Wooden Toy Festival CODEFORCES CHALLENGE

Authors

JAMES ALBANESE STEVEN CHASE ROEY ELMELECH **Date** 8 / 9 / 2 3



Presentation Contents ROEY

- Understanding the problem
- Exploring test cases
- First, brute-force attempt
- Diving in
- Code walk-through
- JProfiler results

The Problem ROEY

- In a town, three woodcarvers prep for a toy festival
- They can each specialize in 1 pattern (x)
- Every guest at the festival can ask for a toy with a pattern (y) to be carved
- A toy of pattern y takes |x-y| time to make
- The carvers aim to minimize maximum wait time
- Output the best achievable maximum wait time

• Time limit per test: 3

seconds

Memory limit per test:

256 megabytes

- Input: standard input
- Output: standard output

Test Cases

- Example of a expected input-output
- Edge cases
- Toy patterns x such that $1 \le x \le 10^9$

```
input {
1 7 7 9 9 9
5 4 2 1 30 60
9
  19 37 59 1 4 4 98
73
```

```
output {
    0
    2
    13
    0
    1
    }
```



Our First Attempt

BRUTE-FORCE APPROACH IN PYTHON

Brute Force STEVEN

• Every possible combination for each 3

carvers (i, j, k)

Minimum wait time for each customer

(based on the carver positions)

- Compare the wait times
- Return the overall lowest maximum wait

Runtime on our test cases: Didn't finish on our largest test case!

```
def toy(patterns):
    # 3 or less patterns to make --> 0 wait time
    if len(set(patterns)) <=3:</pre>
        return 0
    # sort patterns
    patterns.sort()
    max_wait = 1000000000000
    # Check every combination of x patterns for carvers 1,2,3
    first_pattern = patterns[0]
    last_pattern = patterns[-1]
    for i in range(first_pattern, last_pattern-1):
        for k in range(i+1, last_pattern):
            for j in range(k+1, last_pattern+1):
                # Wait array, track waiting time for each person
                wait_array = []
                # Update wait array by each carver
                for pattern in patterns:
                    wait_array.append(min([abs(pattern - i), # carver 1
                                            abs(pattern - k), # carver 2
                                            abs(pattern - j)])) # carver 3
                # Check max wait
                cur_max = max(wait_array)
                if cur_max < max_wait:</pre>
                    max_wait = cur_max
    return max_wait
```



Second Attempt

BINARY SEARCH

```
1, 4, 4, 14, 19, 37, 59, 73, 987
   EX:
                            X2 ≤ 2.€
                                                X2 ≤ 2+ X3 ≤ 2+
Search space of t:
                                                    Carver 1 = 1 -> (1+26)
                                                                1,4,4,14,19,37,59,73,987
    LO
                                  \frac{t=24}{\text{Corver }} = 1 \rightarrow (1+2t) = 1 \rightarrow 49 \left[ \frac{1}{1}, \frac{1}{4}, \frac{1}{4}, \frac{1}{9}, \frac{37}{37} \right]
                                  carver 2= 59 + (59+2+) = 59 ->107 [59, 73, 98]
                               t= 12
Carver 1 = 17 (1+2t) = 1725 [14,4,14,19]
                               carrer 2= 37 + (37+2t) = 37-761 [37, 59]
                               Carver 3 = 73 -> (73+26) = 73-797 [73] = no 98
           13 18 24
           Lo mid H .
                                    t= 12
Corver 1 = 1 -> (1+2t) = 1+27 [14,4,14,19]
                                    carver 2= 37 + (37+2t) = 37->65 [37, 59]
                                    Carver 3 = 73 -> (3+26) = 73-799 [73, 98]
```

Binary Search

STEVEN

- The requested patterns array can be split into 3 sections, 1 for each carver to cover
- There exists a minimal maximum wait time t such that the range of each section is <= 2t
- Estimate t directly using binary search $\rightarrow \theta(\log n)$

Binary Search: Final Java Solution

CODE WALK-THROUGH

```
* Function to calculate the minimum number of se
 * @param n The number of toy patterns
 * @param toyPatterns The array of toy patterns
 * @return The minimum number of seconds to compl
*/
private static int toy(int n, int[] toyPatterns)
    Arrays.sort(toyPatterns);
    // Initialize the lower and upper bounds for
    int lo = 0;
    int hi = toyPatterns[toyPatterns.length-1];
    // Perform binary search
    while (lo < hi) {
        // Calculate the mid point
        int mid = (hi + lo) / 2;
        // Calculate the first range
        int x1 = toyPatterns[0] + 2 * mid;
```

