CS2351 Data Structures

Homework 1

Due: 11:59 pm, March 15, 2010

In this assignment, you may choose to work alone or work in pairs, though the minimum requirements will be different. If you work in pairs, please submit your file together as one copy, and mark clearly your names and your university numbers in the submitted file.

Part I. Written (50%)

1. Your friend, John, has given you an array $A[1..n]$ of n numbers. He told you that there is some i such that $A[1..i]$ is straightly increasing, and $A[i..n]$ is straightly decreasing. This implies that $A[i]$ is the maximum entry in the array.

   (15%) Design an $O(\log n)$-time algorithm to find this maximum entry, and explain why it runs in $O(\log n)$ time. Also briefly show the correctness of your algorithm.

2. Bubble Sort is a very simple algorithm that can sort an array $A[1..n]$ of n numbers. It works in $n-1$ rounds as follows:

   ```
   for (round j = 1, 2, ..., n − 1) {
       for (position i = 1, 2, ..., n − j) {
           if (A[i] > A[i + 1])
               Swap A[i] with A[i + 1];
       }
       if (there is no swapping in a round)
           Break the for-loop so that we do nothing in the remaining rounds;
   }
   ```

   (5%) Show that Bubble Sort is correct.
   Hint: What happens to the last entry after the first round?)

   (5%) Show that the running time of Bubble Sort is $O(n^2)$.

   (5%) Show that in the worst case, the running time of Bubble Sort is $\Omega(n^2)$.
   Hint: Can you find an input array whose running time is $\Omega(n^2)$.

   (5%) The previous two results imply that the worst-case running of Bubble Sort is $\Theta(n^2)$. However, the running time of Bubble Sort is **not** $\Theta(n^2)$. Can you explain why?
   Hint: What is the best-case running time?

3. Your friend, John, now gives you another array $B[1..n]$ that contains only positive integers. He wants to know if there is a contiguous portion of the array, $B[i..j]$, such that the sum of all its entries is equal to the number Y.

   One way to solve this is: For each pair of (i, j), compute the sum of all entries in $B[i..j]$ and check if it is equal to Y.

   Unfortunately, this algorithm is very inefficient, as there are $\Theta(n^2)$ sums to test, and each sum involves $O(n)$ addition operations. A careful analysis show that it will take $\Theta(n^3)$ running time in total.

(15%) Design an $O(n)$-time algorithm to help John find the desired portion $B[i..j]$, and explain why it runs in $O(n)$ time. Also briefly show the correctness of your algorithm.

Note: If your algorithm does not run in $O(n)$ time, but runs in $O(n^2)$ time, your maximum score for this question is 10% (instead of 15%). Otherwise, if it runs in $\omega(n^2)$ time, your score is 0%.

Part II. Programming (35%)

In this part, you are asked to implement the Bubble Sort algorithm, and the Binary Search algorithm. If you work alone, you can choose any one algorithm to implement. If you work in pairs, you will need to implement both algorithms.

You can develop your code using either C or C++. Your code should be compilable using Dev-C++. We will assess your code by the correctness (25%) and the readability (10%). Note that you should not use any predefined system calls or standard template library calls to perform the Bubble Sort or the Binary Search.

A. Specification of Bubble Sort:

Your program will expect two arguments when it runs. The first one gives the filename of the input file that contains the number to be sorted, and the second one gives the filename of the output file where the sorted numbers are stored. For example, suppose your program name is `bubble`, the input filename is `input.txt`, and the output filename is `output.txt`. Then when we run your program, we will type: `./bubble input.txt output.txt` so that your program sorts the numbers in the `input.txt` and store the result in `output.txt`.

The format of the input file is as follows: The first line stores an integer n that indicates how many numbers are to be sorted. The second line stores n integers which are the numbers to be sorted. For example, the input file may look like:

```
10
14  36  5  2  8  100  13  -12  4  -13
```

The format of the output file is simply the sequence of the sorted numbers, each separated by a single space. For example, the output file of the previous example should look like:

```
-13  -12  2  4  5  8  13  14  36  100
```

B. Specification of Binary Search:

Your program will expect two arguments when it runs. The first one gives the filename of the input file that contains a sorted list of integers, together with a couple of binary search queries, and the second one gives the filename of the output file where the answer of the queries are stored. For example, suppose your program name is `bsearch`, the input filename is `input.txt`, and the output filename is `output.txt`. Then when we run your program, we will type: `./bsearch input.txt output.txt` so that your program sorts the numbers in the `input.txt` and store the result in `output.txt`.

The format of the input file is as follows: The first line stores an integer n that indicates how many numbers are in the sorted list. The second line stores the sorted list of n integers. The third line stores an integer m that specifies the number of binary search queries. After that, it is followed by m lines, each line containing two integers x and y. For example, the input file may look like:

```
10
-5  -3  0  1  3  5  7  8  13  14
3
5  8
4  14
16  18
```

For a query with the number x and y, it reports the number of integers in the list whose value is between x (inclusive) and y (inclusive). The format of the output file contains m lines, such that the ith line storing the answer of the ith query. For example, the output file of the previous example should look like:

```
3
5
0
```

Your program should answer each query in $O(\log n)$ time. That is, you cannot use the brute-force scanning to count the number of integers between the query numbers x and y.

Part III. Experiments (15%)

In this part, you are asked to test the running time of several sorting algorithms on your machine. If you work alone, you can compare Merge Sort and any one other sorting algorithms (such as Bubble Sort, Selection Sort, Insertion Sort, ...). If you work in pairs, you will need to compare Merge Sort, Bubble Sort, and any one other sorting algorithms.

To get a better understanding of which algorithm is the best on your machine, we shall test the sorting algorithms on input array of various sizes (such as 10, 100, 1000, 10000, ...). For each size n, we shall generate a few random arrays of size n, and get the average time of running the sorting algorithm on these arrays. Include the following details in your submitted file:

- The configuration of your machine (such as machine model, CPU speed, RAM, OS, ...).
- Indicate which algorithms you have tested, and how you implement them. Also indicate which compiler you use (and the compilation options if any).
- Indicate the number of random arrays where you take the average timing.
- Explain how you generate the random arrays.
- Plot your timing results in a graph, and estimate roughly for which input size which algorithm is the best.

The purpose of the above is to allow people, who are interested in your work, to repeat your experiments and obtain the same results.