CS 2223 D-Term 2020 SAMPLE MIDTERM EXAM Name:

Friday, April 17, 2019

Question 1 *(??? points)*
First Order Linear Recurrences

Solve the following recurrence relation using either the method of forward substitution or the method of backward substitution:

$$\begin{array}{lll}
W(n) = W(n-1) + 1, & \text{for } n > 1; & W(1) = 1 \\
W(n) = W(n-1) + n, & \text{for } n > 1; & W(1) = 1 \\
W(n) = 2W(n-1) + 1, & \text{for } n > 1; & W(1) = 1 \\
W(n) = 2W(n-1) + n, & \text{for } n > 1; & W(1) = 1 \\
W(n) = W(n/2) + n, & \text{for } n > 1; & W(1) = 1, \text{ solve for } n = 2^k
\end{array}$$

Solutions (by forward substitution):

$W(n) = W(n-1) + 1$, for $n > 1$; $W(1) = 1$
$W(1) = 1$
$W(2) = 1 + 1 = 2$
$W(3) = 2 + 1 = 3$
$W(4) = 3 + 1 = 4$
The pattern suggests $W(n) = n$

$W(n) = W(n-1) + n$, for $n > 1$; $W(1) = 1$
$W(1) = 1$
$W(2) = 1 + 2 = 3$
$W(3) = 3 + 3 = 6$
$W(4) = 6 + 4 = 10$
$W(5) = 10 + 5 = 15$
The pattern suggests $W(n) = \frac{n(n+1)}{2}$.

$W(n) = 2W(n-1) + 1$, for $n > 1$; $W(1) = 1$
$W(1) = 1$
$W(2) = 2 \cdot 1 + 1 = 3$
$W(3) = 2 \cdot 3 + 1 = 7$
$W(4) = 2 \cdot 7 + 1 = 15$
$W(5) = 2 \cdot 15 + 1 = 31$
The pattern suggests $W(n) = 2^n - 1$. And the monks keep on keeping on.

$W(n) = 2W(n-1) + n$,for $n > 1$;$W(1) = 1$
This is delightful but just a little too tough for an exam. This exam anyway.

This time by backward substitution:

$W(n) = W(n/2) + n$,for $n > 1$;$W(1) = 1$, solve for $n = 2^k$
$W(1) = 1$
$W(2) = 1 + 2 = 3$
$W(4) = 3 + 4 = 7$
$W(8) = 7 + 8 = 15$
$W(16) = 2 \cdot 15 + 16 = 31$

The pattern suggests $W(2^k) = 2^{k+1} - 1$, but we need to report this in terms of n, not k:

$W(2^k) = 2^{k+1} - 1$
$W(n) = 2 \cdot 2^k - 1$
$W(n) = 2n - 1.$

Question 2 *(??? points)*
Big O

QuickSort and MergeSort are both $O(n^2)$. However, of the two, only MergeSort is $O(n \lg n)$.

Explain why this is so.

The number of comparisons made by MergeSort is independent of the data being compared. Contrariwise, the number of comparisons made by QuickSort does depend on the data being compared through the choice of pivot values.

MergeSort always makes $O(n)$ comparisons on each of the $O(\lg n)$ passes needed to feather single-element lists into a sorted sequence of n elements. Hence MergeSort has running time $M(n) = O(n \log n)$, and any function which is $O(n \log n)$ is also $O(n^2)$.

Under best-case and average-case circumstances, QuickSort behaves similarly to MergeSort, but in the worst-case, i.e. with choices of pivots that create empty sublists on each pass, it makes the same number of comparisons as SelectionSort which has running time $O(n^2)$.

So, while QuickSort almost always runs in $O(n \log n)$ (good implementations are usually faster even than MergeSort), its actual running time is that of SelectionSort, namely $Q(n) = O(n^2)$.

Question 3 *(??? points)*
Master Theorem

Use the Master Theorem to find the running time of a Divide-and-Conquer algorithm that:

a. Divides a problem into *four* subproblems of *one quarter* size in constant time.

b. Divides a problem into *two* subproblems of *one half* size in linear time.

c. Divides a problem into *two* subproblems of *one half* size in quadratic time.

The Master Theorem says that if the recurrence describing our algorithm is $R(n) = aR(\frac{n}{b}) + \Theta(n^d)$, then the running time of our algorithm, $T(n)$, is:

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \quad \text{housekeeping / few leaves,} \\ \Theta(n^d \log n) & \text{if } a = b^d \quad \text{``balance'',} \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \quad \text{(lots of leaves).} \end{cases}$$

So, for a) above: $a = 4, b = 4$, and $d = 0$. Thus, $4 > 1$ and $a > b^d$, so the bottom line obtains, and $T(n) = \Theta(n^{\log_4 4} = n^1 = n)$, or linear time.

