CSCI 3104, Algorithms **Prof. Lijun Chen**
Problem Set 1 **Fall 2017, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

---

1. (10 pts) You are given two arrays of integers A and B, both of which are sorted in ascending order. Consider the following algorithm for checking whether or not A and B have an element in common.

```
findCommonElement(A, B) :
    # assume A,B are both sorted in ascending order
    for i = 0 to length(A) {                    # iterate through A
        for j = 0 to length(B) {                # iterate through B
            if (A[i] == B[j]) { return TRUE }  # check pairwise
        }
    }
    return FALSE
```

   (a) If arrays A and B have size n, what is the worst case running time of the procedure `findCommonElement`? Provide a Θ bound.

   (b) For $n = 5$, describe input arrays A_1, B_1 that will be the best case, and arrays A_2, B_2 that will be the worst case for `findCommonElement`.

   (c) Write pseudocode for an algorithm that runs in $\Theta(n)$ time for solving the problem. Your algorithm should use the fact that A and B are sorted arrays.
   (Hint: repurpose the `merge` procedure from MergeSort.)

2. (15 pts) Suppose we are given an array A of historical stock prices for a particular stock. We are asked to buy stock at some time i and sell it at a future time $j > i$, such that both $A[j] > A[i]$ and the corresponding profit of $A[j] - A[i]$ is as large as possible.

   For example, let $A = [7, 3, 4, 2, 15, 11, 16, 7, 18, 9, 11, 10]$. If we buy stock at time $i = 3$ with $A[i] = 2$ and sell at time $j = 8$ with $A[j] = 18$, we make the maximum profit of $18 - 2 = 16$ megabucks. (Note that "short positions," where we sell stock before buying it back, i.e., where $j < i$, is not allowed here.)

   (a) Consider the pseudocode below that takes as input an array A of size n:

```
makeMaxProfitInHindsight(A) :
    maxProfitSoFar = 0
    for i = 0 to length(A)-1 {
        for j = i+1 to length(A) {
            profit = A[j] - A[i]
            if (profit > maxProfitSoFar) { maxProfitSoFar = profit }
    }}
    return maxProfitSoFar
```

What is the running time complexity of the procedure above? Write your answer as a Θ bound in terms of n.

(b) Explain under what circumstances the algorithm in (2a) will return a profit of 0. Two sentences should suffice in your answer.

(c) Write pseudocode that would calculate a new array B of size n such that

$$B[i] = \min_{0 \le j \le i} A[j] \ .$$

In other words, $B[i]$ should store the minimum element in the subarray of A with indices from 0 to i, inclusive.

What is the running time complexity of the pseudocode to create the array B? Write your answer as a Θ bound in terms of n.

(d) Use the array B computed from (2c) to compute the maximum profit in time $\Theta(n)$.

(e) Rewrite the algorithm above by combining parts (2b)–(2d) to avoid creating a new array B.

3. (15 pts) Consider the problem of linear search. The input is a sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a target value v. The output is an index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

(a) Write pseudocode for a simple linear search algorithm, which will scan through the input sequence A, looking for v.

(b) Using a loop invariant, prove that your algorithm is correct. Be sure that your loop invariant and proof covers the initialization, maintenance, and termination conditions.

4. (15 pts) Gandalf and Elrond are arguing about binary search. Elrond writes the following pseudocode on the board, which he claims implements a binary search for a target value `v` within input array `A` containing n elements.

```
bSearch(A, v) {
   return binarySearch(A, 0, n, v)
}

binarySearch(A, l, r, v) {
   if l >= r then return -1
   p = floor( (l + r)/2 )
   if A[p] == v then return m
   if A[m] < v then
     return binarySearch(A, m+1, r, v)
     else return binarySearch(A, l, m-1, v)
}
```

(a) Help Gandalf determine whether this code performs a correct binary search. If it does, prove to Elrond that the algorithm is correct. If it is not, state the bug(s), give line(s) of code that are correct, and then prove to Elrond that your fixed algorithm is correct.

(b) Elrond tells Gandalf that binary search is efficient because, at worst, it divides the remaining problem size in half at each step. In response Gandalf claims that tri-nary search, which would divide the remaining array A into thirds at each step, would be *even more* efficient. Explain who is correct and why.