**Sorting Assignment**

**Knowledge (/12)**

1. **In groups of three, you are responsible for implementing: (one each)**
   1. **Insertion Sort**
   2. **Bubble Sort**
   3. **Selection Sort**

**Thinking(/6)**

1. **If you look at the chart in the lesson, it’s clear that merge sort can be faster than quicksort. Given this, why is it that most organizations/programming languages use quick sort?? To answer this, you’ll likely have to do a bit of research.**

One reason is because of memory. Merge sort requires extra storage which is considered expensive. This is because the algorithm works with merging sub-arrays. Therefore the use of temporary arrays takes up more space. On the other hand, quick sort does not require any extra storage. This is because the quick sort algorithm swaps elements left and right based on its comparison to a pivot. Therefore, it does not require there to be another array. This is referred to as in-place sorting (no extra storage is needed for sorting). Another factor is also a being cache friendly algorithm. For a CPU to get information from the main memory, it requires a lot of time. Therefore cache is used which lies between the CPU and the main memory. The CPU will be able to get information from the cache which is cheaper and faster. However the size of cache memory is smaller than main memory. Therefore locality of reference decides what information in the main memory should be in the cache. The two types of locality of reference are spatial (data around the current data fetched may be needed soon) and temporal (data being fetched may be needed again soon). Quick sort has good locality of reference. Quick sort's algorithm sorts right in the same array it is working in. You benefit from cache when you make multiple accesses to the same place in memory. When using quick sort, the first access is taken from memory. After that, all other accesses can take information from cache. Since quick sort can make good use of cache, it becomes faster than merge sort. Another factor has to do with number of comparisons. In terms of merge sort, it requires comparisons in its best, average, and worst case. In terms of quick sort, it requires comparisons for its best and average case. For its worst case, it requires comparisons. This shows that merge sort can be faster than quick sort. However the worst case for quick sort can be avoided. This is done by using randomized quick sort. This is a type of quick sort where the pivot element is chosen randomly. The worst case is possible but unlikely. There is a higher portability to choose a right pivot element. Therefore choosing the right element point allows you to be closer to the average case which is as efficient as merge sort. In practice, randomized quick sort is effective. Also note that when using quick sort it is possible to make design choices that will avoid using quadratic time.

**Application (10 / 10) and Communication (8)**

1. **The file *stockdataunsorted.txt*, is a comma-delimited file containing the stock price and other financial information over a period of many decades. The format of the file and an example follows:**

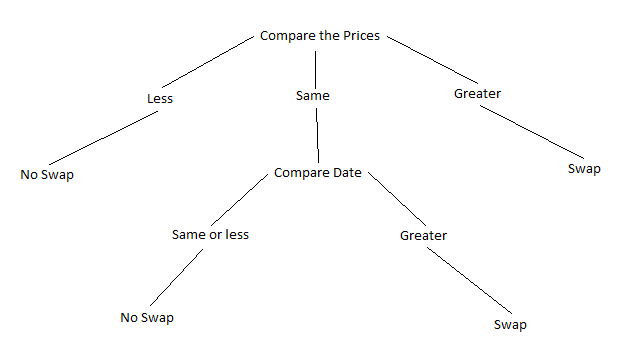
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Price** | **Date (yyyy.mm)** | **Dividend** | **Earnings** | **CPI** | **Interest Rate** | **P/E Ratio** |
| **10.1** | **2009.01** | **.67** | **1.05** | **14.1** | **1.96** | **13.9** |

* 1. **Read the file into an array of your own Class (call it StockInfo). Ensure that this class implements the Comparable interface**
  2. **Sort the array by date, using your helper class**
  3. **Save the sorted information into the file *stockdatasorted.txt***
  4. **If you were asked to sort using two pieces of information (say price first (which may have duplicates), and then by date) how would you change the sorting algorithm to do this? [You don’t have to do it, just explain it]**

The algorithm I used is bubble sort I used for the assignment. The response/diagram below is explained in terms of bubble sort. In order to account for sorting based on multiple criteria, a few changes would need to be made to the algorithm and the StockInfo class.

The algorithm would need to be changed as follows:

You would first need to create an if statement that would compare whether the two prices are the same. If the current price is less than the next price, don't swap. If the current price is greater than the next price, swap. If the prices are the same (duplicate), you would then need to compare them based on their date. If the current date is the same or less then the next date, don't swap. If the current date is greater than the next date, swap.



The compareTo method would need to be changed as follows:

Note that the bubble sort algorithm takes in a Comparable array as a parameter. Therefore when comparing, the method compareTo is called. In the program's current state, it is only comparing the stock's date. Now that you have multiple criteria, you need to be able to compare both prices and date. To account for both in the one compareTo method you can use if statements. The if statement checks if the prices are equal. If the prices are equal, the method will return the result of comparing the dates (return this.date.compareTo(compare.date);). Else the method will return the result of comparing the prices (return this.price.compareTo(compare.price);).

NOTE: Data conversion is a huge thing in the business world, and unfortunately it can be pretty messy. This question in particular, you will find some ‘snags’ or unexpected input. Part of the challenge is determining how you will handle these snags - by hand, naive programming, or gracefully. I just wanted to give you a heads up. ;-D