THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 221: MIDTERM EXAMINATION – October 10, 2019

Full Name: _____ CS Account: _____

Signature: _____ UBC Student #: |_|_|_|_|_|_|_|_|_|

Important notes about this examination

1. You have 120 minutes to complete this exam.
2. No notes or electronics of any kind are allowed.
3. Good luck!

Student Conduct during Examinations

1. Each examination candidate must be prepared to produce, upon the request of the invigilator or examiner, his or her UBCcard for identification.
2. Examination candidates are not permitted to ask questions of the examiners or invigilators, except in cases of supposed errors or ambiguities in examination questions, illegible or missing material, or the like.
3. No examination candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination. Should the examination run forty-five (45) minutes or less, no examination candidate shall be permitted to enter the examination room once the examination has begun.
4. Examination candidates must conduct themselves honestly and in accordance with established rules for a given examination, which will be articulated by the examiner or invigilator prior to the examination commencing. Should dishonest behaviour be observed by the examiner(s) or invigilator(s), pleas of accident or forgetfulness shall not be received.
5. Examination candidates suspected of any of the following, or any other similar practices, may be immediately dismissed from the examination by the examiner/invigilator, and may be subject to disciplinary action:
   i. speaking or communicating with other examination candidates, unless otherwise authorized;
   ii. purposely exposing written papers to the view of other examination candidates or imaging devices;
   iii. purposely viewing the written papers of other examination candidates;
   iv. using or having visible at the place of writing any books, papers or other memory aid devices other than those authorized by the examiner(s); and,
   v. using or operating electronic devices including but not limited to telephones, calculators, computers, or similar devices other than those authorized by the examiner(s)—(electronic devices other than those authorized by the examiner(s) must be completely powered down if present at the place of writing).
6. Examination candidates must not destroy or damage any examination material, must hand in all examination papers, and must not take any examination material from the examination room without permission of the examiner or invigilator.
7. Notwithstanding the above, for any mode of examination that does not fall into the traditional, paper-based method, examination candidates shall adhere to any special rules for conduct as established and articulated by the examiner.
8. Examination candidates must follow any additional examination rules or directions communicated by the examiner(s) or invigilator(s).

Please do not write in this space:

CPSC 221 2019W1: Midterm Exam 1

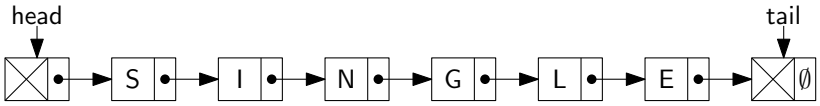
October 10, 2019

1 Who gets the marks? [1 mark] Write your 4 or 5 digit CSID here MMMMM

2 Deque euqeD [6-marks]

A double ended queue or, *deque* for short, is a queue that allows push and pop operations to be performed from both ends of the queue. The four operations `pushL`, `popL`, `pushR`, and `popR` push and pop from the left and right end of the queue, respectively. In the problems below, please assume that the data element in a node **cannot** be changed.

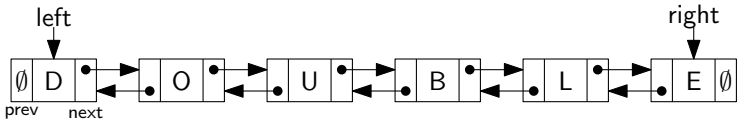
(a) **[2-marks]** Suppose we use a singly-linked list with a dummy head and a dummy tail to implement a deque. The head of the list is at the left of the deque and the tail is at the right.



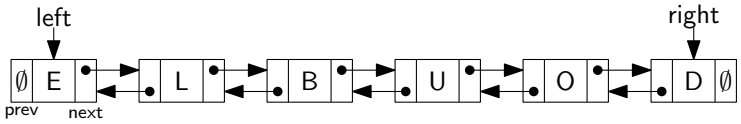
Which operations must take $\Omega(n)$ time in this implementation of a deque containing n items (assuming you may use only a constant amount of additional space).