And for b) above: $a = 2, b = 2$, and $d = 1$. Thus, $2 = 2$ and $a = b^d$, so the middle line obtains, and $T(n) = \Theta(n^1 \log n = n \log n)$, or linearithmic time.

And for c) above: $a = 2, b = 2$, and $d = 2$. Thus, $2 < 4$ and $a < b^d$, so the top line obtains, and $T(n) = \Theta(n^2)$, or quadratic time.

Question 4 *(??? points)* Graph Representations

Construct (visually) the graph that corresponds to the given matrix:

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 1 \\ 5 & 1 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}$$

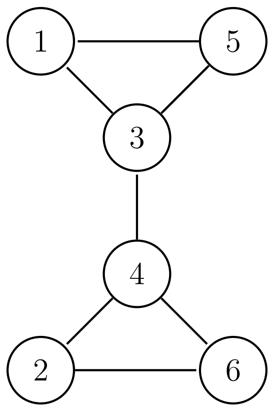
Also provide an adjacency list for the graph.

Adjacency list: {{1,3},{1,5},{2,4},{2,6},{3,4},{3,5},{4,6}}

Is this a graph or a digraph? How can you tell without drawing it?

This is a graph, rather than a digraph (directed graph), and that can be seen visually in that it is symmetric about the main diagonal. That is, for every (x, y) entry present in the matrix, entry (y, x) is also present in the matrix. It is for this reason that the adjacency list above has just seven entries (even though the matrix has 14) and the reason that the edges appear as sets, that is with braces: {1, 3} rather than as ordered pairs with parentheses: (1,3) as they would for a digraph.

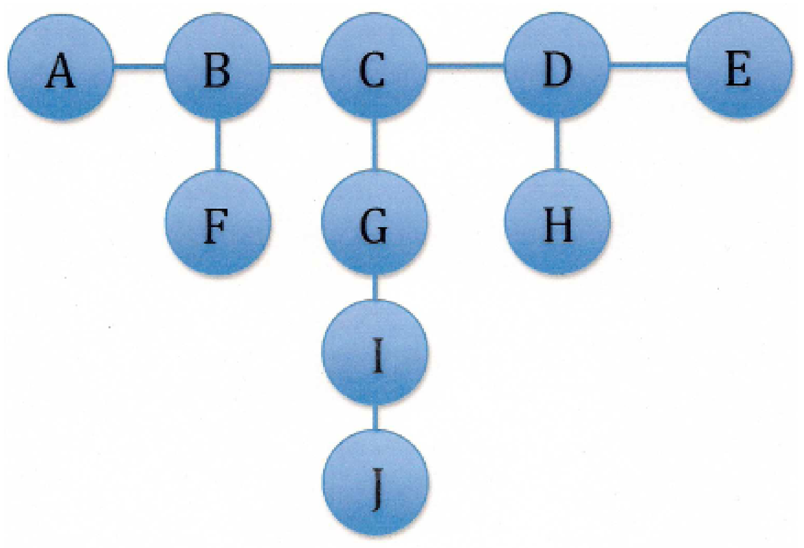
Here's a representation of the graph:



Question 5 *(??? points)*

Perform a traversal of the graph below by executing a Breadth-First search algorithm, starting at vertex A.
Report the vertices in the order they are discovered.

Perform a traversal of the graph below by executing a Depth-First search algorithm, starting at vertex A.
Report the vertices in the order they are discovered, i. e. "push" order.
Also, report the vertices in the order they are discarded, i. e. "pop" order.



BFS order: A,B,C,F,D,G,E,H,I,J

Also: A,B,F,C,G,D,I,H,E,J

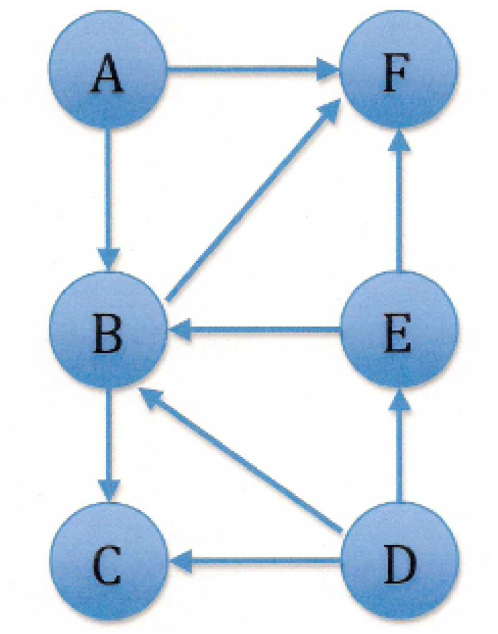
DFS push order of traversal: A,B,C,D,E,H,G,I,J,F

