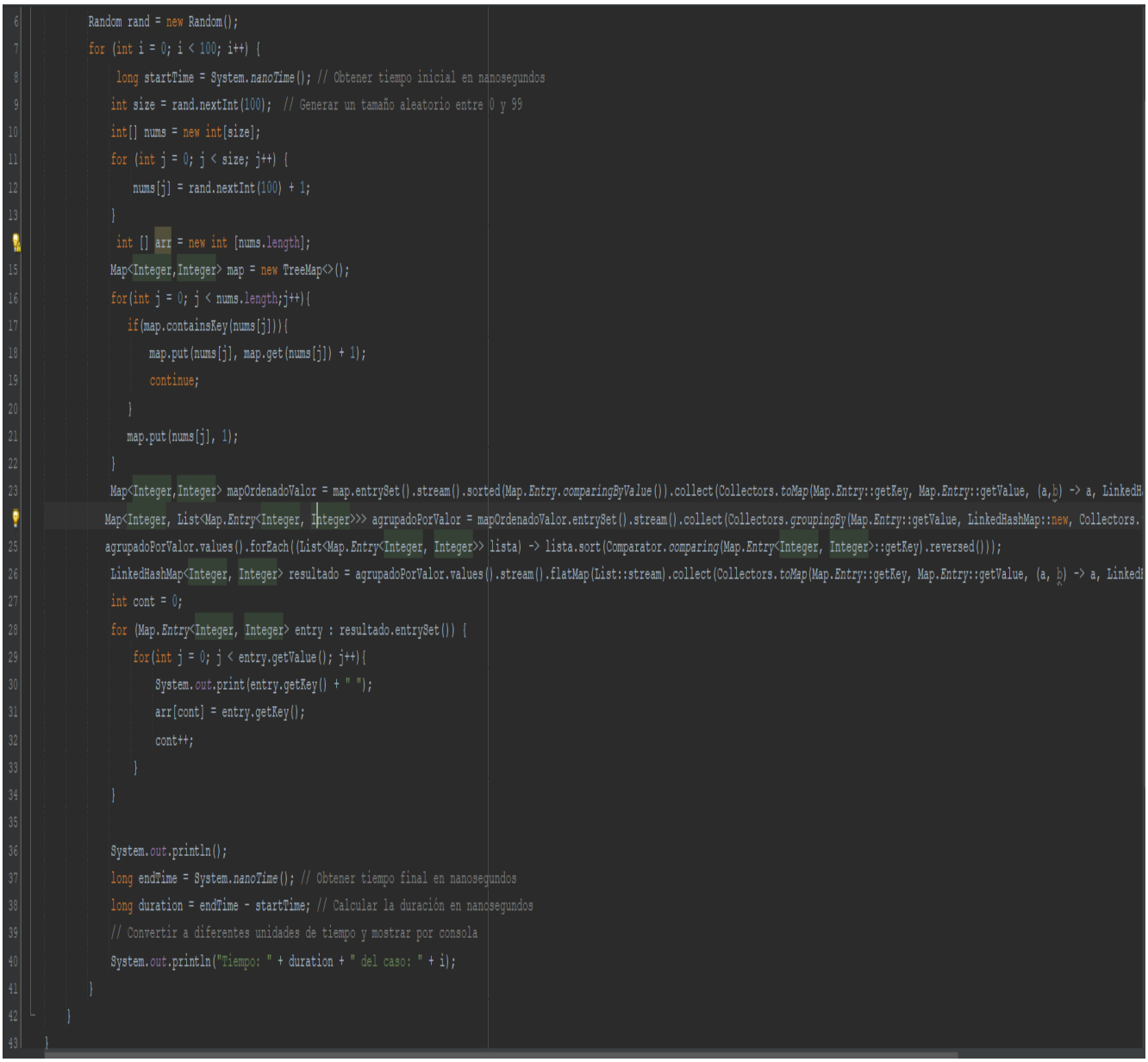
KEVIN ANDRES SARMIENTO SIERRA 1152061



KEVIN ANDRES SARMIENTO SIERRA 1152061



```
        Random rand = new Random();
        for (int i = 0; i < 100; i++) {
            long startTime = System.nanoTime(); // Obtener tiempo inicial en nanosegundos
            int size = rand.nextInt(100);  // Generar un tamaño aleatorio entre 0 y 99
            int[] nums = new int[size];
            for (int j = 0; j < size; j++) {
                nums[j] = rand.nextInt(100) + 1;
            }
             int [] arr = new int [nums.length];
            Map<Integer,Integer> map = new TreeMap<>();
            for(int j = 0; j < nums.length;j++){
               if(map.containsKey(nums[j])){
                   map.put(nums[j], map.get(nums[j]) + 1);
                   continue;
               }
               map.put(nums[j], 1);
            }
            Map<Integer,Integer> mapOrdenadoValor = map.entrySet().stream().sorted(Map.Entry.comparingByValue()).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue, (a,b) -> a, LinkedH
           Map<Integer, List<Map.Entry<Integer, Integer>>> agrupadoPorValor = mapOrdenadoValor.entrySet().stream().collect(Collectors.groupingBy(Map.Entry::getValue, LinkedHashMap::new, Collectors.
           agrupadoPorValor.values().forEach((List<Map.Entry<Integer, Integer>> lista) -> lista.sort(Comparator.comparing(Map.Entry<Integer, Integer>::getKey).reversed()));
            LinkedHashMap<Integer, Integer> resultado = agrupadoPorValor.values().stream().flatMap(List::stream).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue, (a, b) -> a, LinkedH
            int cont = 0;
            for (Map.Entry<Integer, Integer> entry : resultado.entrySet()) {
                for(int j = 0; j < entry.getValue(); j++){
                    System.out.print(entry.getKey() + " ");
                    arr[cont] = entry.getKey();
                    cont++;
                }
            }

            System.out.println();
            long endTime = System.nanoTime(); // Obtener tiempo final en nanosegundos
            long duration = endTime - startTime; // Calcular la duración en nanosegundos
