CSCI 3104 Spring 2018
Problem Set 1

Bailon, Erika

09/28

NUMBER 1.

(a.) $n + 1 = O(n^4)$

$\lim_{n\to\infty} \frac{\partial(n+1)}{\partial(n^4)} \to \lim_{n\to\infty} \frac{1}{n^3} + \frac{1}{n^4} = 0$

Because the limit is equal to 0, then it means that then $f(n) = O(g(n))$. Therefore, the statement is **TRUE**, however, it is not the tightest bound.

(b.) $2^{2n} = O(2^n)$

$\lim_{n\to\infty} \frac{2^{2n}}{2^n} \to \lim_{n\to\infty} \frac{e^{(2ln2)n}}{e^{(ln2)n}} \to \lim_{n\to\infty} e^{2ln2-ln2} = \infty$

Because the limit goes to infinity, it means that $f(n) = \Omega(g(n))$. Therefore, the statement is **FALSE**.

(c.) $2^n = \Theta(2^{n+7})$

$\lim_{n\to\infty} \frac{2^n}{2^{n+7}} \to \lim_{n\to\infty} \frac{2^n}{2^n 2^7} = \frac{1}{2^7} = \frac{1}{128}$

Because the limit is a constant, it means that $f(n) = \Theta(g(n))$. Therefore the statement is **TRUE**.

(d.) $1 = O(\frac{1}{n})$

$\lim_{n\to\infty} \frac{1}{\frac{1}{n}} \to \lim_{n\to\infty} n = \infty$

Because the limit is infinity, it means that $f(n) = \Omega(g(n))$. Therefore the statement is **FALSE**.

(e.) $ln^2 n = \Theta(lg^2 n)$

$\lim_{n\to\infty} \frac{ln^2 n}{lg^2 n} \to \lim_{n\to\infty} \frac{ln^2 n}{\frac{ln^2 n}{ln^2 2}} = ln^2 2$

Because the limit is a constant, it means that $f(n) = \Theta(g(n))$. Therefore the statement is **TRUE**.

(f.) $n^2 + 2n - 4 = \Omega(n^2)$

$\lim_{n\to\infty} \frac{n^2+2n-4}{n^2} \to \lim_{n\to\infty} \frac{2n}{2n} = 1$

Because the limit is a constant, it means that $f(n) = \Theta(g(n))$. Therefore the statement is **FALSE**.

(g.) $3^(3n) = \Theta(9^n)$

$\lim_{n\to\infty} \frac{3^{3n}}{9^n} \to \lim_{n\to\infty} \frac{3^{3n}}{(3^2)^n} \to \lim_{n\to\infty} 3^n = \infty$

Because the limit goes to infinity, it means that $f(n) = \Omega(g(n))$. Therefore, the statement is **FALSE**.

(h.) $2^{n+1} = \Theta(2^{nlg(n)})$

$\lim_{n\to\infty} \frac{2^{n+1}}{2^{nlgn}} \to \lim_{n\to\infty} 2^{(n+1)-(nlg(n))}$

Since we have 2 to the power of something we look at the behavior of that something to determine what the limit is. The derivative of $n + 1 - nlg(n) = 1 - (1lg(n) + n(\frac{1}{n})\Big(\frac{1}{ln(2)}\Big)$ which will go to ∞. Therefore $\lim_{n\to\infty} 2^\infty = \infty$

Because the limit is infinity, it means that $f(n) = \Omega(g(n))$. Therefore the statement is **FALSE**.

(i.) $\sqrt{n} = O(lg(n))$ which is the same as $n^{1/2} = O(lg(n))$

$\lim_{n\to\infty} \frac{n^{1/2}}{lg(n)} \to \lim_{n\to\infty} \frac{n^{1/2}}{\frac{ln(n)}{ln(2)}} \to \lim_{n\to\infty} \frac{n^{1/2}ln(2)}{ln(n)}$ Taking partial derivative

$\lim_{n\to\infty} \frac{\partial n^{1/2}ln(2)}{\partial ln(n)} \to \lim_{n\to\infty} \frac{\frac{1ln(2)}{2}n^{-1/2}}{\frac{1}{n}} \to \lim_{n\to\infty} \Big(\frac{1ln(2)}{2}\Big)\Big(\frac{n^{1/2}}{1}\Big) = \infty$

Because the limit is infinity, it means that $f(n) = \Omega(g(n))$. Therefore the statement is **FALSE**.

(j.) $10^{100} = \Theta(1)$

$\lim_{n\to\infty} \frac{10^{100}}{1} = 10^{100}$

Because the limit is a constant, it means that $f(n) = \Theta(g(n))$. Therefore the statement is **TRUE**.

NUMBER 2.

(a.) What is the running time complexity of the procedure?

