Faculty of Science

Course:	CSCI 3070U: Analysis and Design of Algorithms
Assignment:	#1
Topic:	Chapters 1-4

Note: This assignment is to be completed individually. This means you should not work in teams. These restrictions are in place because the idea of this assignment is to prepare you to do well on the upcoming midterm, which will include questions related to this assignment.

Part 1

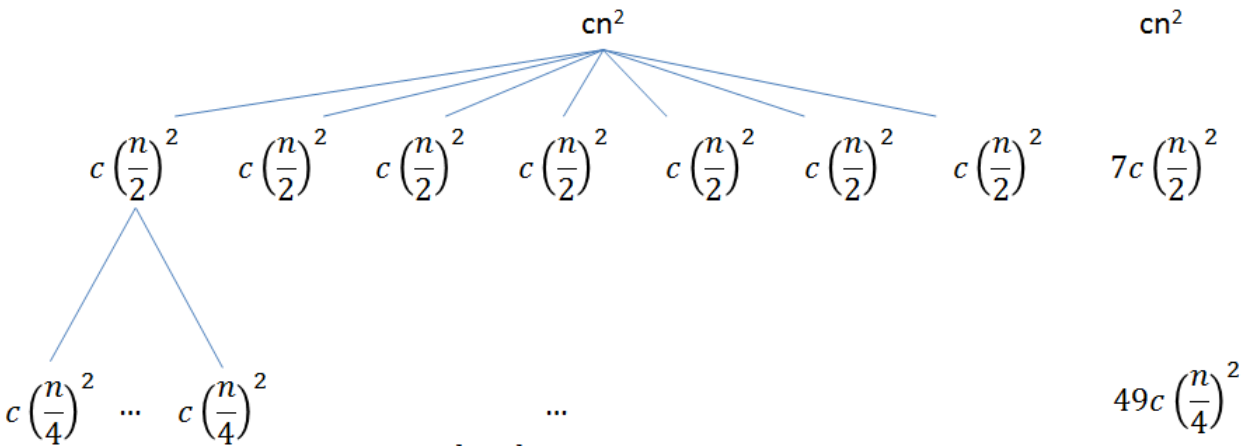
Find upper bounds, O(n) notation, for the following recurrences. Use the Equation editor in MS Word in order to ensure that your answers are readable, except for recursion tree where I'd recommend you create an image like in the week 2 slides. Show your work. Be sure to solve at least one of the problems using each of recursion tree, substitution, and master method.

a. $T(n) = 2T(n/2) + 3n + 7$

   By case 2 of the master method: $\Theta(n \lg n)$

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

   Using recursion tree:



$$h = \lg n$$

$$T(n) = \sum_{i=0}^{h} \frac{7^i}{4^i} n^2$$

Guess: $O(n^3)$

Note: You can verify that by the master method, this recurrence is bound by $n^{2.80}$

c. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Substitution guess: $O(n \lg n)$

Assumption: $T(k) \leq ck \lg k$, for all $k < n$

$$
\begin{aligned}
T(n) &= T(n/2) + T(n/4) + T(n/8) + n && \text{(recurrence)} \\
&= c(n/2)\lg(n/2) + c(n/4)\lg(n/4) + c(n/8)\lg(n/8) + n && \text{(assumption)} \\
&= (7/8)(cn \lg n) - 11cn/8 + n && \text{(algebraic manipulation)} \\
&\leq c\, n \lg n
\end{aligned}
$$

This should be obvious that 7/8 reduces the cn log n term, and the n term can be proven similarly:

$(8-11c)n/8$

For any $n \geq 1$, $c \geq 1$, this expression is negative.

d. $T(n) = 2T(n/4) + \sqrt{n}$

By case 2, $T(n) = \sqrt{n} \lg n$

Part 2

Rank the following functions by their order of growth. Be sure to put similar functions into the same category (f and g are in the same category iff $f(n) = \Theta(g(n))$). The result should be a table. The first column should be the category (and will span multiple answers if there is more than one function in that category). The second column should be the function itself. You do not need to justify your answers for this question.

Constant	None
Logarithmic	$\log_{10} n$ $3 \log_2 n$
Linear	$2^{\log_2 n}$ $18n$
Polynomial	$\sqrt{n^3}$ n^2 $3n^2 + 7n + 15$ $n^3 - \log n$ n^3 $¾n^4$
Exponential	2^n 4^n $n^{71} + 5^n + 17n$
Factorial	$n!$

Part 3

Using divide and conquer, create an algorithm to solve the problem that follows. You can use C, C++, Java, or Python to solve this problem (ask the instructor if you have another programming language in mind). Include the asymptotic upper bound for your algorithm in your response (including the cost of subdivision and combining the results).

> *You have a long string containing many characters (such as this paragraph), and you want to search for a substring within this string. For example, one may want to search for "characters" or "want to" or "bstring wi" or "language". All but the last example should be found.*

Keep in mind that if you use divide and conquer to solve this problem there is one complication. The string to be found could be split between two of the sub-problems (assuming your algorithm divides the string into two smaller strings). You'll need to handle that case as well.

Basic logic:
-The main function takes a boolean arg (startOfSubstring) returns part of the substring that was found (prefix or suffix)
- -If `start ==true`, we search for part of the substring ending at the end of the string
- -If `start ==false`, we search for part of the substring beginning at the start of the string

-Search for the substring in the first half of the string, recursively (`start ==true`)
- -If this recursive call returns the entire substring, we have found it
- -If this recursive call returns some part of the substring, then store that as s1

-Search for the substring in the second half of the string, recursively (`start ==false`)
- -If this recursive call returns the entire substring, we have found it
- -If this recursive call returns some part of the substring, store it as s1

-If neither recursive call returned anything, the substring is not in the string
-If s1 + s2 == the substring, then we have found it (across the barrier)

The following is one example use of divide and conquer to solve this problem:

```
def searcher(start,a,x):
    #start - look at the end of the string to see if it matches target start
    #!start - look at the start of the string to see if it matches target end
    #a - string to search
    #x - target
    if len(x)==1:
        #just do linear search
        for i in a:
            if i == x:
                return x
        return ''

    s1=""
    s2=""
    if len(a)>=len(x):
        #don't bother splitting the string if it's smaller than x
        s1 = searcher(True, a[:len(a)/2], x)
        s2 = searcher(False, a[len(a)/2:], x)
        if s1+s2 == x or s1 == x or s2 == x:
            return x

    if start:
        target = x[0]
        for i in range(len(a)-1,max(len(a)-len(x),-1),-1):
            #see if end of string has the start of x
            if a[i]==x[0]:
                if x.startswith(a[i:]):
                    return a[i:]
    else:
        target = x[-1]
        for i in range(0,min(len(a), len(x))):
            #see if start of string has the end of x
            if a[i]==x[-1]:
                if x.endswith(a[:i+1]):
                    return a[:i+1]
    return ''
```

Another approach:

```
def IsItSplited(s,left,right):

    splited = False
    for i in range(max(0, len(s)-len(right)), min(len(s),len(left))+1):
        if left[max(len(left)-i,0):len(left)] + right[0:len(s)-i] == s :
            splited = True
    return splited


def Find(s,t):


    left = t[0:len(t)/2]
    right = t[len(t)/2:]

    l = r = False

    if IsItSplited(s,left,right):
        return True

    if len(left) > len(s):
        l = Find(s,left)       # what we found in the left
        if l:
            return True

    if len(right) > len(s):
        r = Find(s,right)     # what we found in the left
        if r:
            return True

    return False
```

Part 4

Java-based solutions for this part can be found in the file `Heap.java`.