            // Convertir a diferentes unidades de tiempo y mostrar por consola
            System.out.println("Tiempo: " + duration + " del caso: " + i);
        }
    }
}
```

KEVIN ANDRES SARMIENTO SIERRA 1152061

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ AnalisisAlgoritmos ---
100 99 93 92 79 78 77 74 73 66 65 63 62 59 55 54 53 52 51 50 49 48 47 44 43 42 38 36 29 22 21 20 18 17 7 2 96 96 89 89 88 88 82 82 80 80 69 69 57 57 37 37 34 34 26 26 72 72 72 45 45 45 30 30 30 8 8 8
Tiempo: 92424400 del caso: 0
99 89 75 64 60 56 54 51 41 39 37 33 26 23 19 18 13 11 95 95 87 87 69 69 55 55 46 46 42 42 32 32 15 15 14 14
Tiempo: 1751100 del caso: 1
96 94 85 84 83 81 77 75 72 71 70 67 65 63 59 58 57 54 43 42 40 38 37 35 34 31 19 17 16 13 10 8 7 5 3 95 95 90 90 46 46 89 89 89
Tiempo: 2631100 del caso: 2
99 92 69 66 65 51 36 30 12 11 7 5
Tiempo: 910300 del caso: 3
94 92 80 73 72 69 67 66 60 58 57 53 50 49 48 43 42 38 34 31 29 28 27 23 21 16 15 13 8 7 5 89 89 86 86 75 75 64 64 30 30 26 26 20 20 17 17 91 91 91 88 88 88 78 78 78 46 46 46 41 41 41 19 19 19 36 36 36 36
Tiempo: 3259500 del caso: 4
95 92 91 83 82 80 70 69 68 67 63 62 54 47 46 43 42 40 39 38 33 31 30 29 27 23 22 21 20 15 10 8 7 6 5 99 99 96 96 61 61 58 58 48 48 36 36 28 28 16 16 13 13 3 3 97 97 97 75 75 75 73 73 73 14 14 14 2 2 2 64 64 64 64
Tiempo: 7858300 del caso: 5
100 98 94 93 87 84 82 77 73 71 66 61 58 47 39 38 37 36 32 28 26 24 23 16 14 5 3 2 92 92 91 91 89 89 80 80 78 78 50 50 46 46 31 31 27 27 12 12 10 10 68 68 68 41 41 41 22 22 22 19 19 19 81 81 81 81
Tiempo: 3115000 del caso: 6
99 98 90 78 75 74 71 69 61 60 56 55 52 51 44 40 39 38 35 34 32 29 24 21 19 17 16 12 11 6 4 2 97 97 89 89 85 85 76 76 62 62 50 50 36 36 31 31 20 20 87 87 87 82 82 82 77 77 77 72 72 72 66 66 66 58 58 58 45 45 45 33 33 33 18 18
Tiempo: 2287900 del caso: 7
100 95 93 92 91 87 86 85 81 76 68 64 60 59 57 55 54 53 47 46 44 42 41 39 35 33 28 22 21 17 15 12 11 7 6 4 2 80 80 75 75 72 72 70 70 61 61 51 51 49 49 48 48 38 38 26 26 18 18 84 84 84 79 79 79 67 67 67 56 56 56 36 36 36 20 20
Tiempo: 3194100 del caso: 8
93 85 83 72 25 12 9
Tiempo: 545700 del caso: 9
97 90 62 59 50 3
Tiempo: 354200 del caso: 10