○ `pushL` ○ `popL` ● `pushR` ● `popR` ○ none

(b) **[4-marks]** Now suppose we implement the deque as a doubly-linked list with no dummy nodes.



Add two lines to complete the following code to reverse the contents of the deque.



You may use the function `swap` which **only** swaps two `Node` pointers (which are passed by reference).

```
struct Node { string data; Node *prev, *next; };
void DLL::reverse() {
  Node *p=left;
  while( p != NULL ) {
```

```
swap(p->next,p->prev);
```

```
p=p->prev;
```

```
  }
  swap(left,right);
}
```

3 Lines printed [12-marks]

Enter the letter of the code fragment next to the number of lines printed by the fragment as a function of n, this function in asymptotic notation, and the rank of this function compared to the other fragments (1=smallest). Some boxes will be empty and some boxes might contain two letters. No box in the # Lines printed column contains two letters. You may assume n is a power of 2.

Code fragment		# Lines printed		Asymptotic		Rank
A	`for(int i=1; i<n*n; i=i*2)` `  cout << "A" << endl;`		$\log_2(n)$	D	$\Theta(n^2)$	A	1

B	`for(int i=1; i<n; i=i*2)`
	`  for(int j=1; j<=i; j++)`
	`  cout << "B" << endl;`
		D	$n(n-1)/2$	C	$\Theta(\sqrt{n})$	C	2

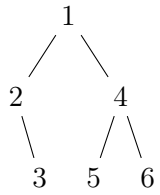
C	`int i=1; int s=1;`
	`while(s <= n) {`
	`  cout << "C" << endl;`
	`  s = s + 2*i + 1;`
	`  i = i + 1;`
	`}`
		C	$\lfloor\sqrt{n}\rfloor$	A	$\Theta(\log n)$	B	3

D	`for(int i=0; i<n-1; i++)`
	`  for(int j=i+1; j<n; j++)`
	`  cout << "D" << endl;`
		A	$\log_2(n^2)$	B	$\Theta(n)$	D	4

			n^2				
		B	$n-1$				

4 Tree traversal [10-marks]

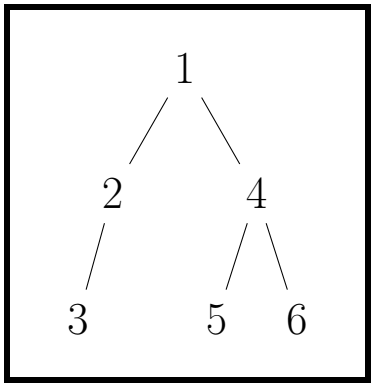
For this question, you may assume that all trees are **binary** trees and that their nodes have been labeled from 1 to n in the order of a **preorder** traversal. For example, one such labeled tree is:



(a) **[2-marks]** What is the sequence of labels produced by a postorder traversal of the above tree?

3 2 5 6 4 1

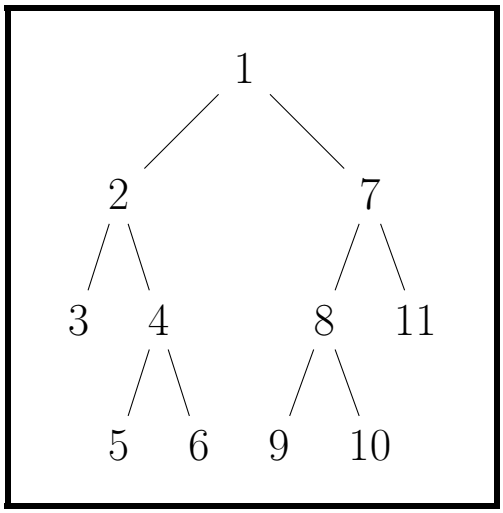
(b) **[2-marks]** Draw a **different** binary tree, with its preorder labeling, that produces the same sequence of labels by a postorder traversal as the above tree.



