CSC 225 SPRING 2021 A01 & A02 (CRN 20713 & 20714)
ALGORITHMS AND DATA STRUCTURES I
FINAL EXAMINATION
UNIVERSITY OF VICTORIA

Student ID: _____

Name: _____

DATE: 17 APRIL 2021
DURATION: 3 HOURS
INSTRUCTOR: RICH LITTLE

THIS QUESTION PAPER HAS **NINE** PAGES INCLUDING THE COVER PAGE.

THIS QUESTION PAPER HAS **EIGHT** QUESTIONS.

ALL ANSWERS CAN BE WRITTEN ON THIS EXAMINATION PAPER, ON YOUR OWN PAPER OR TYPESET, PROVIDED YOU SUBMIT ONE PDF DOCUMENT CONTAINING THEM.

THIS IS AN OPEN BOOK EXAM. YOU MAY REFERENCE ANY COURSE MATERIALS; SLIDES, TEXTBOOKS, CONNEX.

NO OTHER ACCESS TO THE INTERNET IS ALLOWED, WITH THE POSSIBLE EXCEPTION OF AN ONLINE PDF APPLICATION.

NO COMMUNICATION/COLLABORATION ALLOWED WITH ANY OTHER PERSON.

YOUR SOLUTIONS MUST BE SUBMITTED IN THE ASSIGNMENTS TOOL IN CONNEX BY 12 AM VICTORIA TIME. EXAMS SUBMITTED IN ANY OTHER WAY WILL NOT BE ACCEPTED UNDER ANY CIRCUMSTANCES. LEAVE YOURSELF PLENTY OF TIME TO DO THE SUBMISSION PROCESS.

Q1 (8)	
Q2 (8)	
Q3 (8)	
Q4 (8)	
Q5 (8)	
Q6 (8)	
Q7 (8)	
Q8 (8)	
TOTAL (64) =	

1. (a). [3 marks] In terms of n, what is the total running time, $T(n)$, of this algorithm? Count assignments of s only.

```
Algorithm Loop(n):
    s ← 0
    for i ← 1 to n do
        for j ← 1 to 2n do
            s ← s + i
```

every time in loop i

$n(1A + 1A + 2n(1A + 1A))$

$= 2 + n(2 + 2n(2))$

$= 2 + n(2 + 4n)$

$= 2 + 2n + 4n^2$ $O(n^2)$

(b). [3 marks] How many arrangements of the letters in MISSISSIPPI have all four S's together?

count total "letters" = 8! (SSSS) as 1

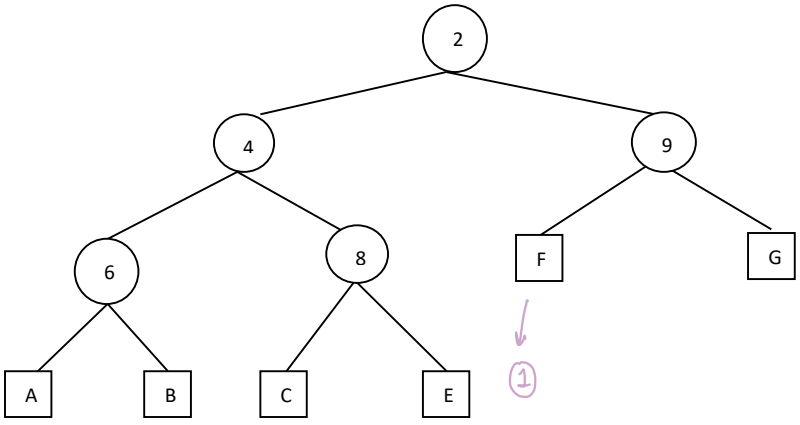
want: MISSSSIIPPI

arrangements = $\frac{8!}{4!\,2!}$

I's P's (repeated)

Rule of product ⇒ simultaneous

(c). [2 marks] Consider the following heap with integer key values.