100 90 87 83 78 77 73 72 71 69 66 64 62 61 58 47 44 40 37 35 32 31 28 20 19 16 13 12 11 4 1 99 99 97 97 94 94 93 93 81 81 74 74 60 60 57 57 56 56 55 55 52 52 41 41 29 29 17 17 10 10 8 8 2 2 89 89 89 14 14 14 25 25 25 25 25
Tiempo: 2097500 del caso: 11
100 97 95 89 86 84 82 79 78 75 72 69 67 62 60 57 55 52 48 47 45 41 35 33 27 24 23 22 20 18 17 10 8 3 99 99 94 94 80 80 66 66 59 59 46 46 44 44 37 37 30 30 26 26 21 21 19 19 12 12 7 7 93 93 93 4 4 4 4 61 61 61 61
Tiempo: 2942600 del caso: 12
93 86 74 73 61 37 34 15 13 1
```

100 97 95 89 86 84 82 79 78 75 72 69 67 62 60 57 55 52 48 47 45 41 35 33 27 24 23 22 20 18 17 10 8 3 99 99 94 94 80 80 66 66 59 59 46 46 44 44 37 37 30 30 26 26 21 21 19 19 12 12 7 7 93 93 93 4 4 4 4 61 61 61 61
Tiempo: 2942600 del caso: 12
93 86 74 73 61 37 34 15 13 1

```
Tiempo: 657500 del caso: 13
98 90 89 85 82 77 70 64 58 52 49 44 39 38 36 34 30 28 21 19 14 4 100 100 94 94 88 88 81 81 46 46 10 10 10
Tiempo: 1730100 del caso: 14
100 99 97 93 91 87 86 84 83 79 78 76 75 69 67 63 61 60 58 55 53 52 49 45 44 43 37 28 27 24 23 20 16 15 10 7 89 89 85 85 72 72 71 71 68 68 66 66 57 57 56 56 48 48 46 46 38 38 29 29 22 22 19 19 9 9 6 6 5 5 1 1 54 54 54 42 42 42
Tiempo: 2509300 del caso: 15
94 87 80 78 48 25 23 19 13 10 8 75 75 30 30
Tiempo: 566600 del caso: 16
100 99 98 93 90 87 86 75 73 71 70 69 66 64 63 57 55 54 51 50 48 46 42 39 38 37 35 34 25 24 23 20 18 14 13 5 95 95 94 94 89 89 85 85 79 79 77 77 67 67 65 65 47 47 26 26 10 10 7 7 16 16 16 2 2 2
Tiempo: 7405600 del caso: 17
98 94 92 91 88 87 80 77 76 75 72 66 64 61 60 59 56 55 44 42 41 33 26 22 21 19 17 16 13 12 7 6 5 4 96 96 79 79 69 69 63 63 45 45 36 36 71 71 71 15 15 15
Tiempo: 1245800 del caso: 18
91 86 84 83 79 78 76 71 70 67 64 59 54 49 45 44 39 35 33 32 27 26 21 20 10 8 1 99 99 95 95 85 85 81 81 22 22 13 13 9 9 6 6 14 14 14 58 58 58 58
Tiempo: 1012700 del caso: 19
99 98 97 89 87 85 82 80 72 71 70 69 67 65 64 63 62 61 60 59 57 54 47 43 41 39 36 35 34 31 30 27 26 22 18 14 8 7 2 1 92 92 77 77 76 76 66 66 50 50 44 44 40 40 32 32 29 29 16 16 11 11 5 5 4 4 88 88 88 52 52 52 51 51 51 33 33 33
Tiempo: 1198400 del caso: 20
97 96 91 85 78 74 71 59 46 42 33 20 19 15 14 12 6 1 61 61
Tiempo: 630300 del caso: 21
