CS2020: Data Structures and Algorithms (Accelerated)

Problems 4–5

Due: January 26th, before discussion groups

Overview. You have two tasks this week. The first involves improving/modifying the VectorTextFile class in various ways. The goal here is to learn how to work with and improve larger pieces of code. The second problem will give you a chance to develop an efficient algorithm to solve a problem using the techniques we have talked about in class.

Collaboration Policy. As always, you are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 4. (MergeSort, Faster)

Asymptotically, when sorting n elements, MergeSort runs in time $O(n \log n)$, while InsertionSort runs in $O(n^2)$. Thus, for sorting large arrays, MergeSort is significantly faster than InsertionSort.

However, for very small arrays, InsertionSort is actually faster than MergeSort. This is due to the cost of recursion, the cost of allocating space, etc. And since MergeSort is a *divide-and-conquer* algorithm, every large instance of MergeSort eventually is divided into many small instances of MergeSort. Thus it is possible to speed-up MergeSort by using InsertionSort when there are only a small number of elements to sort.

Problem 4.a. VectorTextFile.java and VectortextFile2.java both contain a version of InsertionSort. The InsertionSort routine sorts the entire array `m_WordList`. Modify the InsertionSort routine so that it only sorts a specified portion of the array. That is, the new InsertionSort should take two parameters: int Begin and int End, and it should sort all the words in the array `m_WordList[Begin..End]`, without modifying any entry of the array that is not in the specified range.

(*Remember:* a good routine will check whether the input is valid, i.e., whether `Begin` is ≥ 0 and whether `End<m_FileWordCount`.)

For this part, submit only the modified version of InsertionSort, along with any other changes to VectorTextFile.java. (Do not submit the entire class file.)

Problem 4.b. In the current version of MergeSort (see VectorTextFile3.java), the base case of the recursion occurs when there is only one element left to sort:

```
// If there is only one element in the list to sort, then
// by definition, it is already well sorted.
if (NumWords < 2)
{
      return;
}
```

Fix a constant *MinMergeSize.* Modify MergeSort to use InsertionSort when there are fewer than *MinMergeSort* items to sort, and the regular MergeSort routine (i.e., the recursive sorting and merging) when there are at least *MinMergeSort* items to sort.

For this part, submit only the modified version of MergeSort, along with any other changes to VectorTextFile3.java. (Do not submit the entire class file.)

Problem 4.c. Run the revised version of MergeSort on hamlet.txt. Profile the program to determine how long it takes to perform the MergeSort. How long does it take to sort when *MinMergeSort* $= 0$? How about when *MinMergeSort* $= 100$? For what value of *MinMergeSort* is the sorting fastest? (You will have to experiment with several values.)

Problem 4.d. We will now analyze the asymptotic cost of the modified MergeSort. For the purpose of the remaining parts, let n be the number of items to sort and let $k =$ *MinMergeSort.*

What is the asymptotic cost (in big-O notation) of running InsertionSort once on k items? How many times is InsertionSort executed (on k items, each time)? What is the asymptotic cost of all the calls to InsertionSort?

Problem 4.e. What is the total asymptotic cost of the modified MergeSort? What is the largest value of k for which the modified MergeSort is still O(n log n)?

Problem 5.

Professor Sara H. Acker is running an election to discover her students' favorite algorithm. Each student writes his/her favorite algorithm on a piece of paper, and at the end of class, Professor Acker has a large pile of n papers. She wants to know whether any of the algorithms received more than $n/2$ of the votes, and if so, which one. The goal of this problem is to determine how she can answer this question efficiently.

For this purpose of this problem, assume that there are n students and m different algorithms. (Note that n and m may be very large.) Assume that the votes are tabulated in an array V of size n. That is, for student s, the location $V[s]$ holds the vote of student s. We say that an algorithm A is the winner if $> n/2$ students voted for algorithm A, i.e., $|\{i : V[i] = A\}| > n/2$.

The simplest solution to finding a winner is to count how many times each element appears in the array. However, this requires at least m space to store the m integers for counting the m different algorithms. However, Professor Acker has no scrap paper available. She only has enough extra space to write down $O(\log n)$ integers. (Note that m may be larger than $\log n$.)

Problem 5.a. Explain (briefly) how to solve Professor Acker's problem using sorting.

Problem 5.b. Assume that Professor Acker cannot sort. In fact, she cannot modify the array V (as it is read-only), and remember, she has very little extra space to work with: she can only store $\Theta(\log n)$ additional integers.

Give an algorithm that determines whether some algorithm is a winner, and if so, which one. In your solution, give:

- an outline (in words) of how your algorithm works,
- pseudocode for your algorithm,
- a proof that your algorithm works,
- an analysis of the asymptotic performance of your algorithm (using big-O notation).

Present the most efficient algorithm you can.

Hint: There is an efficient divide-and-conquer solution to this problem. Find this solution first. There is also a harder, but more efficient, algorithm that is not as obviously based on divide-and-conquer.

Problem 5.c. Unfortunately, it turns out that no algorithm is a winner. Now, Professor Acker wants to know whether any algorithm received at least $n/4$ votes. (Note that more than one algorithm may have received this many votes, but she will be happy if you can identify even one of them.) Explain how Professor Acker might solve this problem. How long would it take to find an algorithm that gets at least n/k votes?