Which leaf would an item with key 1 be initially inserted into? into F and bubble up to preserve heap shape property & bubble for heap order

2. (a). [3 marks] Prove that $\log^2 n$ is $o(n^{1/2})$.

little o: $\lim_{n\to\infty} \frac{f(n)}{g(n)} =$

little o: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Little Ω: $\lim_{n\to\infty} \frac{f(n)}{(n)} = \infty$

$\lim_{n\to\infty} \frac{\log^2 n}{n^{1/2}} = \frac{\infty}{\infty} \Rightarrow$ (log is squared) $\frac{2\log(n)\cdot\left(\frac{1}{\ln 2\cdot n}\right)}{2n^{-1/2}}$ (chain rule)

$= \frac{2\log n \cdot \frac{1}{\ln 2\cdot n}}{\frac{1}{n^{1/2}}} = 4\log(n)\cdot\frac{n^{1/2}}{\ln 2 n} = \frac{4}{\ln 2 n}\cdot\frac{\log(n)}{n^{-1/2}}$

$= \frac{4}{\ln 2n}\cdot\frac{\ln(2)\cdot n}{2n\cdot n}$ ... $\in O(n^{1/2})$

(b). [5 marks] Give a loop invariants argument to prove the correctness of selection sort.

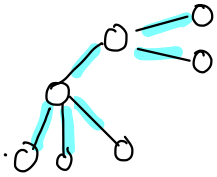
Algorithm selectionSort(A,n):
Input: Array A of size n
Output: Array A sorted

for $k \leftarrow 0$ **to** $n-2$ **do**
  min $\leftarrow k$
  for $j \leftarrow k+1$ **to** $n-1$ **do**
    if $A[j] < A[min]$ **then**
      $min \leftarrow j$
  swap($A[k]$, $A[min]$)

Selection sort: when i swap the next iteration with the min value found in array.

3. (a). [2 marks] Add up the number of nodes in a forest of at least two trees together with the number of edges. Can the number be even? Justify.

let number of nodes be m

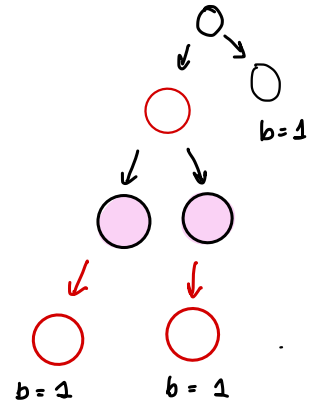


forest

yes, number can be even as in a tree, we connect 2 vertex with 1 edge (no parallel). That means, that to connect all the nodes, we need at least m-1 edges. So if we have an odd number of vertex, we can get an even # of edges.

(b). [3 marks] What is the least number of internal nodes in a red-black tree with 3 red edges?

→ alternating with red nodes (longest path)



black height: # of blacks to reach root

only way path will satisfy properties
- alternating red and black
  - ↳ we want "shortest" long path possible

at least 3 nodes

to form tree try to get away with recolour and rotate
- → red trees only in left
- → if we have 2 red children, red up.

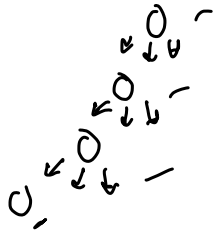
(c). [3 marks] What is the maximum number of nodes in a 2-3 tree with height 4 (recall, height of a leaf is 0 and leaves are empty/null)?

e f g n i j k r s t

insert, k

1

2



3

4 $3^{d-1} = 3^3 = 3 \cdot 3 \cdot 3 = 27$ as the max. number of children we can have is 3 per level, and leaves (in 4th height are 0).

4. Let T be a proper binary tree. Let the height of the tree be denoted by h. Justify all your answers below.

(a). [1 marks] What is the minimum number of external nodes in T in terms of h?

2^h * because we can have up to 2 children per level, and we want to know number of leaves max. (# of nodes at height h)

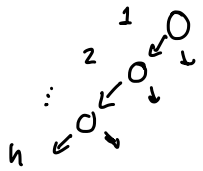
(b). [1 marks] What is the maximum number of external nodes in T, also in terms of h?

* we want at least 1 node as it is a binary tree.

$h + 1$

(c). [1 marks] What is the minimum number of internal nodes in terms of h?

h ∘ in case of having only 1 child



(d). [1 marks] What is the maximum number of internal nodes in terms of h?

$2^h - 1$ → - 1 as we don't want to count last level (leaves)

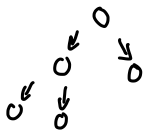
(e). [2 marks] Let n be the total number of nodes in T. What are the lower and upper bounds on n, in terms of height h?