99 97 96 93 88 86 84 77 70 66 65 64 62 61 57 54 52 47 46 41 40 38 35 32 30 28 26 22 20 19 18 15 14 11 7 6 2 94 94 89 89 81 81 48 48 45 45 42 42 36 36 31 31 24 24 10 10 60 60 60 3 3 3
Tiempo: 1508600 del caso: 22
99 98 97 95 93 92 91 90 89 88 86 84 83 82 81 79 77 76 74 71 68 67 64 54 53 52 48 47 45 43 42 41 40 37 33 30 28 27 25 22 20 19 13 12 8 5 3 2 100 100 87 87 78 78 72 72 60 60 57 57 44 44 38 38 36 36 26 26 18 18 17 17 11 11 7 7 6
Tiempo: 1562600 del caso: 23
100 96 90 89 88 87 86 83 77 74 73 69 68 65 59 57 54 49 46 45 43 41 40 38 37 33 29 26 24 23 17 15 14 12 10 7 3 81 81 80 80 72 72 64 64 47 47 30 30 11 11 4 4 93 93 93 75 75 75 39 39 39
Tiempo: 1101200 del caso: 24
99 94 91 88 85 84 81 80 75 62 57 53 45 43 41 40 39 37 35 29 26 24 23 18 16 15 11 4 2 1 98 98 93 93 92 92 66 66 58 58 50 50 44 44 25 25 20 20 14 14 13 13 5 5 3 3 79 79 79 52 52 52 51 51 51 31 31 31 30 30 30 65 65 65 6 6 6 6
Tiempo: 1866100 del caso: 25
97 95 92 91 85 84 75 73 72 69 67 55 53 52 50 49 48 47 46 44 43 42 39 37 33 32 31 28 23 21 20 16 15 14 13 11 9 6 4 2 1 90 90 87 87 74 74 58 58 54 54 34 34 29 29 25 25 22 22 8 8 8 7 7 3 3 96 96 96 89 89 89 81 81 81 80 80 80 65 65
Tiempo: 1530000 del caso: 26
100 96 95 93 86 84 83 81 75 74 72 71 70 69 60 56 53 46 40 39 38 33 24 20 17 11 90 90 51 51 42 42 41 41 32 32 27 27 4 4 1 1 88 88 88 78 78 78 50 50 50 45 45 45 34 34 34 28 28 28 12 12 12 79 79 79 79 8 8 8 8
Tiempo: 921300 del caso: 27
96 77 57 28 21 9
Tiempo: 573100 del caso: 28
99 95 94 93 84 83 81 80 75 68 57 56 55 54 50 49 47 43 34 33 31 21 20 14 13 11 10 7 2 98 98 85 85 37 37 26 26 19 19 12 12 5 5
Tiempo: 966600 del caso: 29
80 78 72 70 68 47 8
Tiempo: 444900 del caso: 30
97 96 93 92 86 85 81 78 77 73 71 69 61 58 57 56 55 52 47 46 45 43 42 37 33 22 21 20 14 13 11 9 8 95 95 91 91 80 80 74 74 68 68 66 66 64 64 54 54 51 51 49 49 44 44 41 41 40 40 39 39 36 36 32 32 30 30 28 28 23 23 12 12 1 1 88 8
Tiempo: 1115300 del caso: 31
99 94 87 86 82 81 75 73 71 69 64 62 58 56 49 44 43 42 40 38 36 34 33 30 29 25 22 20 19 13 11 10 9 6 4 2 97 97 93 93 92 92 89 89 88 88 80 80 78 78 67 67 59 59 57 57 54 54 45 45 32 32 31 31 27 27 16 16 15 15 7 7 3 3 1 1 83 83 8
Tiempo: 1061400 del caso: 32
```

KEVIN ANDRES SARMIENTO SIERRA 1152061

El costo algorítmico de este código depende del tamaño de la entrada, que en este caso es la longitud del arreglo "nums".