Also: A,B,C,G,I,J,D,H,E,F

DFS pop order of reporting: E,H,D,J,I,G,C,F,B,A

Also: J,I,G,H,E,D,C,F,B,A

Question 6 *(??? points)*
Perform a Topological Sort of the graph below. You may use the Source Removal Algorithm or the Depth-First Search algorithm.



Here are two solutions:

A, D, E, B, F, C

D, E, A, B, C, F

There are more.

Question 7 *(??? points)*
MergeSort or Quicksort

Trace the execution of the (ascending order) MergeSort algorithm on the following sequence:

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 5&8&7&4&6&1&2&3\\\hline\end{array}$$

Here's the top-down version.

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 5&8&7&4&6&1&2&3\\\hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline 5&8&7&4\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 6&1&2&3\\\hline\end{array}$$

$$\begin{array}{|c|c|}\hline 5&8\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 7&4\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 6&1\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 2&3\\\hline\end{array}$$

$$\boxed{5}\quad\boxed{8}\quad\boxed{7}\quad\boxed{4}\quad\boxed{6}\quad\boxed{1}\quad\boxed{2}\quad\boxed{3}$$

$$\begin{array}{|c|c|}\hline 5&8\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 4&7\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 1&6\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 2&3\\\hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline 4&5&7&8\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 1&2&3&6\\\hline\end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 1&2&3&4&5&6&7&8\\\hline\end{array}$$

The bottom up version looks like this:

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 1&2&3&4&5&6&7&8\\\hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline 4&5&7&8\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 1&2&3&6\\\hline\end{array}$$

$$\begin{array}{|c|c|}\hline 5&8\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 4&7\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 1&6\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 2&3\\\hline\end{array}$$

$$\boxed{5}\quad\boxed{8}\quad\boxed{7}\quad\boxed{4}\quad\boxed{6}\quad\boxed{1}\quad\boxed{2}\quad\boxed{3}$$

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 5&8&7&4&6&1&2&3\\\hline\end{array}$$

Question 8 *(??? points)*
Heaps

Construct a heap from the numbers below

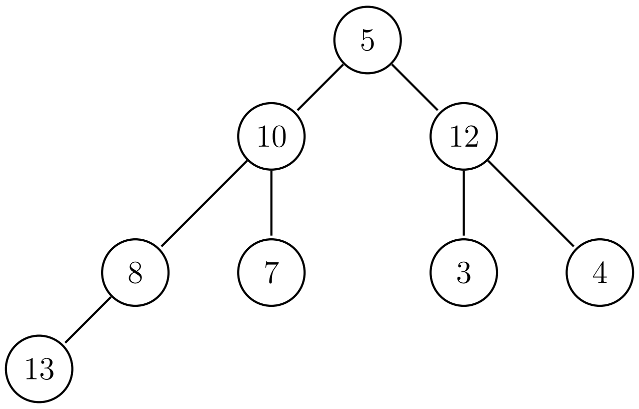
a) Using the bottom-up method, and

b) Using the top-down method.

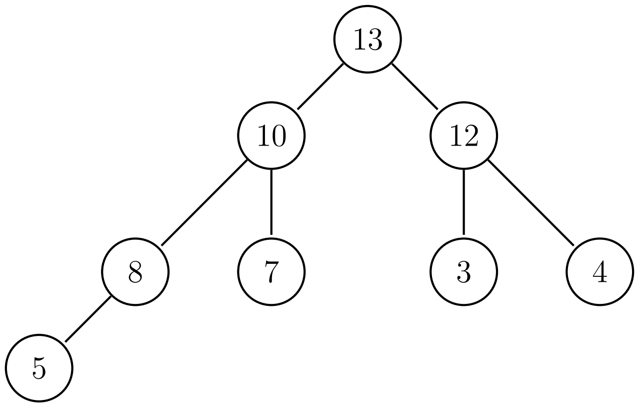
(Solutions will assume a left-to-right sequence of insertions. Will right-to-left result in a different heap?)

5,10,12,8,7,3,4,13

Using the Bottom-Up method, we seed a complete binary tree with the numbers in the order presented.



