Determining the Optimal k Value for a Hybrid Sorting Algorithm

Introduction

The purpose of this report is to describe the process of finding the optimal value of k for a hybrid sorting algorithm that combines merge sort and binary insertion sort. The optimal k represents the threshold at which the algorithm switches from merge sort to binary insertion sort, allowing it to take advantage of the strengths of both algorithms and achieve better performance.

> ⚠️ Please note that the data presented in this report was collected on a 16-inch, 2021 MacBook Pro with an Apple M1 Pro processor and 16 GB of Memory, running macOS Ventura 13.2.1 and virtualising Ubuntu 22.10. The performance results may vary on different systems and configurations. The algorithm runs single-threaded.

Methodology

1. A sorting function was implemented that accepts an array of data, its size, and a k value as parameters. This function uses merge sort for sorting larger subarrays and switches to binary insertion sort for subarrays of size k or smaller.
2. To find the optimal k value, a binary search was performed in the range of 1 to 100. For each candidate k value, a copy of the original dataset was made and the hybrid sorting function was used to sort the data.
3. The sorting time, comparison count, and data move count were measured for each candidate k value. The sorting time was recorded using the system clock, while the comparison and data move counts were captured by modifying the sorting algorithm to track these operations.
4. For each iteration, the sorting time, comparison count, and data move count were compared with the best values observed so far. The best k values for each criterion

were updated accordingly, and the binary search continued until the search range was exhausted.

5. Finally, an overall best k value was determined based on all criteria, considering a weighted combination of sorting time, comparison count, and data move count.

Score Calculation

The score of a sorting algorithm is calculated based on two main factors: comparison count and data move count. These factors are crucial in determining the algorithm's overall efficiency, with lower values indicating better performance. The score combines these factors to provide a single number that represents the algorithm's performance.

1. Comparison count: This is the total number of comparisons made between elements during the sorting process. Comparisons determine the relative order of elements in the array. A more efficient algorithm will minimize the number of comparisons required to sort the array.
2. Data move count: This is the total number of times an element in the array is moved, copied, or swapped during the sorting process. A more efficient algorithm will minimize the number of data moves required to sort the array.

The score is calculated using the following formula:

```
Score = Comparison Count + Data Move Count
```

This formula gives equal weight to both factors, and the resulting score provides an overall measure of the algorithm's efficiency. A lower score indicates better performance, as fewer comparisons and data moves were required to sort the array.

Results

After performing multiple iterations and examining various candidate k values, we found the optimal k values for each criterion. The weights used were:

- Sorting time: 1.0
- Data move count: 1.0
- Comparison count: 1.0

1. Time:
   Field1: k = **100** with a sorting time of **3.373061** seconds
   Field2: k = **96** with a sorting time of **1.958449** seconds
   Field3: k = **96** with a sorting time of **1.933404** seconds
2. Comparison count:
   Field1: k = **94** with a count of **342578924** comparisons
   Field2: k = **94** with a count of **360868637** comparisons
   Field3: k = **94** with a count of **360888351** comparisons
3. Data move count:
   Field1: k = **94** with a count of **693190778** data moves
   Field2: k = **83** with a count of **695199944** data moves
   Field3: k = **85** with a count of **694877332** data moves

The overall best k values considering all criteria and for each field were found to be:

1. Field1: k = **94** with a combined score of **1035769705.002398**
2. Field2: k = **94** with a combined score of **1056290546.925573**
3. Field3: k = **88** with a combined score of **1056185550.937305**

Conclusion

This report detailed the methodology used to find the optimal k value for a hybrid sorting algorithm that combines merge sort and binary insertion sort. The optimal k value allows the algorithm to efficiently balance the performance benefits of both sorting methods, resulting in improved overall sorting performance.