We have a nested for loop, therefore we get:

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n} c_1 \to c_1\Big(\sum_{i=0}^{n-1}\sum_{j=i+1}^{n}\Big) \to c_1\Big(\sum_{i=0}^{n-1} n\Big) \to c_1\Big(\Big(\frac{n-1}{2}\Big)n\Big) \to c_1\Big(\frac{n^2-n}{2}\Big)$$

$\to c_1\frac{1}{2}n^2 - c_1 n$ We don't care about the constants $\Rightarrow \mathbf{\Theta(n^2)}$.

(b.) The makeWizardMoney algorithm will return 0 coins with either of these 2 conditions.
1^{st} The array is sorted in descending order. This will cause the operation A[j] - A[i] be a negative number which will not be a greater number than 0, which is the value given at the beginning of the code for maxCoinsSoFar, and therefore it will return 0 because we will never replace the value of maxCoinsSoFar with a negative number that is smaller than 0.
2^{nd} The elements in the array are the same value. For example if we have an array that is only $\langle 6, 6, ..., 6, 6, \rangle$, when the code runs coins = A[j] - A[i], it will always be equal to zero. Therefore, the program will return 0.

(c). The running time complexity of the pseudocode to create the array M is $O(n)$. Since we are using the function $M[i] = \min_{0 \leqslant j \leqslant i} A[j]$, we have a linear operation that will go through the array n times. Therefore $O(n)$ is the time complexity and because it is guaranteed that it will run n times we get $\Theta(n)$.

(d.) The answer is 5. Because we would end up with the array $[8, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]$, and since we are doing a modified code with complexity $\Theta(n)$, we can do the max value - the min value, which will be $8 - 3 = 5$.

(e.)

```
makeWizardMoney(A):
     maxCoinsSoFar = 0
     tempM = A[i]
     temp = A[i]
     for i = 0 to lenght(A) -1
     {
            if(A[i] < A[i+1] && temp >= A[i])
                   temp = A[i]
            else if (A[i] > A[i+1] && tempM < A[i])
                  tempM = A[i]
     }
     maxCoinsSoFar =  tempM - temp
     return maxCoinsSoFar
```

**The code is provided at the end of the problem set to prove that it runs

NUMBER 3.

(a.)

```
SLSA (A,v):
      temp = v         \\I do this just to keep the original value safe
      for i = 0 to lenght(A)-1
      {
            if (A[i] == temp)
                  return A[i]
      }
      return NIL
```

(b.) Initialization. For initialization we have a base case. Let's say the array contains only 1 element, this is $n = 1$. Then the loop goes through the first iteration and compares if the value is in position 0 of the array, since we are starting our $i = 0$. If the only element in the array is the target value, then we will exit the loop and we will get an answer. If the only element of the array is not the target value, we increment i but will not go through the loop since $1 - 1 = 0$. Then the code falls into the return statement out of the loop and we will get NIL. Therefore the loop invariant holds.
Maintenance. The $forloop$ inside the functions is starting at position 0, which we have taken as out base case and it works. Not lets say we have an array of size n. The $forloop$ will starts at 0 and increment the position (i) by 1. This means that after the first iteration (which was our base case), $i = i + 1$ and now the condition if will check the new value of i. If $A[i] == targetvalue$ then it will go to $returnA[i]$. If is not the $targetvalue$, then it will increment again i by 1 and so on, repeating the same process until it finds, or doesn't, the $targetvalue$.
Termination. There are two different conditions that will cause the termination of the $forloop$. 1^{st})The target value is found while $i \leq length(A) - 1$ and it $returnsA[i]$.
2^{nd})The $targetvalue$ is not found in the range $i \leq length(A) - 1$ causing i to be equal to n or greater than n and that position is not in the array, therefore it does not go through the $forloop$ and it falls into the $returnNIL$ statement. Therefore, the algorithm is **correct**.

NUMBER 4.

(a.) **Errors I found

```
bSearch(A, v)
{
     return binarySearch(A, 1, n-1, v)   *It cannot go from (1 to n-1)
                              is it either from  (0 to n-1), or  (1 to n)
}
binarySearch(A, l, r, v)
{
```

```
       if l <= r then return -1    **The inequality should not be <=, it
                    needs to be l > r so it goes through the entire array
       p = floor( (l + r)/2 )
       if A[p] == v then return p
       if A[p] < v then
          return binarySearch(A, p+1, r, v)
          else return binarySearch(A, l, p-1, v) *** I do not know if this
              was in purpose or not, but the else statement should NOT be
                 indented as the same indentation as the return statement
   **** Lastly, there is no curly brace to end the binarySearch function.

 Those are the 4 errors I found.
```