Taking the parent nodes in order from right-to-left and bottom-to-top (a relic of the old array representation techniques, perhaps), we have the 13 swap with 8, the 12 stays, the 13 swaps with the 10, the 13 swaps with the 5, the 5 with the 10 and finally with the 8, to yield:



Using the top-down method to build a heap means inserting the numbers one at a time and then swapping where necessary to preserve the heap property. (The shape property is preserved by default in the array representation.

5,10,12,8,7,3,4,13

We'll insert them one-at-time into an array of appropriate length (with a sentinel at the head of the array).

0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
1	5							

0	1	2	3	4	5	6	7	8
2	10	5						

0	1	2	3	4	5	6	7	8
3	12	5	10					

0	1	2	3	4	5	6	7	8
4	12	8	10	5				

0	1	2	3	4	5	6	7	8
5	12	8	10	5	7			

0	1	2	3	4	5	6	7	8
6	12	8	10	5	7	3		

0	1	2	3	4	5	6	7	8
7	12	8	10	5	7	3	4	

0	1	2	3	4	5	6	7	8
8	13	12	10	8	7	3	4	5

Is this the same heap the the Bottom-Up process produced?

Question 9 *(??? points)*
Horner's Rule

Evaluate $f(x) = 3x^5 + 4x^4 - 3x^2 + 2x - 1$ at $x = 2$ using Horner's Rule.

Note: You *must* show your work to receive credit, but you should also evaluate the polynomial in the traditional way in order to check your work.

Note also: There's a term missing. Don't forget to incorporate that into your evaluation. (Call it a "trick question", if you must.)

Note finally: Answers will be provided for $f(0), f(1), f(-1), f(-2)$, and $f(5)$ as well, so further practice will be rewarded. Of course, you can invent your own polynomial and test yourself.

$f(x) = 3x^5+4x^4-3x^2+2x-1$ can be written as $f(x) = x(x(x(x(3x+4)+0)-3)+2)-1$

$f(2) = 2(2(2(2(3 \cdot 2 + 4) + 0) - 3) + 2) - 1$
$f(2) = 2(2(2(2(6 + 4) + 0) - 3) + 2) - 1$
$f(2) = 2(2(2(2(10) + 0) - 3) + 2) - 1$
$f(2) = 2(2(2(20 + 0) - 3) + 2) - 1$
$f(2) = 2(2(2(20) - 3) + 2) - 1$
$f(2) = 2(2(40 - 3) + 2) - 1$
$f(2) = 2(2(37) + 2) - 1$
$f(2) = 2(74 + 2) - 1$
$f(2) = 2(76) - 1$
$f(2) = 152 - 1$
$f(2) = 151$

Traditionally:
$f(2) = 3 \cdot 2^5 + 4 \cdot 2^4 - 3 \cdot 2^2 + 2 \cdot 2 - 1$
$f(2) = 3 \cdot 32 + 4 \cdot 16 - 3 \cdot 4 + 2 \cdot 2 - 1$
$f(2) = 96 + 64 - 12 + 4 - 1$
$f(2) = 151$, as expected.

Here's a shorty cut suggested by our text:

Coefficients:	3	4	0	-3	2	-1
add coeff		10	20	37	76	**151**
"times $x = 2$"	6	20	40	74	152	

Evaluate $f(x) = 3x^5 + 4x^4 - 3x^2 + 2x - 1$ at some other values using the shorty cut Horner's Method:

$f(0) =$

Coefficients:	3	4	0	-3	2	-1
add coeff		4	0	-3	2	**-1**
"times $x = 0$"	0	0	0	0	0	

$f(1) =$

Coefficients:	3	4	0	-3	2	-1
add coeff		7	7	4	6	**5**
"times $x = 1$"	3	7	7	4	6	

$f(-1) =$

Coefficients:	3	4	0	-3	2	-1
add coeff		1	-1	-2	4	**-5**
"times $x = -1$"	-3	-1	1	2	-4	

$f(-2) =$

Coefficients:	3	4	0	-3	2	-1
add coeff		-2	4	-11	24	**-49**
"times $x = -2$"	-6	4	-8	22	-48	

$f(5) =$

Coefficients:	3	4	0	-3	2	-1
add coeff		19	95	472	2362	**11809**
"times $x = 5$"	15	95	475	2360	11810	

Question 10 *(??? points)*
Use a dynamic programming algorithm to solve the coin row problem (maximize the sum without taking adjacent coins) for the following arrangement of coins:

3	2	1	6	4	7	8	5

We examine the Value, that is the sum of the cells we choose.
We define $V(0) = 0$ and $V(1) = c_1$, the value of the first cell.

