Sam Ruditsky Due 9/20/17

Problem Set 1 COSI 21A – Antonella Di Lillo

Question 1

Prove that:

$\boldsymbol{(n+1)^5 \; is \; O(n^5)}$

$(n+1)^5 \leq cn^5$

Set n=1, $2^5 \leq \; c, \; c = 32$

For n_0=1 and c=32, $(n+1)^5 \; is \; O(n^5)$

$\boldsymbol{2n^4 - 3n^2 + 32n * \sqrt(n) - 5n + 60 \; is \; \Theta(n^4)}$

$c_1 n^4 \leq 2n^4 - 3n^2 + 32n * \sqrt{n} - 5n + 60 \leq c_2 n^4$

$\boldsymbol{O(n)}$: $2n^4 - 3n^2 + 32n * \sqrt{n} - 5n + 60 \leq cn^4$

$Set \; n = 1, \qquad 2 - 3 + 32 - 5 + 60 \leq c, \qquad 86 \leq c$

$For \; n_0 = 1 \; and \; c = 86, \qquad 2n^4 - 3n^2 + 32n * \sqrt(n) - 5n + 60 \; is \; O(n^4)$

$\boldsymbol{\Omega(n)}$: $2n^4 - 3n^2 + 32n * \sqrt{n} - 5n + 60 \geq cn^4$

$Set \; n = 1, \qquad 2 - 3 + 32 - 5 + 60 \geq c, \qquad 86 \geq c$

$For \; n_0 = 1 \; and \; c = 1, \qquad 2n^4 - 3n^2 + 32n * \sqrt(n) - 5n + 60 \; is \; \Omega(n^4)$

$Since \; 2n^4 - 3n^2 + 32n * \sqrt(n) - 5n + 60 \; is \; O(n^4) \; and,$

$2n^4 - 3n^2 + 32n * \sqrt(n) - 5n + 60 \; is \; \Omega(n^4)$

$2n^4 - 3n^2 + 32n * \sqrt(n) - 5n + 60 \; is \; \Theta(n^4)$

(I'm assuming we are still using log base 2)

$\boldsymbol{5n\sqrt{n} * \log(n) \; is \; O(n^2)}$

$5n\sqrt{n} * \log(n) \leq cn^2$

$Set \; n = 1, 0 \leq c, \;\; c = 1, 0 \leq 1$

For n_0=1 and c=1, $5n\sqrt{n} * \log(n) \; is \; O(n^2)$

$\boldsymbol{n^2 is\ \Omega(nlogn)}$

$n^2 \geq cnlog(n)$

Set n = 2, $4 \geq 2c,\ c = 1,\ 4 \geq 2$

For n_0*=2 and c=2,* $n^2 is\ \Omega(nlogn)$

Question 2

For each of the following program fragments give a big-O analysis of the running time:

Fragment 1 = $O(n)$

Fragment 2 = $O(n)$

Fragment 3 = $O(n^2)$

Fragment 4 = $O(n)$

Fragment 5 = $O(n^3)$

Fragment 6 = $O(n^2)$

Fragment 7 = $O(n^5)$

Fragment 8 $= O(logn)$

Fragment 9 = $O(n^4)$

<u>Question 3</u>

What do the following two algorithms do? Analyze its worst-case running time and express it using big-O notation:

Foo(a, n)

Input: two integers, a and n

Output: a^n

This algorithm returns the integer a to the exponent of n with running time $O(n)$.

Bar(a, n)

Input: two integers, a and n

Output: a^n

This algorithm returns the integer a to the exponent of n with running time $O(\text{logn})$.

Question 4

The input is an n by n matrix of numbers that is already in memory. Each individual row is increasing from left to right. Each individual column is increasing from top to bottom. Describe in pseudocode an O(n) worst-case algorithm that decides if a number X is in the matrix:

```
matrixSearch(A,n,X)
    Input: An n by n matrix A, with increasing rows and columns, already in memory and
    element X to find
    Output:  True if X is in the array and false if it is not

    // I start searching from the top right element, so A[n][n], and increment down or left by
    subtracting vertical and horizontal counts as needed
    verticalCount ← 0   //distance from top of matrix
    horizontalCount ← 0   //distance from right edge of matrix
    while horizontalCount < n and verticalCount < n do
        if A[n-horizontalCount][n-verticalCount] = X then
            return true
        else if A[n-horizontalCount][n-verticalCount] > X then
            horizontalCount ← horizontalCount + 1
        else if A[n-horizontalCount][n-verticalCount] < X then
            verticalCount ← verticalCount + 1
    return false
```

<u>Question 5</u>

Describe in pseudocode an efficient algorithm for finding the ten largest elements in an array of size n. What is the running time of your algorithm?

The running time of this algorithm is O(n).

```
tenLargest(A,n)
    Input: array A of size n
    Output: ordered array of the ten largest elements
    // I'm creating 10 variables to store the ten largest elements
    one ← A[1] //first element of the array
    two ← A[1]
    three ← A[1]
    four ← A[1]
    five ← A[1]
    six ← A[1]
    seven ← A[1]
    eight ← A[1]
    nine ← A[1]
    ten ← A[1]

    //This loop checks all the variables created above from one to ten, to see if the current
    element is larger than the stored integer. If it is, it reassigns the variable to that value and
    reassigns every variable smaller than it to the next largest integer, thus adding the new
    number and removing the smallest number out of the ten stored so far.
    for i ← 0 to n-1 do
        if x > one then
            ten ← nine
            nine ← eight
```

```
        eight ← seven
        seven ← six
        six ← five
        five ← four
        four ← three
        three ← two
        two ← one
        one ← x
    else if x > two then
        ten ← nine
        nine ← eight
        eight ← seven
        seven ← six
        six ← five
        five ← four
        four ← three
        three ← two
        two ← x
    else if x > three then
        ten ← nine
        nine ← eight
        eight ← seven
        seven ← six
        six ← five
        five ← four
        four ← three
        three ← x
```

```
    else if x > four then
        ten ← nine
        nine ← eight
        eight ← seven
        seven ← six
        six ← five
        five ← four
        four ← x
    else if x > five then
        ten ← nine
        nine ← eight
        eight ← seven
        seven ← six
        six ← five
        five ← x
    else if x > six then
        ten ← nine
        nine ← eight
        eight ← seven
        seven ← six
        six ← x
    else if x > seven then
        ten ← nine
        nine ← eight
        eight ← seven
        seven ← x
    else if x > eight then
```

```
                ten ← nine
                nine ← eight
                eight ← x
            else if x > nine then
                ten ← nine
                nine ← x
            else if x > ten then
                ten ← x
    tenLargest ← {one,two,three,for,five,six,seven,eight,nine,ten}
    return tenLargest
```

Question 6

An array A contains n - 1 unique integers in the range [0, n-1], that is, there is one number from the range that is not in A. Describe in pseudocode an O(n) time algorithm for finding that number. You are not allowed to use any extra array besides the array A itself.

findOddXOut(A,n)

    Input: Array of unique integers in the range [0, n-1] (I think it was actually supposed to be [0,n] right?), A of size n

    Output: Integer x in the range of [0, n-1] not in the array

```
    //rangeSum uses the equation for sum of all numbers from 1-n
    rangeSum ← (n*(n+1))/2
    for i ← 0 to n-1 do
        arraySum ← arraySum + A[i]
    x ← rangeSum – arraySum
    return  x
```