(c) **[6-marks]** This sequence of labels was produced by a postorder traversal of a **full** binary tree whose **preorder** traversal was labeled from 1 to 11 (in order).

$$\underline{3\ 5\ 6\ 4\ 2}\ \boxed{9\ 10\ 8\ 11\ 7}\ \textcircled{1}$$

Circle the label of the root in the sequence, underline the labels in the left subtree of the root, and draw a box around the labels in the right subtree of the root. Finally, use the recursive definition of a full binary tree to reconstruct the original tree,



5 Weighty measures [11-marks]

You have been given a prize! It's a basket of n eggs. One of the eggs contains a gold coin. The other eggs are rotten, and you do **not** want to open one of those. Fortunately, the gold egg weighs slightly more than a rotten egg (all rotten eggs weigh the same) *and* you have a balance that can compare two piles of eggs. The balance will tell you if one pile of eggs weighs **less** or **more** or **the same** as the other pile. You should assume that the only way to gain information about eggs is to weigh them against other eggs. So no metal detectors or x-ray machines or anything like that.

(a) **[3-marks]** Fill in the following table with the fewest number of weighings in the worst case needed to find the gold egg among n eggs for $n = 2, \ldots, 10$.

$n=1$	2	3	4	5	6	7	8	9	10
0	1	1	2	2	2	2	2	2	3

(b) **[4-marks]** What is the **first** comparison one should perform to find the gold egg among n eggs using the fewest number of weighings in the worst case?

Weigh one pile of $\lfloor (n+1)/3 \rfloor$ eggs against another pile of $\lfloor (n+1)/3 \rfloor$ eggs.

(c) **[4-marks]** What is the fewest number of weighings, as a function of n, that the best algorithm needs in the worst case to find the gold egg? Your solution should not be a recurrence, contain a summation, or use asymptotic notation. (Note: This is a question about the complexity of solving a problem using a particular model of computation.)

$\lceil \log_3 n \rceil$

6 Quicksort [14-marks]

Quicksort is a recursive sort like mergesort. It chooses one element p, called *the pivot*, from its input array A and splits the input into those elements that are Lower, Equal, and Higher than the pivot. Then, it recursively sorts the Lower elements and recursively sorts the Higher elements. Finally, it concatenates (using $\circ$) the sorted Lower, the Equal, and the sorted Higher elements to produce its output.

Here's an outline of how it works:

1. **Quicksort**$(A[0 \ldots n-1])$
2. if $(n \leq 1)$ return A
3. $p =$ **pivot**(A)
4. $Lo =$ elements in A that are $< p$
5. $Eq =$ elements in A that are $= p$ (including p)
6. $Hi =$ elements in A that are $> p$
7. $Lo =$ **Quicksort**(Lo)
8. $Hi =$ **Quicksort**(Hi)
9. return $Lo \circ Eq \circ Hi$

(a) **[3-marks]** Let $Q(n)$ be the worst-case number of steps taken by **Quicksort** on an input of size n. Assume that lines 3,4,5,6, and 9 take a total of cn steps. What is the recurrence equation for $Q(n)$ expressed using functions of c, n, $|Lo|$, and $|Hi|$ (where $|X|$ is the size of array X)?

$$Q(0) = Q(1) = 1$$

$$Q(n) = \boxed{Q(|Lo|) + Q(|Hi|) + cn} \quad \text{for } n > 1$$

(b) **[1-marks]** The function **pivot**(A) returns an element from the array A. This implies that

$$|Lo| + |Hi| \leq \boxed{n - 1}$$

(c) **[3-marks]** Recall that we defined $Q(n)$ to be the worst-case number of steps taken by **Quicksort** on an input of size n. Suppose **pivot** always returns a pivot p so that one of Lo and Hi is empty. What is $Q(n)$ in this case?

