LINK TO THE PROBLEM

Wooden Toy Festival

CODEFORCES CHALLENGE

Authors

JAMES ALBANESE STEVEN CHASE

ROEY ELMELECH

Date

8/9/23

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

JAMES

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

```
input {
5
6
1 7 7 9 9 9
6
5 4 2 1 30 60
9
14 19 37 59 1 4 4 98
73
1
2
6
3 10 1 17 15 11
}
```

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

Our First Attempt

BRUTE-FORCE APPROACH IN PYTHON

Brute Force STEVEN

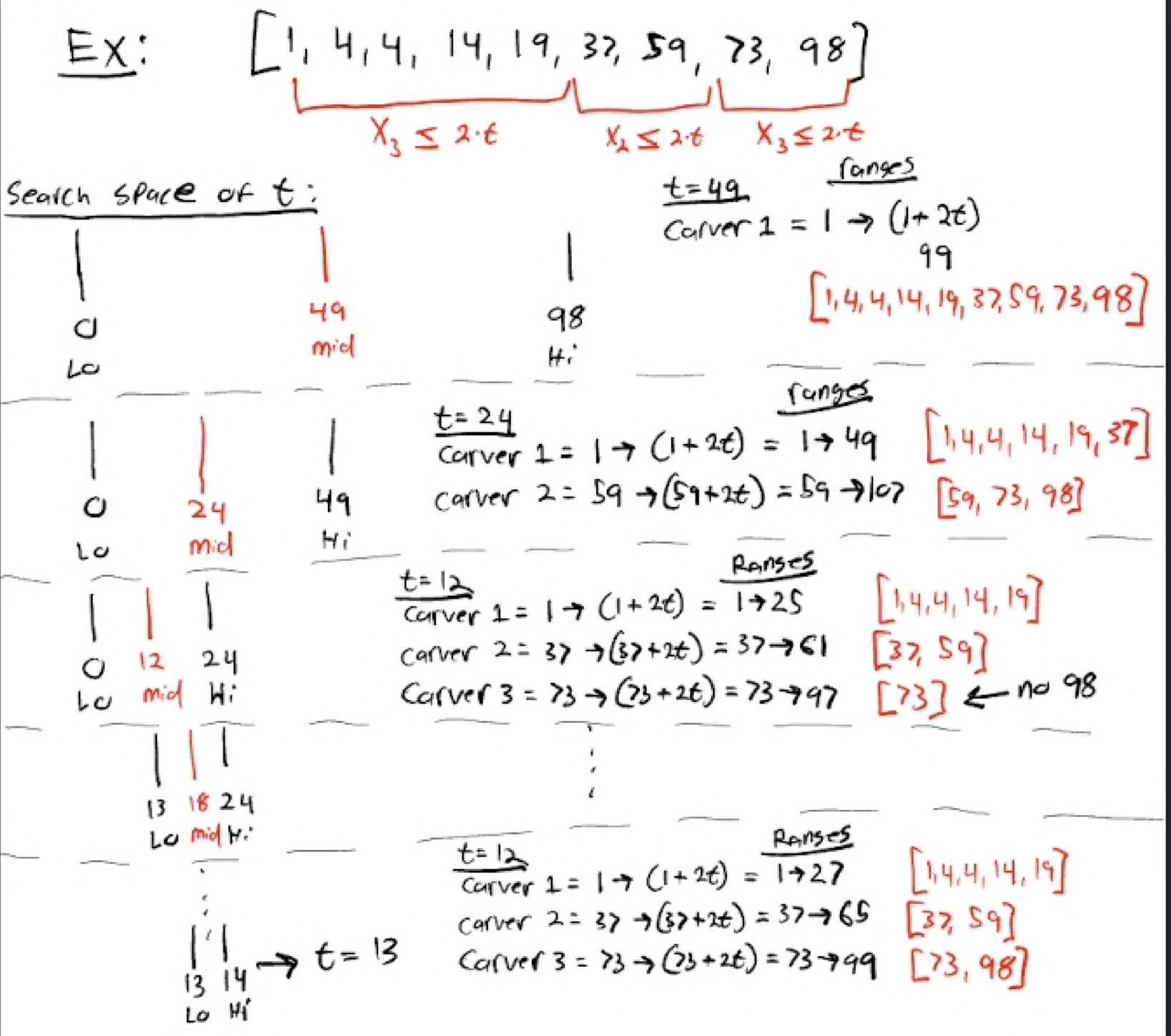
- Every possible combination for each 3 carvers (i, j, k)
- Minimum wait time for each customer (based on the carver positions)
- Compare the wait times
- Return the overall lowest maximum wait

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

```python
def toy(patterns):
    # 3 or less patterns to make --> 0 wait time
    if len(set(patterns)) <=3:
        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:
                    max_wait = cur_max

    return max_wait
```

Second Attempt

BINARY SEARCH



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 —> θ(*log* n)