↳ lower: $2h+1$ → 2 children per node plus root

↳ upper: $2^{h+1} - 1$ → max. of nodes, without leafs.

(f). [2 marks] What are the lower and upper bounds on the height of the tree in terms of n?

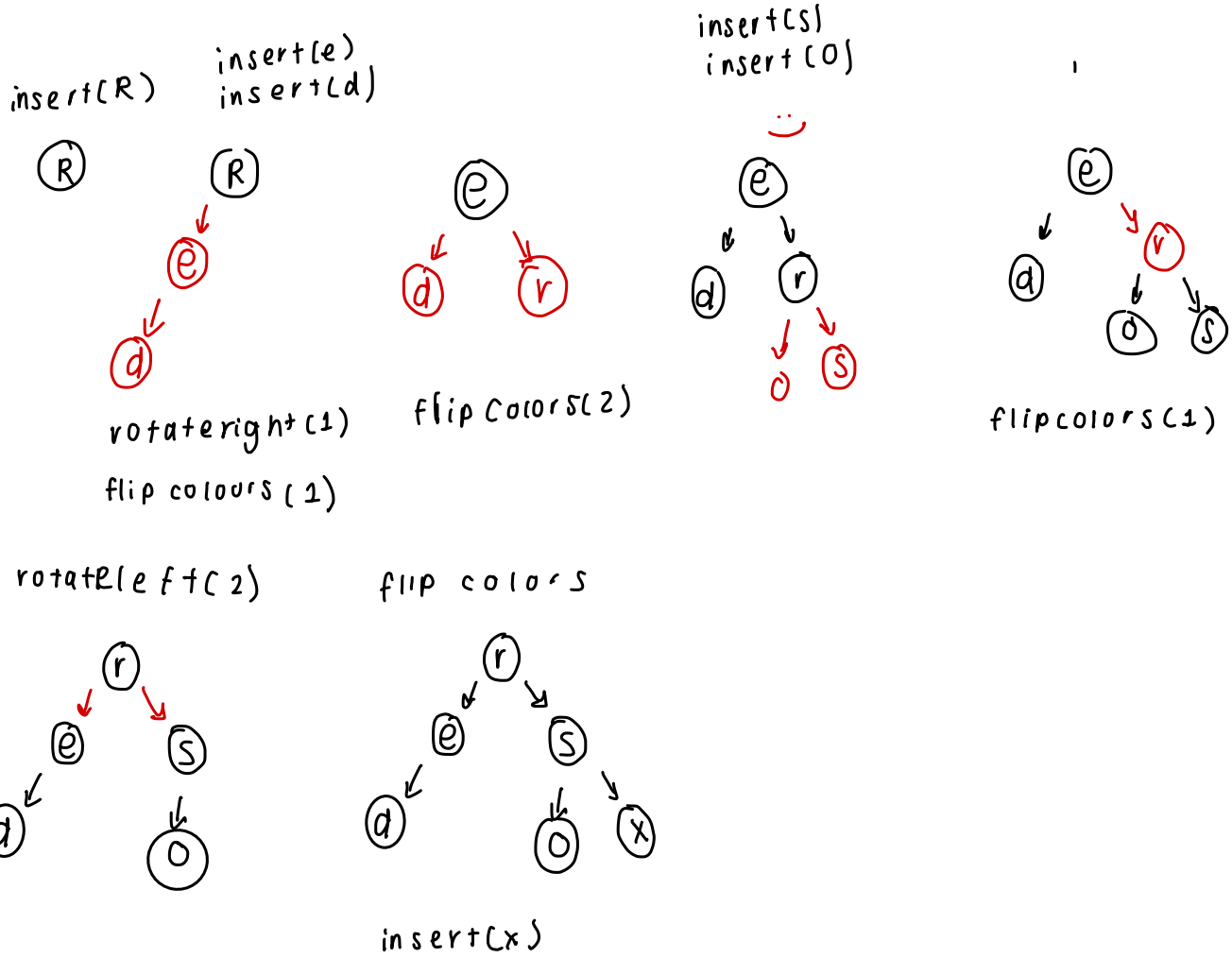
- lower bound is $\log(n)$: tree if tree is balanced



