Stock Sorting Using Merge Sort

INTRODUCTION

- The project implements a C program for sorting stock data using the merge sort algorithm.
- Stock data includes attributes such as symbol, price-to-earnings ratio, dividend yield, and growth potential.
- Merge sort, a divide and conquer algorithm, is chosen for its efficiency in sorting large datasets.

Abstract:

- Defines a Stock structure to store details of each stock, including symbol, price-to-earnings ratio, dividend yield, and growth potential.
- Implements mergeSort function to sort an array of Stock structures based on user-specified sorting key.
- Utilizes merge function to merge two subarrays while maintaining sorting order according to the chosen key.
- Allows user input for multiple stocks' details and selection of sorting key (p for price-to-earnings ratio, d for dividend yield, g for growth potential).
- Displays sorted stock data based on the selected sorting key, providing insights into the stocks' attributes.

What is Merge Sort

Merge sort is a popular sorting algorithm that follows the divide-and-conquer strategy. It works by dividing the array into two halves, recursively sorting each half, and then merging the sorted halves to produce a single sorted array. Here's an overview of how merge sort works:

1. Divide: The original array is divided into two halves, roughly equal in size. This step is repeated recursively until each subarray contains only one element.
2. Conquer: Once the subarrays reach a size of one (i.e., they cannot be divided further), they are considered sorted.
3. Merge: The sorted subarrays are then merged back together into a single sorted array. This merging process involves comparing elements from the two subarrays and placing them in the correct order.
4. Repeat: Steps 1-3 are repeated recursively for each pair of subarrays until the entire array is sorted.

Alogrithm

```
1.MERGE_SORT(arr, beg, end)
2.if beg < end
3. set mid = (beg + end)/2
4. MERGE_SORT(arr, beg, mid)
5. MERGE_SORT(arr, mid + 1, end)
6. MERGE (arr, beg, mid, end)
7.end if
END MERGE_SORT
merge(a[], beg, mid, end)
   // Calculate the sizes of the two subarrays
   n1 = mid - beg + 1
   n2 = end - mid

   // Create temporary arrays to hold the data of the two subarrays
   LeftArray[n1], RightArray[n2]

   // Copy data from the original array into temporary arrays
   for i = 0 to n1:
      LeftArray[i] = a[beg + i]
   for j = 0 to n2:
      RightArray[j] = a[mid + 1 + j]

   // Merge the two subarrays back into the original array
   i = 0, j = 0, k = beg
   while i < n1 AND j < n2:
      // Compare elements from the two subarrays and copy the smaller one into the original array
      if LeftArray[i] <= RightArray[j]:
         a[k] = LeftArray[i]
         i++
      else:
         a[k] = RightArray[j]
         j++
      k++

   // Copy any remaining elements from LeftArray
   while i < n1:
      a[k] = LeftArray[i]
      i++
      k++

   // Copy any remaining elements from RightArray
   while j < n2:
      a[k] = RightArray[j]
      j++
      k++
```

Advantages

Efficiency:
Merge sort has a time complexity of O(n log n) in the worst-case scenario, making it highly efficient for sorting large datasets of stocks. In stock trading environments where thousands or even millions of transactions may occur, having an efficient sorting algorithm is crucial for timely data analysis and decision-making.