Binary Search: Final Java Solution

ROEY

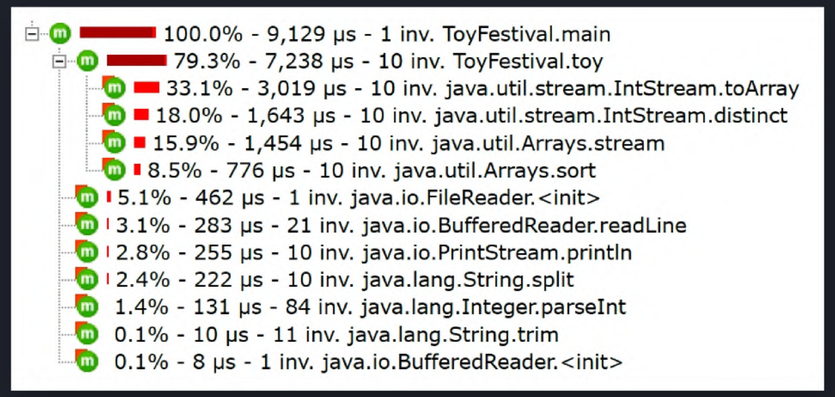
CODE WALK-THROUGH

```java
* 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;
```

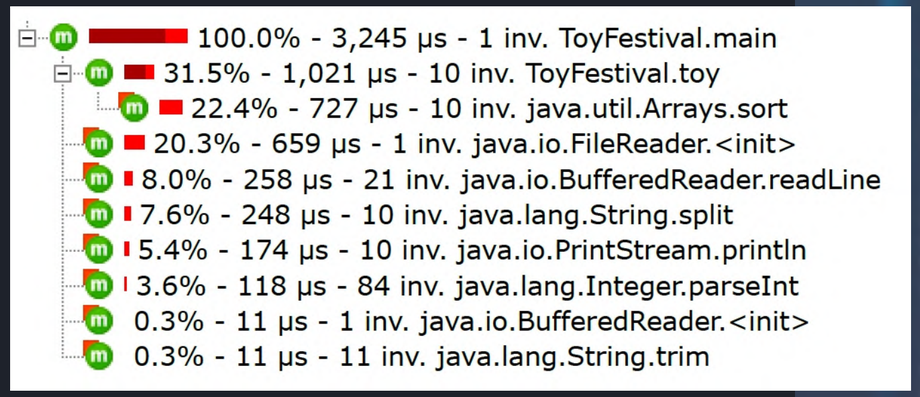
JProfiler Results

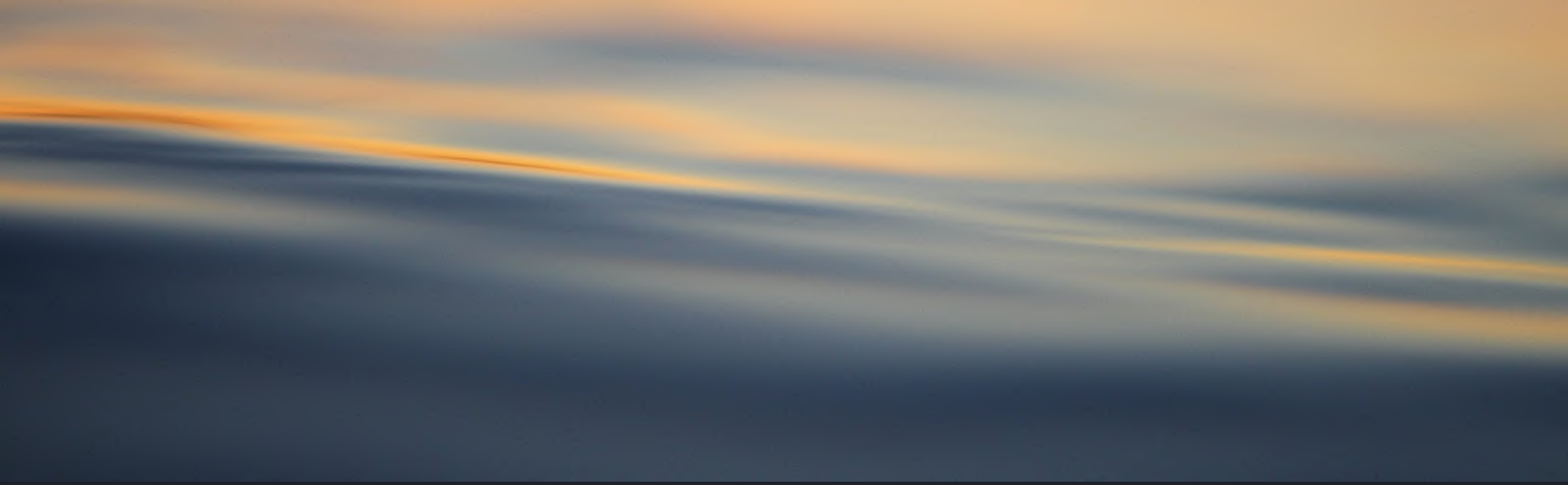
JAMES

- Checking for base case



- Without checking for base case





Thank You For Listening

JAMES

Any questions for the team?