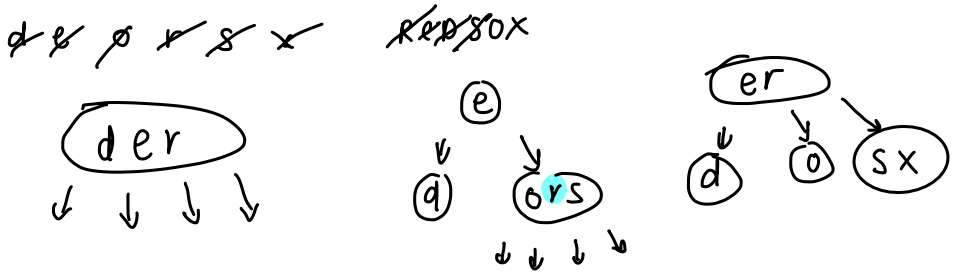
- upper bound is $O(n)$: tree is unbalanced.

5. (a). [6 marks] Insert the keys ~~R E D S O X~~ into an initially empty left-leaning red-black tree, in the given order. Show your work, there should be at least six trees along the way. Any time you need to invoke a rotateRight(h), rotateLeft(h) or flipColors(h), indicate it between trees and specify the node h to be passed to the method by its key.

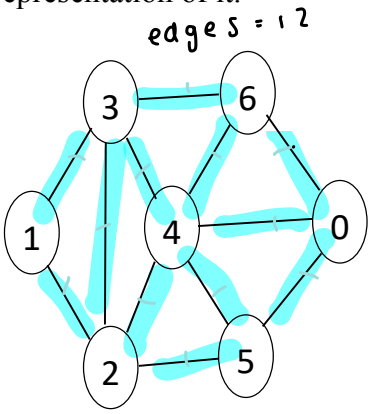
order: d e o r s x



(b). [2 marks] Draw the 2-3 tree that corresponds to the final red-black tree in (a).



6. (a). [3 marks] Consider the following simple undirected graph G and draw the adjacency-matrix representation of it.



"from"

to

row

col

	0	1	2	3	4	5	6
0	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0
2	0	1	0	1	1	1	0
3	0	1	1	0	1	0	1
4	1	0	1	1	0	1	1
5	1	0	1	0	1	0	0
6	0	0	0	1	1	0	0

adding row/colum is the degree

(b). [3 marks] Using pseudocode, write the graph method `numEdges()` which takes as input a simple undirected graph G, in adjacency-matrix form, and an integer n, the number of vertices in G (i.e. the size of the matrix). Your method should return an integer containing the number of edges in the graph.

```
Algorithm numEdges( G, n ):        -> we count edges twice, so
                                      divide total sum of edges

    count ← 1  // at least 1 edge connecting vertex
    for int i ← 1 to n  do: // column number
        for int j ← 1 to n  do: // row number
            if  G[j] = 1  do:
                count ← count + 1
            end
        end
    end
    result ← count / 2
    return result
end
```

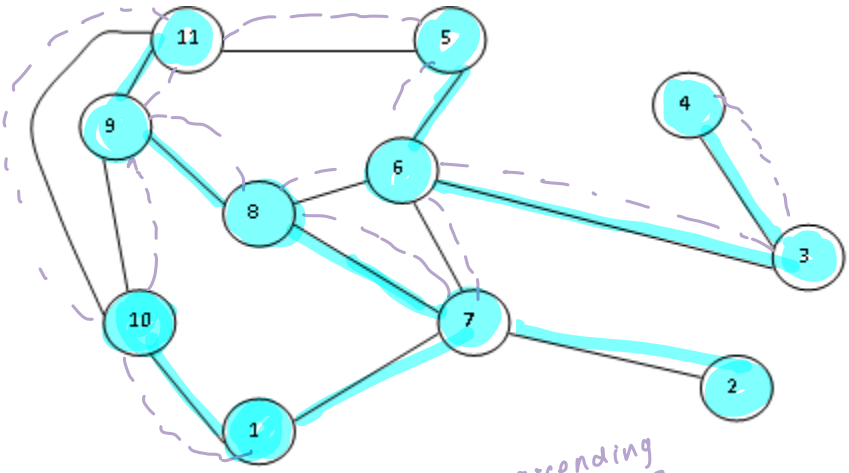
(c). [2 marks] Analyze the worst-case running time of your algorithm in (b).

