CSCI 3104 Fall 2017 Problem Set 1
Austin Griffith (09/09)

---

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 bound.*

   The outer for-loop must run n times and the inner for-loop must run n times so the run time of findCommonElement is $\Theta(n^2)$

(b) *For n = 5, describe input arrays A1, B1 that will be the best case, and arrays A2,B2 that will be the worst case for findCommonElement*

For n=5 size array, the A and B array will have the best case run time if:

A = [1,2,3,4,5] and B = [1,6,7,8,9]

For n=5 size array, the A and B array will have the worst case run time if

A = [1,2,3,4,5] and B = [-3,-2,-1,0,5] - worst case is when matching values in the array at the very end and best case is when the matching values are at the very beginning of the array.

(c) *Write pseudocode for an algorithm that runs in* $\Theta(n)$ *time for solving the problem. You algorithm should repurpose the merge procedure from MergeSort.)*

A and B are sorted arrays in ascending order

```
FindCommonElement(A,B)
{   i=0 , j=0
    while(i=! length(A) and j=! length(B))
    {
        if(A[i] == B[j]){return True}
        else(A[i] < B[j]){i++}
        else(A[i] > B[j]){j++}
    }
    return False;
}
```

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 bound in terms of n.

   The function has two for-loops, one inner and one outer. For each value of the outer loop, the inner loop runs n times so the total run time of this function is $\Theta(n^2)$

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

   If all the elements in the array are in non-increasing order where all the elements are not equal or or decreasing as you get further in the array, then this Algorithm will always produce a max profit equal to zero. ex) A = [100,90,89,80,70]

(c) *Write pseudocode that would calculate a new array B of size n such that B[i] = min 0 j 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 bound in terms of n.*

```
MimimunPriceIndex = 0   //maintains an index of min for 0 --> i
B[0] = A[MinimumPriceIndex]
for(i=1 to length(A)-1)
{
    if(A[i] < A[MinimumPriceIndex])
    {
        MinimumPriceIndex = i   //stores index of array A not actual value
        B[i] = A[MinimunPriceIndex]
    }
}
```

The run time of this Algorithm is $\Theta(n)$

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

```
def makeMaxProfit(A,B)
    maxProfitSoFar = 0
    for(i=0 to length(A)-1)
    {
        profit = A[i] - B[i]
        if(profit > maxProfitSoFar)
        {
            maxProfitSoFar = profit
        }
    }
    return maxProfitSoFar
```

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

```
def makeMaxProfit(A)
        maxProfitSoFar = 0
        minPrice = A[0]
        for(i=1 to length(A)-1
        {
            profit = A[i] - minPrice
            if(profit > maxProfitSoFar)
            {
                minPrice = A[i]
            }
```

```
}
return maxProfitSoFar
```

To make max profit you must buy at the lowest price and sell at the highest price on the condition that the buy must take place before the sell. The run time of this Algorithm $\Theta(n)$

3. *(15 pts) Consider the problem of linear search. The input is a sequence of n numbers A = a1, a2, . . . , ai 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.*

   Inputs:
   A array of values
   v target value

```
linearSearch(A,v)
{
    int i;
    for(i=0 to length(A)-1)
    {
        if(A[i] == v)
        {
            return i ;
        }
    }
    return NIL
}
```

   (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.*

   Initialization:
   The array A[ ] contains a subarray A[b[ ]] that contains all the searched values of A such that A=[0,1,2,3] and A [b[0,1,2] 3].

   Maintenance:
   On each iteration, the subarray b[0....n] get bigger as A[0...n] is searched for the target value v.

   Termination:
   Upon termination if A[i] == v, the loop will terminate and output i (the index of array A[ ]) or will output the the special case NIL where the target value v was never found and array A[ ].

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.*

Shown below is the correct algorithm. In this Algorithm a few changes have been made. The value m to p and also now searching through n-1 as to not access out of bounds memory.

```
int binarySearch(int A[], int l, int r, int v, int n)
 {
    if (l >= r)  //check to see if subtree is empty
    {
        return -1;
    }

    int p = floor((l + r)/2);
    //find middle of subtree and check to see if first value is V
```

```
        if (A[p] == v) //if value is found return index
        {return p;}

        if (A[p] < v) //if value is less than target value go right
        {
            return binarySearch(A, p+1, r, v,n);
        }
        else   //else go left
        {
            return binarySearch(A, l, p-1, v,n);
        }
}

int bSearch( int A[], int v, int n)
{
    return binarySearch(A, 0, n-1, v, n);
}
```

loop invariant: a[l] <= v <= a[r] if target exits in A

Initialization: l = 0 r = last index in array therefor target must be in A[l,r] which is A

Maintenance:
3 cases:

if A[p] < v we update l = p +1 which means A[l] <= v so loop invariant holds

if A[r] > v we update r = p -1 which means A[p] >= v so loop invariant holds

if A[p] == target algorithm returns p which is the intended behavior of algorithm

Termination:

upon loop termination, if the target is found: index of target gets returned or l ¿ r and target not found. If not found algorithm returns -1, which is the intended behavior

(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.*

Elrond is actually correct. A trinary search wouldn't work on a binary tree. If you divided the tree in to thirds p = ((l+r)/3) you would not be able to cover the entirety of the tree.