CP312: Algorithm Design & Analysis

Assignment #2

CP312, WLU, 2022

Instructor: Masoomeh Rudafshan

Due: Friday January 28th, 2022

Declan Hollingworth
190765210
holl5210@mylaurier.ca

Nishant Tewari
190684430
tewa4430@mylaurier.ca

Q1. The following algorithm finds the maximum element in the given array. Prove the correctness of the algorithm. In addition, analyze its time complexity in the worst-case and best-case scenarios.

```
def max(A)
    m = A[0]
    for i =  1 to n-1 do
        if A[i] > m then m = A[i]
    return(m)
```

Loop Invariant: At the start of the iteration with index i , … array A[0:n-1]

Initialization: Before the first iteration of the loop, m (the max element) is set to A[0] which is just the first element in the array. If there is only one element in the array then m[0] is the max element and still a valid return for m, therefore the initialization is valid.

Maintenance: The algorithm then loops through the array comparing m to the next item in the array. Suppose A[1] > A[0], the algorithm will then recognize this and replace m with the value at A[1]. This process is repeated for all elements in the array companing m to A[i], guaranteeing that once the loop reaches the end of the array, m with me the greatest value.

Termination: The loop terminates itself once it has gone through the entire array comparing m with A[i], then returns m. By the process shown in the maintenance step, the value of m is guaranteed to be the greatest value in array A and the algorithm successfully terminates.

Worst-case time complexity: O(n)
Regardless of the size of n or position of the max value, the loop will traverse through the entire array size n once. Given these conditions, worst case and best case performance will be O(n).

Best-case time complexity: O(n)
Regardless of the size of n or position of the max value, the loop will traverse through the entire array size n once. Given these conditions, worst case and best case performance will be O(n).

Q2. The following algorithm returns the product of two numbers, a and b. The parameters x and y are natural numbers. First, prove the correctness of the algorithm. Then, analyze the time complexity of the algorithm in the worst case scenario

```
function mult(a, b)
    if b = 0:
        return 0
    else if b is odd:
        return(mult(2a, b / 2) +a)
    else:
        return(mult(2a, b / 2))
```

Base Case:

let a $\in$ N and b = 0

When b is equal to 0 therefore it will return zero.

This means that mult(a, b) = 0 where b = 0 $\rightarrow$ mult(a ,0) = 0

Induction Hypothesis:

Assume mult(a,b) = a * b where, a $\in$ N & b $\in$ N

Inductive Step:

We need to prove mult(a, b+1) = a * (b+1) note* b = b+1

When b+1 is even $\rightarrow$ sub mult(2a, b/2) note* a = 2a & b = b+1

mult(a, b+1) = 2a * $\frac{b+1}{2}$ 2's cancel out

= a * (b+1)

When b+1 is odd$\rightarrow$ sub mult(2a, b/2)+ a note* a = 2a & b = b+1

mult(a, b+1) = 2a * $[\frac{b+1}{2}]$ + a $[\frac{b+1}{2}] = (\frac{b}{2})$ (floor)

= 2a($\frac{b}{2}$) +a 2's cancel out

= a * b + a

= a * (b + 1)

Q3. Solve the following recurrence using substitution method and then prove the correctness using induction:

$T(1) = 1 \qquad n \leq 2$

$T(n) = 2\,T(n - 1) + n - 1 \quad n > 2$

$T(1) = 1,\ n \leq 2 \qquad T(n) = 2T(n-1) + n-1,\ n > 2$

$T(n) = 2T(n-1) + n - 1$
$= 2\,[2T\,(n - 2) + n - 2] + n - 1$
$= 2^2\,T(n-2) + 2n - 4 + n - 1$

$T(n-1) = 2T((n - 1) - 1) + (n-1) - 1$
$= 2T(n - 2) + n - 2$

$T(n - 2) = 2T((n - 2) - 1) + (n - 2) - 1$
$= 2T\,(n - 3) + n - 3$

$= 2^2\,[2T(n - 3) + n - 3] + 2n - 4 + n - 1$
$= 2^3\,T(n - 3) + 4n - 12 + 2n - 4 + n - 1$

