CSI 3105 Computer Science Spring/Summer 2019 University of Ottawa

Assignment #2 (60 points, weight 5%)
Due: Monday June 17, 11:30 PM

The assignment is to be uploaded on Brightspace electronically (you may type or write by hand legibly and scan it). Only a single PDF file is accepted.
In your assignment, please be sure to explain all of your solutions CLEARLY, or you will lose marks. To be complete, in any worst case analysis you do you must state the input size, the basic operations being counted, and a general input of your fixed input size which gives the worst case (you should state these things in your solution even if some of them are given in the questions). Also, you will be marked on the efficiency of any algorithms you design.

Late submission is accepted from 1 min late up to 24 hs for 30% off (i.e. your mark is multiplied by 0.7).

This assignment will be marked out of 60.

1. [10 marks]
Consider the following idea for searching for a number x in a list S of n numbers in sorted order, where n is an exact multiple of 4: Look at every fourth number, comparing it to x. If you find x, stop. If you encounter a number bigger than x, then go backwards one number at a time in S at most three times, looking for x. Again, if you find x, stop.

Here is the pseudocode for this algorithm:

Inputs: positive integer n which is a multiple of 4, sorted array S of n keys indexed from1 to n, a key x.
Outputs: position, the location of x in S (0 of x is not in S).

```
void FourthSearch( int n, const keytype S[], keytype x, index& position)
{
index i, last;
boolean done;
position = 0;
done = false;
i = 4;
while ((position = = 0) && (i<=n) && (done = = false )) {
```

```
        if (x = =S[i])
                position = i;
        else if ( x > S[i] )
                i= i + 4;
          else { //go backwards
            last = i-3;
            while (not done) {
                i = i-1
                if (S[i] = = x) {
                        position = i;
                        done = true;
                        }
                else if ((x > S[i]) or (i = = last))
                        done = true;
                } //endwhile
            } //endelse
      } //endwhile
   }
```

Do an average case analysis of this algorithm, assuming that n = 4, and that all input types are equally likely and all keys in S are distinct. Count, as your basic operation, comparisons of x with keys in S (and count ALL such comparisons). You should have all inputs partitioned into 9 types if you are doing this analysis correctly. You must explain your input partitions clearly.

2. [14 marks]

Consider the following algorithm based on the divide and conquer strategy which, given three positive integers x, n and p where n is a power of 2, n>=2 (i.e. that $n = 2^k$ for k >=1), finds the remainder when x^n is divided by p.

Inputs: Three positive integers x, n and p, where $n = 2^k$ for some k>=1.

Output: The remainder when x^n is divided by p.

```
int findRem(int x, int n, int p)
{
        int r;
        if (n = =2)
                return (x*x) mod p;
        else {
                r = findRem(x, n/2, p);
                return (r*r) mod p;
                }
   }
```

Do a worst-case analysis for the above algorithm. Of course, assume that n is a power of 2, i.e. $n=2^k$ for some k>=1. Do the following as part of your analysis:

a) [4 marks]
Write a recursive formula for the worst-case analysis of this algorithm (don't forget your base case in this formula). As your basic operation counted use divisions, multiplications and number of mod function calls

b) [4 marks]
Through back substitution, solve the recurrence relation from a). Please show your work, and follow the method as shown in class.

c) [5 marks]
Verify your solution found in b) using induction.

d) [1 marks]
What is the complexity of this algorithm? You just need to state your answer here

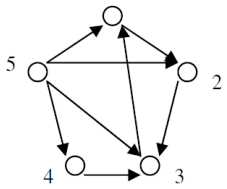
3. [10 marks]
Suppose you have an algorithm which has the following recurrence relation for W(n), assuming n is a power of 2, i.e. assuming $n=2^k$, $k \geq 0$:

$W(n) = 2W(n/2) + 2n + 2$ for $n > 1$
$W(n) = 1$ for $n = 1$

Using back substitution and assuming n is a power of 2, i.e. $n = 2^k$, find an exact (i.e. non-recursive) formula for W(n). Be sure to show your work, and to simplify your final answer as much as possible. Note that you do **NOT** need to verify your final answer using induction.

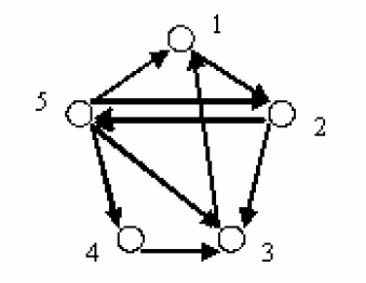
4. [10 marks total]
Suppose you are given a directed graph G with *n* nodes, stored in an *n x n* adjacency matrix W (i.e. W[i,j] is 1 if there is an arc that goes from vertex *i* to vertex *j*, and is 0 otherwise). Note that it is possible to have an arc from node *i* to node *j*, and from node *j* to node *i*. A "start node" in G is a node v such that there is an arc from v to *i* to all the other nodes *i* in G, and no arcs directed into v. We wish to know if G has a start node or not, and to find that start node if it exists.

Example: The directed graph below has a start node, namely node 5.



$$W = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Here is another directed graph below, which has no start node:



a) [7 marks]
Design a divide and conquer algorithm for this problem. You must first describe the idea behind your algorithm clearly in English, and then write the pseudocode for your algorithm. You will be marked on the clarity and the efficiency (in terms of complexity class) of your algorithm.
(Hint: Try to design your algorithm such that, at each stage of the recursion, you get to eliminate one node from consideration).

b) [3 marks]
Do a worst case analysis of your algorithm using the input size n, and counting the number of "looks" your algorithm makes to elements in W. In your analysis, find a recurrence relation for your worst case W(n), and solve this recurrence relation using back-substitution. You do NOT need to verify your back substitution guess using induction in this case.

5. [16 marks total]

Suppose you have an array S indexed from 1 to n which contains n numbers, not in any particular order, and you wish to count how many times a given number x occurs in S. Consider the recursive algorithm below for this which finds the number of occurrences of x in the index range i...j in S. Of course, solving the problem would involve an initial call to the algorithm for the range 1..n:

```
int CountOccur (int i,j)
{
int mid, count;
if j < i
        return 0;
else {
        mid =Floor((j + i)/2)
        if S[mid] = x
                Count = 1 + CountOccur(i,mid-1) + CountOccur(mid+1,j);
        else
                Count = CountOccur(i,mid-1) + CountOccur(mid+1,j);
        return Countt;
        }
   }
```

a) [12 marks]
Design a dynamic programming version of the above recursive algorithm and write the pseudocode for your algorithm. For this algorithm, you should have a table which has an entry for every index pair i and j, 1 i n, 1 j n. For index pair i and j your table should contain the number of times x appears in S in the range i...j. And of course, you must allow for the fact that you will need entries for j<i sometimes too (so you may need to enlarge your table for this—think about this).

b) [3 marks]
Illustrate your algorithm on the following example:
S = [2, 3, 6, 2, 7, 8, 2, 9], x =2.

c) [1 marks]
State the complexity of your algorithm, where you are counting the number of table entries calculated.

You are done.