$O(n^2)$ ⇒ double for loop

7. (a). [3 marks] Consider the following undirected graph G. Order the vertices as they are visited in a BFS traversal, starting at vertex 1. Assume adjacent vertices are given in ascending order according to their label.

use queue

de menor a mayor →



discovery edges

cross

BFS: go to new node and discover ALL of it, and then go to next node adjascent in ascending order.

↳ decide paths on val.

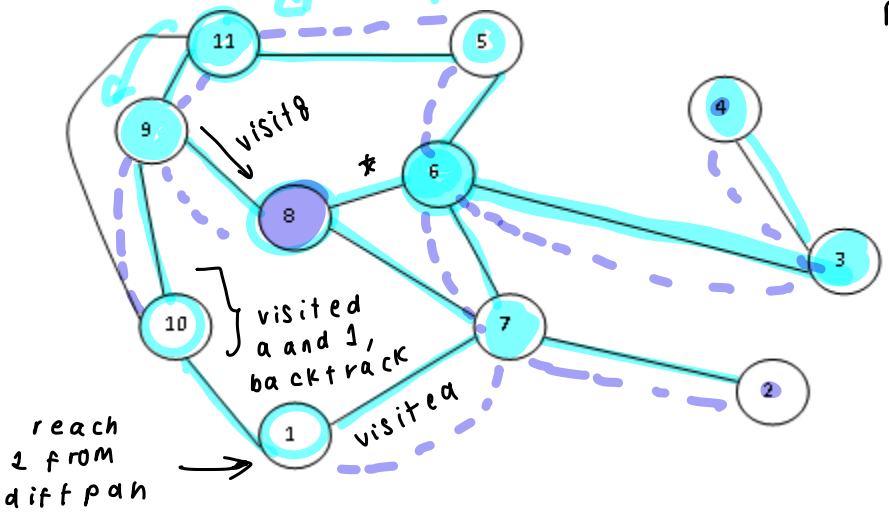
ascending ↑ from 7

1 7 10 2 6 8 9 11 3 5 4

(b). [1 mark] On the graph, indicate which edges are *discovery* edges and which are *cross* edges.

(c). [3 marks] Now consider the same graph G, again assuming adjacent vertices are given in ascending order according to their label. This time order the vertices according to their postorder, using a DFS traversal.

post order: values popped off stack



postorder: when back track

backtrack

preOrder: 1, 7, 2, 6, 3, 4, 5, 11, 9,

* up until this point 6 and 7 have been visited, back track

2 4 3 8 10 9 11 5 6 7 1

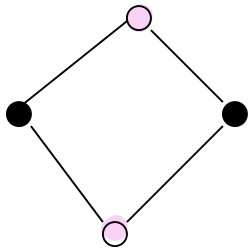
(b). [1 marks] On the graph, indicate which edges are *discovery* edges and which are *back* edges.

main difference: In BFS as soon as we see a new node, we visit all of its adjascents and continue from the smallest of adjascent (ascending order).

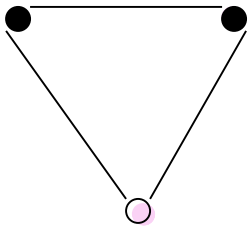
DFS: we visit an visit untill i reach lonely vertex and backtrack. Start from one before

8. [8 marks] A *2-colouring* of an undirected graph with n vertices and m edges is the assignment of one of two colours (say, black or white) to each vertex of the graph, so that no two adjacent nodes have the same colour. So, if there is an edge (u,v) in the graph, either node u is black and v is white or vice versa. Give (pseudocode!) an $O(n+m)$ time algorithm to 2-colour a graph or determine that no such colouring exists, and justify the running time. The following shows examples of graphs that are and are not 2-colourable:

2-colourable



Not 2-colourable



O(n+m) is runtime of BFS

O graph is colourable if we have even # vertex.

↳ traverse through using a stack

ouse BFS implementation with queue.

```
v ← node
q ← queue
for each edge e, insident to v do:
      if e is unexplored then
      enqueue(v)
```

push unexplored vertex into a stack.
↳ have 2 temporary stacks stk1, stk2
pop until not empty
pop()x twice { stk1.(push(pop1)) ; stk2.(push(pop2)) }

if #elements in stk1 = #ele. in stk2 then graph is 2 colourable, otherwise it's not

THE END