CST 370 – Spring A 2020
Homework 3

Name: Sara Kazemi

Class ID: 4444

1. (5 points) Consider the following recursive algorithm.

```
Algorithm Q(n)
// Input: A positive integer n
if n = 1 return 1
else return Q(n - 1) + 2 * n - 1
```

(a) Present the return value for Q(1). **1**

(b) Present the return value for Q(2). **4**

(c) Present the return value for Q(3). **9**

(d) Determine what the algorithm computes for a general number n: Q(n).
The algorithm computes n^2.

2. (10 points) In the class, you learned the recurrence relation and backward substitution to get the time complexity of a recursive algorithm. To remind you of the topic, read the Google document again at https://goo.gl/HmoUNQ

   Solve the following recurrence relations using the backward substitution. You have to present **intermediate steps** as described.

   (a)

   M(n) = 2*M(n – 1) for n > 1 // recurrence relation
   M(1) = 3 // initial condition

   M(n) = 2*M(n-1) // replace M(n-1) with 2*M(n-2)
   = 2*[2*M(n-2)]
   = 2*2*M(n-2) // replace M(n-2) with 2*M(n-3)
   = 2*2*[2*M(n-3)]
   = 2*2*2*M*(n-3)
   …
   = 2^i * M(n-i) // for a general number i
   …
   = 2^{n-1}*M(1) // Initial condition M(1) = 3
   = 2^{n-1}*3 ∈ Θ(2^n)

   (b)

   M(n) = 3*M(n – 1) for n > 1 // recurrence relation
   M(1) = 4 // initial condition

   M(n) = 3*M(n-1) // replace M(n-1) with 3*M(n-2)
   = 3*[3*M(n-2)]
   = 3*3*M(n-2) // replace M(n-2) with 3*M(n-3)
   = 3*3*[3*M(n-3)]
   = 3*3*3*M*(n-3)
   …
   = 3^i * M(n-i) // for a general number i
   …
   = 3^{n-1}*M(1) // Initial condition M(1) = 4
   = 3^{n-1}*4 ∈ Θ(3^n)

3. (15 points) Consider the following recursive algorithm.

```
Algorithm Riddle(A[0..n − 1])
//Input: An array A[0..n − 1] of real numbers
if n = 1 return A[0]
else temp ← Riddle(A[0..n − 2])
     if temp ≤ A[n − 1] return temp
     else return A[n − 1]
```

(a) What does this algorithm compute?
This algorithm computes the value of the smallest element in the array.

(b) Set up a recurrence relation for the **comparison operation** at the **"if n = 1" in the line number 3**.
C(n) = C(n-1) +1, for n > 1

(c) Provide **an initial condition** for the recurrence relation you develop at the question (b)
C(1) = 1

(d) Solve the **recurrence relation** using backward substitution method. Present the intermediate steps and the time complexity of the result

C(n) = C(n-1) + 1 // replace C(n-1) with C(n-2) +1
= [C(n-2) +1] + 1
= C(n-2) +1 + 1 // replace C(n-2) with C(n-3) +1
= [C(n-3) +1] + 1 + 1
= C(n-3) +1 + 1 + 1
…
= C(n-i) + 1*i // This is for the general number i
…
= C(1) + 1*(n-1) // Initial condition is C(1) = 1
= 1 + 1*(n-1)
= 2*(n-1) ∈ Θ(n)

4. (15 points) Consider the following recursive algorithm for positive integer n

ALGORITHM S(n)
//Input: A positive integer n
if n = 1 return 1
else return S(n − 1) + n ∗ n ∗ n

(a) What does this algorithm compute? **The sum of the first n cubes of the positive integers**.
(b) Set up a recurrence relation for the **multiplication** as the **basic operation**
M(n) = M(n-1) + 2
(c) Provide **an initial condition** for the recurrence relation
M(1) = 0
(d) Solve the **recurrence relation** using backward substitution method. Present the intermediate steps and the time complexity of the results

M(n) = M(n-1) + 2 // replace M(n-1) with M(n-2) + 2
= [M(-2) + 2] + 2
= M(n-2) + 2 + 2 // replace M(n-2) with M(n-3) + 2
= [M(n-3) + 2] + 2 + 2
= M(n-3) + 2 + 2 + 2
…
= M(n-i) + 2*i // For a general number i, we will get this
…
= M(1) + 2*(n-1) // Initial condition M(1) = 0
= 0 + 2(n-1)
= 2*(n-1) ∈ Θ(n)

5. (15 points) Describe **an efficient algorithm** (in English) for finding all **common elements** in two **sorted lists** of numbers. For example, for the lists 2, 5, 5, 5 and 2, 2, 3, 5, 5, 7, the output should be 2, 5, 5. As another example, for the lists 20, 30, 50, 70, 90, 100 and 10, 20, 30, 50, 80, the output should be 20, 30, 50.
[Hint: You can do this task in the **linear time**.]

For two sorted lists, A and B, we can find the intersection of A and B using the following algorithm:

1. Using two index variables, *i* and *j*, to keep track of our position in arrays A and B, we initialize both *i* and *j* to 0.
2. While *i* is less than the length of A AND j is less than the length of B do the following:
3. If the elements A[*i*] and B[*j*] are equivalent, output element A[*i*] onto the terminal. Increment both index variables *i* and *j* by 1 and go back to Step 2.
4. Otherwise, if A[*i*] and B[*j*] are NOT equivalent:
- **If element A[*i*] is less than the element B[*j*], increment index variable *i* by 1 and go back to Step 2.**
- **If the element B[*j*] is less than the element A[*i*], increment index variable *j* by 1 and go back to Step 2.**

6. (20 points) Write a C++ program called **power.cpp** to compute 2^n for a nonnegative integer n. In this program, you have to develop **a recursive function** to calculate it. And also, you **can't use a library** in the program.
For the problem, you can assume that the user always enter a correct integer number which is greater than or equal to zero.

[Hint: Use the formula: $2^n = 2*2^{n-1}$ for the recursive function.]

Here are some sample test cases:

```
Test case 1
Enter a number: 0
Result: 1


Test case 2
Enter a number: 4
Result: 16

Test case 3
- We read your program and check if it uses a recursive function.
```

7. (20 points) Write a C++ program called **all_subsets.cpp** which displays all subsets of a set. In the problem, your program should read *n* characters from a user and display all subsets of the characters. In the program, you can assume that the number of input characters is less than or equal to 15. Your program should ask a user to enter the number of input characters. After that, it should read the characters. For the problem, you can assume that the input characters are always distinct.

```
Test case 1
Number of input characters: 1
Enter 1 characters: a
===== All Subsets =====
empty
{a}


Test case 2
Number of input characters: 2
Enter 2 characters: s t
===== All Subsets =====
empty
{s}
{t}
{s,t}


Test case 3
Number of input characters: 3
Enter 3 characters: c b a
===== All Subsets =====
empty
{a}
{b}
{c}
{a,b}
{b,c}
{a,c}
{a,b,c}
```

```
Test case 4
Number of input characters: 4
Enter 4 characters: v x y z
===== All Subsets =====
empty
{v}
{x}
{y}
…
{v,x,y,z}


Test case 5
Number of input characters: 10
Enter 10 characters: a b c d e f g h i j
===== All Subsets =====
empty
{a}
{b}
{c}
…
{a,b,c,d,e,f,g,h,i,j}
```