CSCI 3104 Spring 2018
Problem Set 1
Merola, Michael
06/04/1998

# Problem Set 1

## Problem 1

**(10 pts total) For each of the following claims, determine whether they are true or false. Justify your determination. If the claim is false, state the correct asymptotic relationship.**

**(a)** $\mathbf{n + 1 = O(n^4)}$

$$\lim_{n\to\infty}\left[\frac{n+1}{n^4}\right] = \lim_{n\to\infty}\left[\frac{1}{4\,n^3}\right] = \frac{1}{\infty} = 0$$

$\therefore$ TRUE

**(b)** $\mathbf{2^{2\,n} = O(2^n)}$

$$\lim_{n\to\infty}\left[\frac{2^{2\,n}}{2^n}\right] = \lim_{n\to\infty}\left[\frac{(2^n)^2}{2^n}\right] = \lim_{n\to\infty}[2^n] = \infty$$

$\therefore$ FALSE, $n + 1 = \Omega(n^4)$

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

$$\lim_{n\to\infty}\left[\frac{2^n}{2^{n+7}}\right] = \lim_{n\to\infty}\left[\frac{2^n}{2^n * 2^7}\right] = \frac{1}{2^7}$$ *constant

$\therefore$ TRUE

**(d)** $\mathbf{1 = O\left(\frac{1}{n}\right)}$

$$\lim_{n\to\infty}\left[\frac{1}{\frac{1}{n}}\right] = \lim_{n\to\infty}[n] = \infty$$

$\therefore$ FALSE, $1 = \Omega\left(\frac{1}{n}\right)$

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

$\lim_{n\to\infty}\left[\frac{\ln^2 n}{\lg^2 n}\right] = \lim_{n\to\infty}\left[\left(\frac{\ln n}{\lg n}\right)^2\right] = \lim_{n\to\infty}\left[\left(\frac{\lg n / \lg e}{\lg n / \lg 2}\right)^2\right] = \lim_{n\to\infty}\left[\left(\frac{\lg 2}{\lg e}\right)^2\right] =$ *some constant*

$\therefore$ TRUE

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

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

$\therefore$ FALSE, $n^2 + 2n - 4 = \Theta\left(n^2\right)$

**(g) $3^{3n} = \Theta(9^n)$**

$\lim_{n\to\infty}\left[\frac{3^{3n}}{9^n}\right] = \lim_{n\to\infty}\left[\frac{(3^n)^2 * 3^n}{(3^n)^2}\right] = \lim_{n\to\infty}[3^n] = \infty$

$\therefore$ FALSE, $3^{3n} = \Omega(9^n)$

**(h) $2^{n+1} = \Theta\left(2^{n\lg n}\right)$**

$\lim_{n\to\infty}\left[\frac{2^{n+1}}{2^{n\lg n}}\right] = \lim_{n\to\infty}\left[\frac{2^n * 2}{(2^n)^{\lg n}}\right] = \lim_{n\to\infty}\left[\frac{(2^n)^{1-\lg n}(2)}{1}\right] = \lim_{n\to\infty}\left[\frac{2^{n-n\lg n+1}}{1}\right] = 0$

$\therefore$ FALSE, $2^{n+1} = O\left(2^{n\lg n}\right)$

**(i) $\sqrt{n} = O(\lg n)$**

$\lim_{n\to\infty}\left[\frac{\sqrt{n}}{\lg n}\right] = \lim_{n\to\infty}\left[\frac{1/2 * n^{-1/2}}{1/n}\right] = \lim_{n\to\infty}\left[\frac{n}{2n^{1/2}}\right] = \lim_{n\to\infty}\left[\frac{n^{1/2}}{2}\right] = \infty$

$\therefore$ FALSE, $\sqrt{n} = \Omega(\lg n)$

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

$\lim_{n\to\infty}\left[\frac{10^{100}}{1}\right] = 10^{100}$ *f does not rely on n ; RAM model for O(1) constant

$\therefore$ TRUE

# Problem 2

**(15 pts) Professor Dumbledore needs your help optimizing the Hogwarts budget. You'll be given an array A of exchange rates for muggle money and wizard coins, expressed at integers. Your task is help Dumbledore maximize the payoff by buying at some time i and selling at a future time j > i, such that both A[j] > A[i] and the corresponding difference of A[j] – A[i] is as large as possible.**

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

```
makeWizardMoney(A) :
    maxCoinsSoFar = 0
    for i = 0 to length(A)-1 {
        for j = i+1 to length(A) {
            coins = A[j] - A[i]
            if (coins > maxCoinsSoFar) { maxCoinsSoFar = coins }
    }}
    return maxCoinsSoFar
```

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

*let $n$ = length(A)
**the code has 2 for-loops based on array size $n$

so... $\sum^{n-1}_{i=0}[\sum^{n}_{j=j+1}[A[j]+A[i]]]$

∴ time complexity is $\Theta(n^2)$

**(b) Explain under what conditions on the contents of A the *makeWizardMoney* algorithm will return 0 coins.**

The algorithm will return 0 coins if A contained all ints of the same value *(ex.. A = [5, 5, 5, 5, 5])*, and if A was sorted entirely in decreasing order *(ex.. A = [5, 4, 3, 2, 1])*.

