Time Complexity Analysis of Trapezium Algorithm

Roy Ananth, Sambahangphe Mishek, Shea Jackson, Ng Sunnathan, and Desai Param

November 28, 2023

1 Introduction

Analysis of the trapezium algorithm in sequential and parallel forms has been tested with 100,000,000 partitions. The analysis shows that the sequential algorithm has a time complexity $\mathcal{O}(n)$ and the parallel algorithm has a time complexity of $\mathcal{O}(n)$. The parallel algorithm is faster because it is able to split the input n into sub threads and work on them simultaneously. The time complexity of the algorithms is the same.

2 Sequential Trapezium Algorithm

```java
public static double trapezium(double a, double b, int n, FPFunction f) {
  double range = checkParamsGetRange(a, b, n);
  double nFloat = (double) n;
  double sum = 0.0;
  for (int i = 1; i < n; i++) {
    double x = a + range * (double) i / nFloat;
    sum += f.eval(x);
  }
  sum += (f.eval(a) + f.eval(b)) / 2.0;
  return sum * range / nFloat;
}
```

2.1 Hypothesis

The sequential trapezium algorithm has a time complexity of $\mathcal{O}(n)$. The main for loop iterates 'n' times where 'n' is the input size, while the other operations outside of the for loop have a time complexity of $\mathcal{O}(1)$, which is constant and does not depend on 'n'. Hence, the time complexity only depends on the for loop which will equate to $\mathcal{O}(n) + \mathcal{O}(1) = \mathcal{O}(n)$

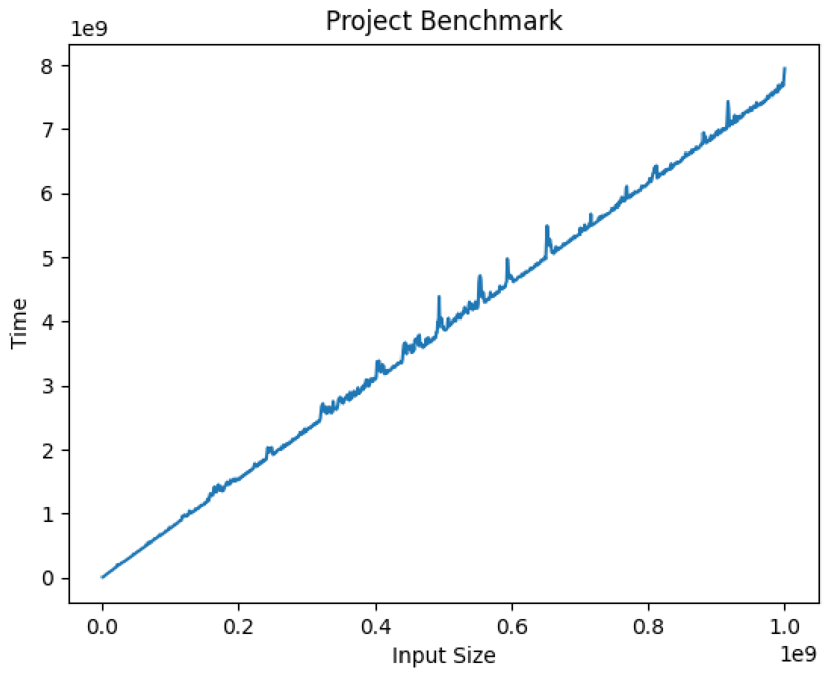
3 Parallel Trapezium Algorithm

```java
public static double trapezium(double a, double b, int n, FPFunction f)
  {
    double range = checkParamsGetRange(a, b, n);

    return (IntStream.range(0, n).parallel().mapToDouble(i-> a + range * (double)

  }
```

3.1 Hypothesis

The parallel trapezium algorithm has a time complexity of $\mathcal{O}(n)$. Streams work the same as for-loops under the hood so the time complexity should be the same as a single for-loop, or $\mathcal{O}(n)$. The difference is that parallel streams can work simultaneously on the inputs by splitting into sub-threads.

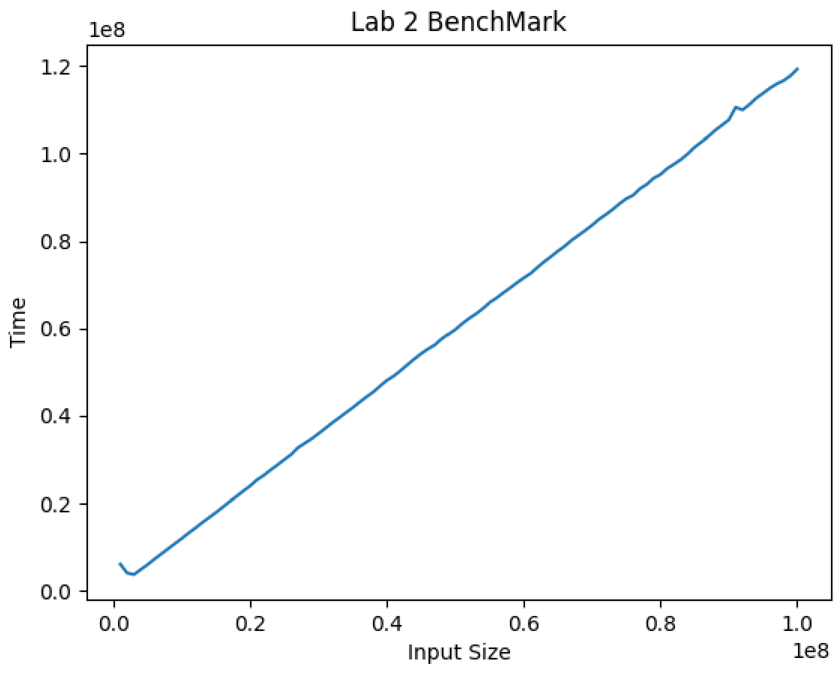
4 Graph of Sequential Trapezium Algorithm



4.1 Analysis

Our hypothesis that the sequential trapezium algorithm has a time complexity of $\mathcal{O}(n)$ was correct. As n or input size grows the time required to complete the calculation increases linearly. $\mathcal{O}(n)$ increases linearly as the input size grows, therefore our hypothesis was correct. The sequential trapezium algorithm through analysis of the graph does indeed have a time complexity of $\mathcal{O}(n)$.

5 Graph of Parallel Trapezium Algorithm



5.1 Analysis

Our hypothesis that the parallel trapezium algorithm has a time complexity of $\mathcal{O}(n)$ was correct. As n or input size grows the time required to complete the calculation increases linearly. $\mathcal{O}(n)$ increases linearly as the input size grows, therefore our hypothesis was correct. The parallel trapezium algorithm through analysis of the graph does indeed have a time complexity of $\mathcal{O}(n)$.

6 Conclusion

Both the parallel and Sequential Trapezium algorithms have a time complexity of $\mathcal{O}(n)$. Although they share the same time complexity, the parallel algorithm is faster because it is able to split the input n into sub threads and work on them simultaneously. When implementing the code, the sequential method is easier to understand but it is slower. The parallel method is faster but it is more difficult to understand.