$= 2^3\,T\,(n - 3) + 2^2(n - 3) + 2^1(n - 2) + (n - 1)$

$$\therefore\ T(n) = 2^k\,T(n - k) + \sum_{i=0}^{k-1} 2^i\ *\ (n - (k + 1))$$

<u>Induction:</u> $T(1) = 1 \rightarrow T(n - k) = 1$

let $k = n - 1$

note* $(n - k) = 1$
$n = k + 1$
$n - 1 = k$

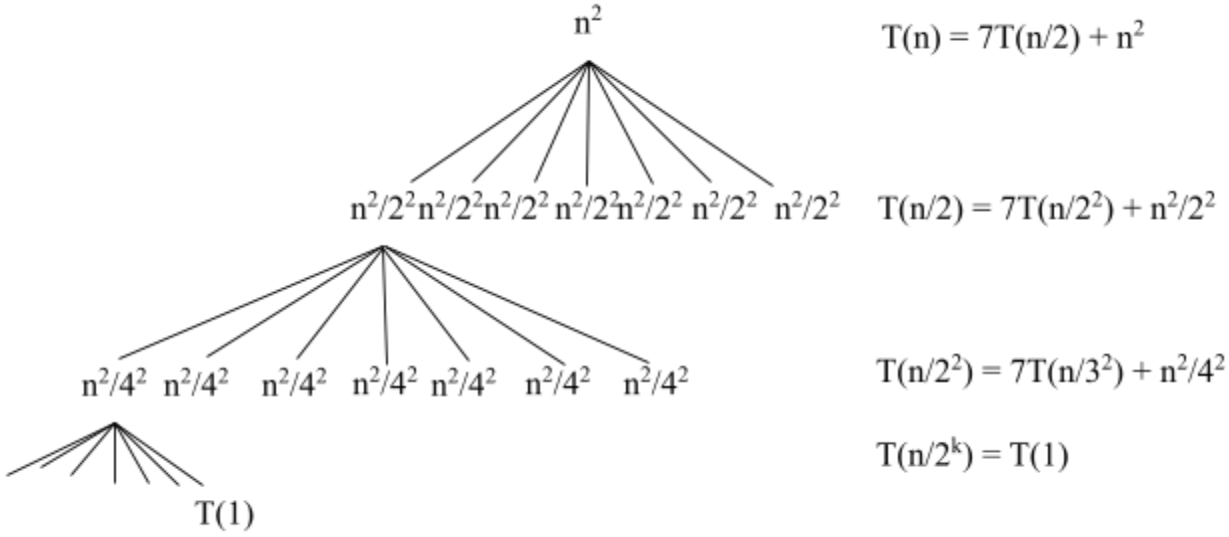
$$T(n) = 2^k\,T(n - k) + \sum_{i=0}^{k-1} 2^i\ *\ (n - (k + 1))$$
$$= 2^{n-1}\,T(n - (n - 1)) + \sum_{i=0}^{n-2} 2^i\ *\ (n - ((n-1) + 1))$$
$$= 2^{n-1}\,T(1) + \sum_{i=0}^{n-2} 2^i\ *\ 0$$
$$= 2^{n-1}$$

$\therefore$ $O(2^n)$ is the worst case time complexity!

Q4. Solve the following recurrence relation using recursion tree and master theorem:

$T(n) = c \qquad n \leq 2$
$T(n) = 7T(n/2) + n^2 \qquad n > 2$

Recursion Tree:



Master Method:

$T(n) = 7T(n/2) + n^2$

$a = 7, b = 2, f(n) = n^2$

$n^{\log_b a} = n^{\log_2 7} = n^{2.8}$

$f(n) = n^2 < n^{2.8}$

Given the conditions we now look at case 1: $T(n) = \Theta(n^{\log_b a})$

Therefore T(n) is as follows..

$T(n) = \Theta(n^{\log_2 7})$

$T(n) = \Theta(n^{2.8})$

$\therefore T(n) = \Theta(n^2)$

Q5. In the merge-sort algorithm we studied in the class, a problem is divided into two subproblems. Design and analyze a new version where each problem is divided into n subproblems. Write the pseudocode of the algorithm and analyze its time complexity.

Q6. For the polynomial multiplication algorithm we did in the class, write the pseudocode of the most efficient algorithm we discussed in class. In addition, prove the correctness of the algorithm?

Given two polynomial of degree n

$A(x) = a_0 + a_1 x + \ldots + a_n x^n$

$B(x) = b_0 + b_1 x + \ldots + b_m x^m$

Compute the product A(x)B(x)

$AB = A_1B_1 + (A_1B_2 + A_2B_1)\, x^{n/2} + A_2B_2\, x^n$

We need to compute:

$Q = (A_1+A_2)\,(B_1+B_2) = A_1B_1 + A_1B_2 + A_2B_1 + A_2B_2$

$Q = X + Y + Z + W$

$Y + Z = Q - X - W$

```
def betterPolyMult(A, B):
    n = len(A)
    If (n = 1): #(A*B)
        temp = [][]
        temp[0][0] = A[0][0] * B[0][0]
        temp[0][1] = A[0][1] * B[0][1]
        temp[1][0] = A[1][0] * B[0][0] + A[0][0] * B[1][0]
        temp[1][1] = A[1][1] * B[0][1] + A[0][1] * B[1][1]
    for i in range A[ :n/2] #(1st half of array A):
        A1 = [][]
        B1 = [][]
        A1[i][0] = A[i][0]
        A1[i][1] = A[i][1]
        B1[i][0] = B[i][0]
        B1[i][1] = B[i][1]
    for j in range A[n/2:] #(2nd half of array A):
        A2 = [][]
        B2 = [][]
        A2[i][0] = A[i][0]
        A2[i][1] = A[i][1]
        B2[i][0] = B[i][0]
        B2[i][1] = B[i][1]

    Q = betterPolyMult((A1+A2), (B1+B2))
    X = betterPolyMult(A1, B1)
    W = betterPolyMult(A2, B2)

    return (X + [Q - X - W])
```