**(c) Dumbledore knows you know that makeWizardMoney is wildly inefficient.**

**With a wink, he suggests writing a function to make a new array M of size n such that...**

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

**That is, M[i] gives the minimum value in the subarray of A[0::i].**

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

To create array M, we would need to implement a for-loop into the code that would run through the length of A and add the nessecary elements to array M. Because the code only requires one loop based on size *n*, the time complexity is Θ(*n*).

**(d) Use the array M computed from (2c) to compute the maximum coin return in time Θ(n).**

By using the M array, the maximum coin return would be 16 coins *(19 max - 3 min)*.

**(e) Give Dumbledore what he wants: rewrite the original algorithm in a way that combine parts (2b)(2d) to avoid creating a new array M.**

```
def makeWizardMoney-new(A) {
    maxCoinsSoFar = 0
    min = A[0]
    for (int i = 0; i < length(A)-1; i++) {
        if (A[i] < min)
            min = A[i]
        if (A[i] - min > maxCoinsSoFar)
            max = A[i] - min
    }
    return maxCoinsSoFar
}
```

# Problem 3

**(15 pts) Consider the problem of linear search. The input is a sequence of *n* numbers A = [ $a_1, a_2, \ldots, a_n$] 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*.**

```
def linearSearch(A, v) {
    var f = NIL
    for (int i = 0; i < length(A)-1; i++) {
        if (A[i] == v)
            f = i
            return f
    }
    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.**

Loop Invariant: The variable *f* will always contain either NIL or the index i such that v=A[i].

Initialization: f = NIL ; initialized to NIL in the beginning of the algorithm.
Maintenance: During every loop, *f* either remains NIL or f = i if v=A[i]
Termination: If *v* is not in A, f = NIL
If *v* is in A, f = i

# Problem 4

**(15 pts) Crabbe and Goyle are arguing about binary search. Goyle 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) {
1,2    return binarySearch(A, 1, n-1, v)
}
binarySearch(A, l, r, v) {
3    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)
}
```

**(a) Help Crabbe determine whether this code performs a correct binary search. If it does, prove to Goyle 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 Goyle that your fixed algorithm is correct.**

Potential Bugs:

1) The return statement cuts out the first index; should be *return binarySearch(A, 0, n-1, v)*.

2) n is never defined; should define above return statement as *n = length(A)*.

3) Left bound is always less than right so the code will never run; should be *if (L > R) then...*

Fixed Code:

```
bSearch(A, v) {
    n = length(A)
    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)
}
```

Loop Invariant: If v is in A, then *v* must always be between A[l] and A[r] or equal to one of them (A[l] ≤ v ≤ A[r])

Initialization: *v* is a value in array A with bounds *l = 0* and *r = length(A)* ; so (A[l] ≤ v ≤ A[r])

Maintenance: During every recursion, the bounds reduce by half.
If *v* < *p*, then it is between bounds l = 0 and r = p-1
If *v* > *p*, then it is between bounds l = p+1 and r = length(A)
If *v* = *p*, then the algorithm terminates.

Termination: Either *v* is found in A, or the bounds *l* and *r* eventually become equal and determine *v* is not in A.

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

In this case, neither Crabbe or Goyle are correct. To find the time complexity for the binary search algorithm, one must consider how to get 1 by dividing N/2 a specific number of times $(1 = N / 2^x)$. This results in a time complexity of O(log(n)). For a four-nary search, the difference is a constant $(1 = 4 N / 2^x)$, resulting in a time complexity of

O(4log(n)). Because we are working with the RAM model of computation, this difference of a constant between the time complexities of the algorithms is negligible when it comes to pure efficiency. Therefore, neither Crabbe or Goyle are correct.

# Problem 5

**(10 pts extra-credit) 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.**

*the code has 2 for-loops based on size *n*

*so*... $\sum^{n}_{i=0}[\sum^{n}_{j=0}[\ \ldots\ldots]$

$\therefore$ time complexity 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.**

A1/B1 would be the best case if they both had the same number at their respective first index. *(A1 = [2, 4, 5, 6], B1 = [2, 3, 5, 8])*
A2/B2 would be the worst case if neither had a common element. *(A2 = [2, 4, 6, 8], B2 = [3, 5, 7, 9])*

**(c) Write pseudocode for an algorithm that runs in Θ(n) time for solving the problem. Your algorithm should use the fact that A and B are sorted arrays.**

```
def mergeFindCommon(A, B) {
    int C[]  //initialize C array same length as A and B
    length = length(C)

    while (i < length and j < length) {
        if (A[i] < B[j])
            C[k++] = A[i++];
        else
            C[k++] = B[j++];
    }

    for (int n = 0; n < length; n++) {
        if (C[n] == C[n+1])
            return True
    }
    return False
}
```

# Sources

**Websites:**

For Question 4b - https://stackoverflow.com/questions/8185079/how-to-calculate-binary-search-complexity

For Question 5c - https://www.geeksforgeeks.org/merge-two-sorted-arrays/

**People:**

Krish Dholakiya

George Allison

Selena Quintanilla

Eric Oropezealwood