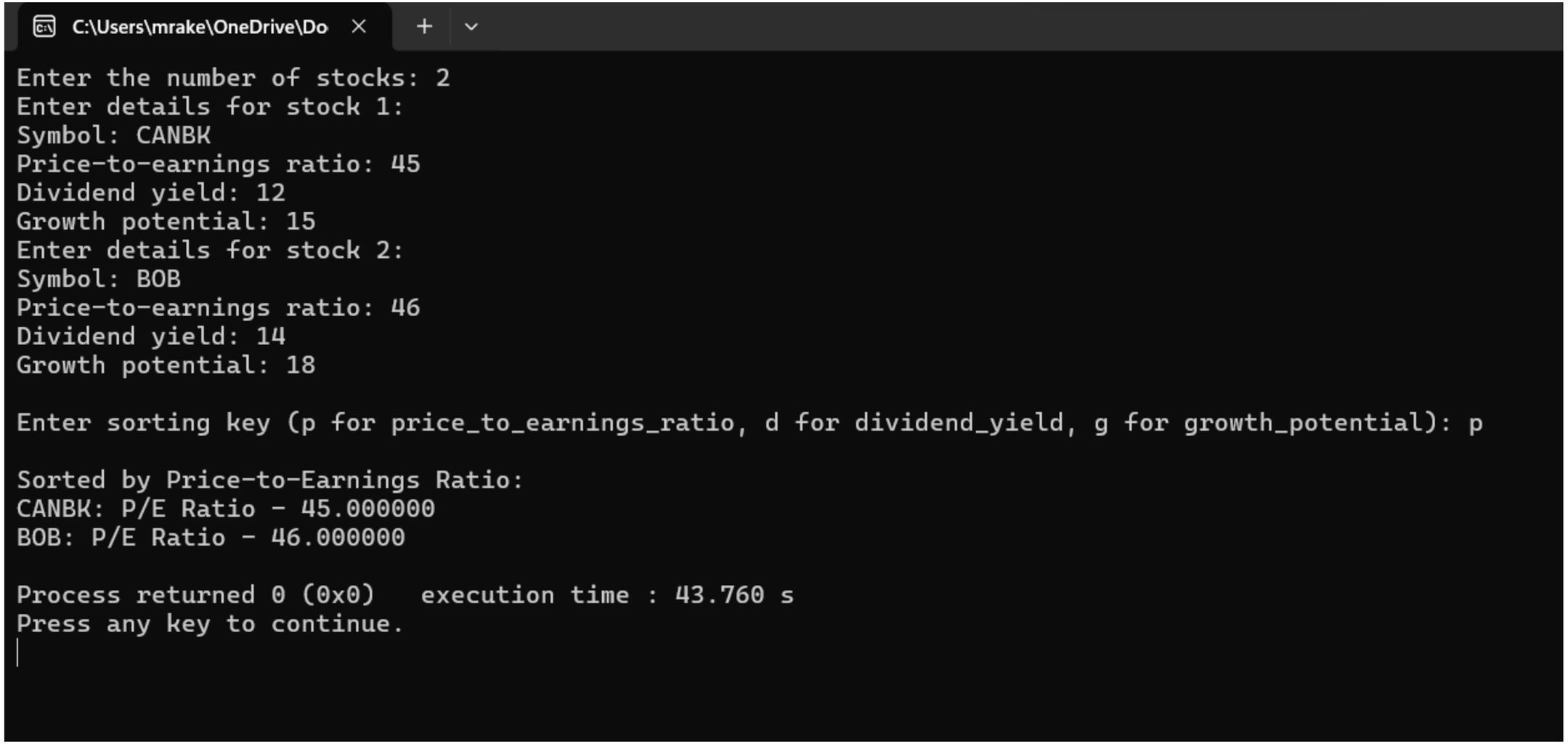
Stability:
Merge sort is a stable sorting algorithm, meaning it preserves the relative order of equal elements. In the context of sorting stock data, stability ensures that if two stocks have the same sorting key value (e.g., two stocks with the same price-to-earnings ratio), their original order will be maintained after sorting. This stability can be important for maintaining the integrity of historical trading data and analysis results.

Time Complexity Analysis

Merge sort has a time complexity of O(n log n) in all cases, where n is the number of elements in the array being sorted. This time complexity analysis can be understood by considering the operations performed in each step of the merge sort algorithm:

1. **Divide Step**:
   - In the divide step, the array is recursively divided into halves until each subarray contains only one element. This process requires O(log n) divisions, as each division reduces the array size by half.
2. **Merge Step**:
   - In the merge step, the sorted subarrays are merged together to form a single sorted array. This merging process involves comparing and rearranging elements from two subarrays to construct the final sorted array.
   - The merge operation takes O(n) time, where n is the total number of elements being merged.
   - Since the merge operation is performed at each level of the recursion tree (O(log n) levels), the total time spent on merging is O(n log n).
3. **Overall Time Complexity**:
   - Combining the time complexities of the divide and merge steps, we get O(n log n) as the overall time complexity of merge sort.

Output


C:\Users\mrake\OneDrive\Do
Enter the number of stocks: 2
Enter details for stock 1:
Symbol: CANBK
Price-to-earnings ratio: 45
Dividend yield: 12
Growth potential: 15
Enter details for stock 2:
Symbol: BOB
Price-to-earnings ratio: 46
Dividend yield: 14
Growth potential: 18
Enter sorting key (p for price_to_earnings_ratio, d for dividend_yield, g for growth_potential): p
Sorted by Price-to-Earnings Ratio:
CANBK: P/E Ratio - 45.000000
BOB: P/E Ratio - 46.000000
Process returned 0 (0x0) execution time : 43.760 s
Press any key to continue.