The correct code should be:

```
bSearch(A,v)
{
    return binarySearch(A, 0, n-1, v)
}
binarySearch(A, l, r, v)
{
    if l > r then return -1
    p = floor( (l + r)/2 )
    if A[p] == v then return p
    if A[p] < v then
         return binarySearch(A, p+1, r, v)
    else return binarySearch(A, l, p-1, v)
}
```

**The code that runs on c++ is also attached at the end of the problem set.

To prove that the algorithm in correct:
Initialization. Lets have our base case when the array has one element. This means $n = 1$. It will go to the operation p which will give 0 so the first if statement is going to check the position 0 of the array, if it is the *targetvalue* then it will return p. If is not the *targetvalue* it will go to the second $if statement$ to check if the value in the position is smaller than the *targetvalue*. Either if it is, or is not, the function *binarySearch* is called again, which will go to check if $l\ r$ and since this condition will be true, the function will $return - 1$. So it work for our base case.
Maintenance. After our base case, we have that n will be greater or equal to 2. *binarySearch* will start and divide the array into 2. After grabbing the value p, it will check if it is the *targetvalue* is the same as the value in the position $A[p]$, if it is will return p, if is not, it will check if the value of the element in the position $A[p]$ is smaller than the *targetvalue*, if it is, it will change the range of search, throwing away the half where for sure the *targetvalue* is not located, then the part of the array where the *targetvalue* might be, will go through

binarySearch again until it finds the value. The function will keep getting rid of the part of the array where for sure, the *targetvalue* would NOT be found. The range that is changed each time, make sure that it grabs the position where the array was cut, that way we have the guarantee that each element of the array will be checked. The same process occurs if the value $A[p]$ is greater then the *targetvalue* since it will get rid of the right half of the array and will continue to change the range and checking the point p with each partition to make sure it goes through the entire array and it gets rid of the partition of the array where the *targetvalue* is NOT.
Termination. The *binarySearch* will stop if one of these 2 cases happens.
1^{st} The function called itself the maximum number of times so it will return -1.
2^{nd} When the *targetvalue* is located before the function reaches its maximum times of recursion.

(b.) Recursive binary search has atomic operation and 2 operation that are (1/n) since the array is cut in half each iteration. The fourth-nary search has twice the number of atomic operations as the binary search. However, because this numbers are constants, it does not affect the time complexity of the linear function. Therefore they both are $\Theta(n)$.

NUMBER 5. (ExC) . The worst case scenario would be $\Theta(n^2)$
It contains 2 for loops. Therefore, the worst case scenario is to suppose that the arrays do not contain a common element. Both of the loops will run up to n, the next loop will run. to n-1. Giving us a time complexity of n^2.

(b.) We would have the best case when A_1 and B_1 have the same value at positions $A[0]$ and $B[0]$ since the function will go once and return true in the first case.
The worst case scenario would be when A_1 and B_1 do not have any common element and the code will run 5 times and then return false.

(c.)

```
findCommonElement(A, B) :
    A1 = 0, A2 = 0, A3 = 0
    while(i < length(A) and j < length(A))
    {
        if(A[A1] < B[A2])
            C[A3++] = A[A1++]
        else
            C[A3++] = B[A2++]
    }
    Var = 0
    while(Var + 1 < length(C))
```

```
        if(C[Var] == C[Var +1])
            return TRUE
    Var++
return FALSE
```

WORKED with Eric Oropeza Elwood and Selena Quintanilla and George Allison

**CODES

```
#include <iostream>
#include <sstream>
#include <fstream>
using namespace std;

int main()
{
     int A[12] = {8, 9, 1, 4, 14, 12, 15, 22, 7, 8, 12, 11};

     int maxCoinsSoFar = 0;
     int i = 0;
     int tempM = A[i];
     int temp = A[i];
     for (i=0; i <=(11); i++)
     {
          if(A[i] < A[i+1] && temp >= A[i])
               temp = A[i];
          else if (A[i] > A[i+1] && tempM < A[i])
               tempM = A[i];
     }
     maxCoinsSoFar =  tempM - temp;
     cout << maxCoinsSoFar<< endl;
     return maxCoinsSoFar;
}
```

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <math.h>
using namespace std;

int bSearch(int A[],int v);
int binarySearch(int A[], int l , int r, int g);

int main()
{
int A[1] = {20};
int v = 20;
int result = bSearch(A, v);

   return 0;
}
```

```
int bSearch(int A[],int v)
{
int x =  binarySearch(A, 0, 9, v);
cout << x << endl;
return x;

}

int binarySearch(int A[], int l , int r, int g)
{
if(l > r)
return -1;
int p = floor((l+r)/2);
if(A[p] == g)
return p;
if(A[p] < g)
return binarySearch(A, p+1, r, g);
else
return binarySearch(A, l, p-1, g);

}
```