Our algorithm says $V(n) = \max \begin{cases} V(n-1) \\ V(n-2) + c_n \end{cases}$, that is, we keep just what we have from the most recent step OR we go back two steps and add the current cell (coin).

Coins	:	3	2	1	6	4	7	8	5		
V(0)	=									=	0
V(1)	=	3								=	3
V(2)	=	3	-							=	3
V(3)	=	3	-	1						=	4
V(4)	=	3	-	-	6					=	9
V(5)	=	3	-	-	6	-				=	9
V(6)	=	3	-	-	6	-	7			=	16
V(7)	=	3	-	-	6	-	-	8		=	17
V(8)	=	3	-	-	6	-	7	-	5	=	21

Question 11 *(??? points)*
Read Pseudocode

Describe what the following algorithm does:

```
ALGORITHM DoppelGänger[1]($A[0..n-1]$)
//??????????
//Input: (possibly empty) array $A$ of arbitrary type
//Output: Boolean

tempFlag ← False
$i \leftarrow 1$
while $n > 1$ and $i \leq= n - 1$
        if $A[i-1] = A[i]$
            tempFlag ← True
        $i \leftarrow i + 1$
return tempFlag
```

This algorithm returns a Boolean value; **True** if and only if there exists a pair of adjacent cells with identical entries. (And, of course, **False** otherwise.)

Note: Empty arrays and arrays with a single entry return **False**, essentially by default. They never enter the loop where tempFlag might be set to **True**.

Also, there's a silly inefficiency in the algorithm. Do you see it? Can you fix it easily?

[1] *Doppelgänger* is German for "double goer". Think "twin".

Question 12 *(??? points)*
Write Pseudocode

Write pseudocode for an algorithm that examines array A of length n and returns "True" if it is sorted in ascending order and returns "False" otherwise. An array of length 0 should be considered sorted; you may NOT assume the array contains no duplicates.

```
**ALGORITHM** $isAscending(A[0..n-1])$
//Determines whether an array is sorted in ascending order
//Input: (possibly empty) array $A$ of arbitrary type
//Output: Boolean

tempFlag $\leftarrow$ True
$i \leftarrow 1$
**while** $n > 1$ **and** $i \leq= n - 1$ **and** tempFlag
    **if** $A[i-1] > A[i]$
        tempFlag $\leftarrow$ False
    $i \leftarrow i + 1$
**return** tempFlag
```

This algorithm looks quite a bit like *DoppelGänger*, but it does *not* suffer from the same inefficiency. Can you see the difference?

Question 13 *(??? points)*
Sums

$\sum_{k=0}^{n} 2^k =$ a) $2^{n-1}-1$ b) $2^{n-1}+1$ **c) $\mathbf{2^{n+1}}$-1** d) $2^{n+1}+1$

$\sum_{k=1}^{n} k =$ a) n^2 b) $\frac{n(n-1)}{2}$ **c) $\mathbf{\frac{n(n+1)}{2}}$** d) $\frac{(n-1)(n+1)}{2}$

$\sum_{k=1}^{n} k^2 =$ a) n^3 **b) $\mathbf{\frac{n(n+1)(2n+1)}{6}}$** c) $\frac{n(n+1)(n+2)}{6}$ d) $\frac{(n+1)(n+2)(n+3)}{6}$

This is simply multiple choice. Correct answers are shown in **bold**.

If these are not immediately recognizable, simply evaluate the sum for the first value of its index/limit. (In this case it's $n = 0$ or $n = 1$.) Then evaluate the expressions listed. Most will be eliminated immediately. If more than one survives, evaluate the sum for another convenient value, perhaps, $n = 1$ or $n = 2$, until only one choice remains.

Of course, you *could* then try to prove these by induction.

Question 14 *(??? points)*
Use the Euclidean algorithm to find the greatest common divisor of 210 and 308.

$308 = 1 \times 210 + 98$

$210 = 2 \times 98 + 14$

$98 = 7 \times 14 + 0$

Since 14 is the last non-zero remainder, 14 is the greatest common divisor of 210 and 308.

Also:
$308 = 2 \times 2 \times 7 \times 11 = 2^2 \cdot 7 \cdot 11$
$210 = 2 \times 3 \times 5 \times 7 = 2 \cdot 3 \cdot 5 \cdot 7$
They share a single 2 and a 7, hence $2 \times 7 = 14$ is the gcd by the factorization method, as well. (As it had better be.)