La primera línea crea un arreglo "arr" del mismo tamaño que "nums", lo que tiene un costo constante O(1).

Luego, el código utiliza un "TreeMap" para contar la cantidad de ocurrencias de cada número en el arreglo "nums". El bucle for recorre todo el arreglo "nums", lo que tiene un costo de O(n), donde "n" es la longitud de "nums". Dentro del bucle, se utiliza el método "containsKey" del mapa, que tiene un costo promedio de O(log n) para un "TreeMap". Si el número ya está en el mapa, se actualiza su conteo, lo que también tiene un costo de O(log n). Si el número no está en el mapa, se agrega con un conteo inicial de 1, lo que también tiene un costo de O(log n). Por lo tanto, el costo total del bucle for es O(n log n).

A continuación, el código ordena el mapa por valor utilizando un "stream" de Java 8. El costo de la ordenación depende del algoritmo utilizado, pero en promedio es O(n log n) para la mayoría de los algoritmos de ordenación. Luego, agrupa los elementos del mapa ordenado por valor en una lista de listas, donde cada sublista contiene elementos con el mismo valor. El costo de la agrupación es O(n), ya que se recorre el mapa ordenado una vez.

A continuación, se ordenan las sublistas por clave (en orden descendente) utilizando un "stream" de Java 8. El costo de la ordenación es O(m log m), donde "m" es el número total de elementos en todas las sublistas, ya que cada sublista se ordena por separado.

A continuación, se crea un nuevo mapa que contiene los mismos elementos que el mapa original, pero ordenados por valor y luego por clave. Esto se hace utilizando otro "stream" de Java 8 y el método "flatMap". El costo de este proceso es O(n), ya que se recorre el mapa ordenado una vez.

Finalmente, se recorre el mapa ordenado y se imprime cada clave tantas veces como indique su valor, y se agrega cada clave al arreglo "arr". El costo de este proceso es O(n), ya que se recorre el mapa ordenado una vez.

El costo algorítmico total de este código es O(n log n), donde "n" es la longitud del arreglo "nums".

Video de youtube: https://youtu.be/Vv2j1SiQ6sM

GitHub: https://github.com/KevinSarmiento07/An-lisis-de-Algoritmos