JProfiler Results

JAMES

Checking for base case

```
100.0% - 9,129 μs - 1 inv. ToyFestival.main

79.3% - 7,238 μs - 10 inv. ToyFestival.toy

33.1% - 3,019 μs - 10 inv. java.util.stream.IntStream.toArray

18.0% - 1,643 μs - 10 inv. java.util.stream.IntStream.distinct

15.9% - 1,454 μs - 10 inv. java.util.Arrays.stream

8.5% - 776 μs - 10 inv. java.util.Arrays.sort

15.1% - 462 μs - 1 inv. java.io.FileReader.<init>

13.1% - 283 μs - 21 inv. java.io.BufferedReader.readLine

2.8% - 255 μs - 10 inv. java.io.PrintStream.println

1.4% - 131 μs - 84 inv. java.lang.String.split

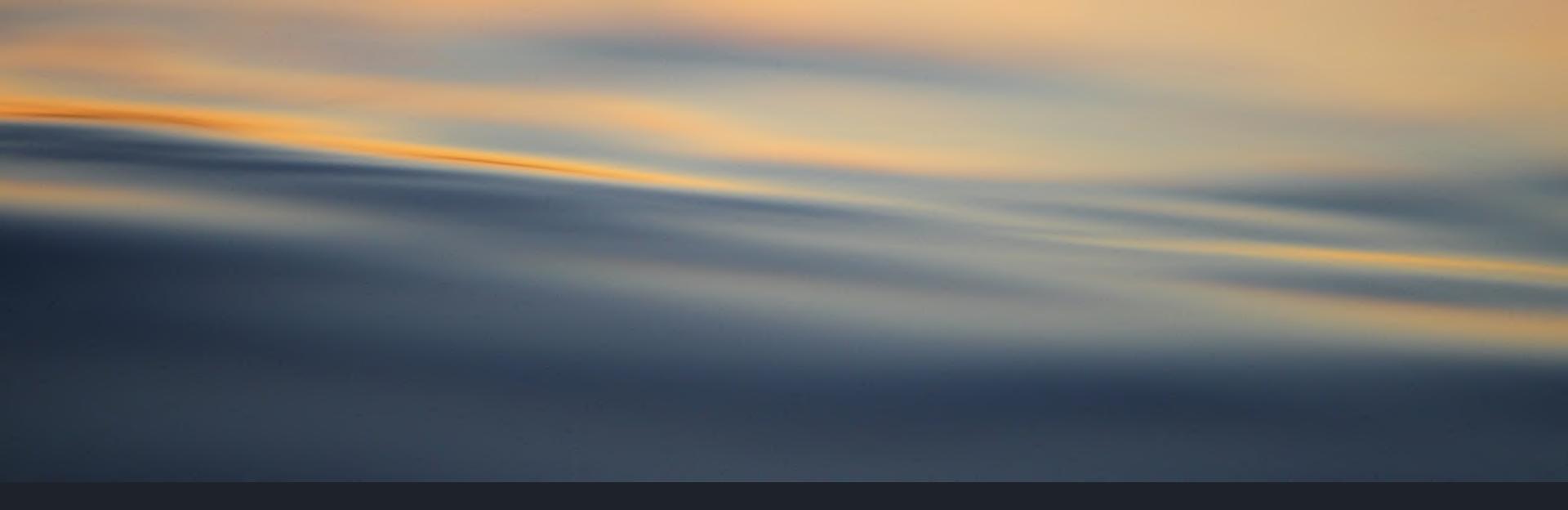
1.4% - 10 μs - 11 inv. java.lang.String.trim

0.1% - 8 μs - 1 inv. java.io.BufferedReader.<init>
```

Without checking for base case

```
100.0% - 3,245 μs - 1 inv. ToyFestival.main
31.5% - 1,021 μs - 10 inv. ToyFestival.toy
22.4% - 727 μs - 10 inv. java.util.Arrays.sort
20.3% - 659 μs - 1 inv. java.io.FileReader.<init>
8.0% - 258 μs - 21 inv. java.io.BufferedReader.readLine
7.6% - 248 μs - 10 inv. java.lang.String.split
5.4% - 174 μs - 10 inv. java.io.PrintStream.println
3.6% - 118 μs - 84 inv. java.lang.Integer.parseInt
0.3% - 11 μs - 1 inv. java.io.BufferedReader.<init>
0.3% - 11 μs - 11 inv. java.lang.String.trim
```





Thank You For Listening JAMES

Any questions for the team?