○ $\Theta(n)$ ● $\Theta(n^2)$ ○ $\Theta(\log n)$ ○ $\Theta(n^2 \log n)$ ○ $\Theta(n \log n)$

(d) **[3-marks]** Suppose function **pivot** always returns a pivot p so that both $|Hi|$ and $|Lo|$ are at most $2n/3$. What is $Q(n)$ in this case?

○ $\Theta(n)$ ○ $\Theta(n^2)$ ○ $\Theta(\log n)$ ○ $\Theta(n^2 \log n)$ ● $\Theta(n \log n)$

(e) **[3-marks]** Suppose we remove lines 7 and 8 from **Quicksort** so it makes no recursive calls but only splits the input into Lo, Eq, and Hi and then returns $Lo \circ Eq \circ Hi$. What are the possible values for the pivot p if the returned array is:

10, 7, 5, 12, 19, 13, 16, 21, 22, 39, 35, 28, 37, 46, 56, 61, 54, 73, 86, 75, 99

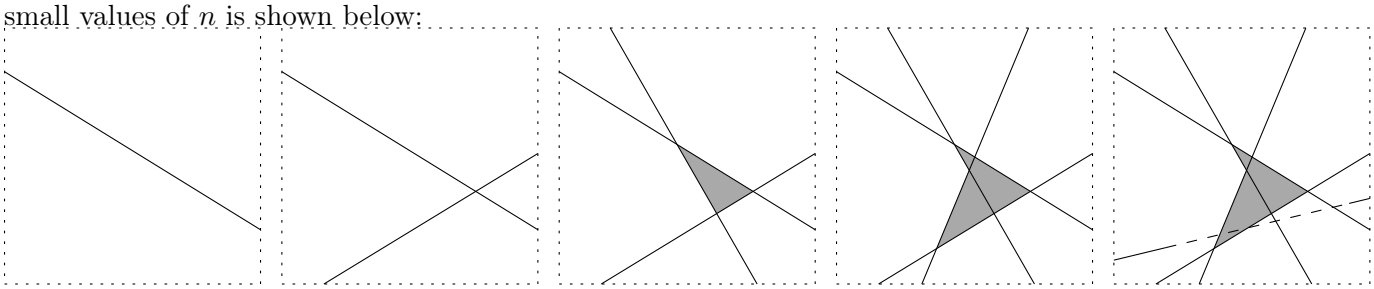
Possible pivots: 12,21,22,46,73,99

(f) **[1-marks]** Is **Quicksort** asymptotically faster than Mergesort? ○ Yes ● No

7 City planning [7-marks]

We've hired a team of engineers to build streets for a new city. The amazing part is that every street is perfectly straight and **every pair of streets intersects**! The streets divide the city into regions. Some regions are completely surrounded by streets (they are *bounded*) and others are not. If the engineers build n streets, how many bounded regions are there?

Let $B(n)$ be the number bounded regions in such an arrangement of streets. The value of $B(n)$ for small values of n is shown below:



$B(1) = 0$ $B(2) = 0$ $B(3) = 1$ $B(4) = 3$ $B(5) =$ 6

(a) **[1-marks]** Fill in the value of $B(5)$. You should trace the dashed portion of the street to see exactly how bounded regions are created.

(b) **[3-marks]** Complete the following recurrence relation for $B(n)$:

$$B(1) = 0$$

$$B(n) = B(\boxed{n-1}) + \boxed{n-2} \quad \text{for } n \geq 2$$

(c) **[3-marks]** What is $B(n)$ as a function of n? Your solution should not be a recurrence, contain a summation, or use asymptotic notation. (To check your formula, note that $B(6) = 10$.)

$$\boxed{(n-1)(n-2)/2}$$

8 Big sum [12-marks]

The following code is intended to find the largest sum of contiguous elements (elements that occupy consecutive locations) in an input vector of positive and negative integers. But you're not sure it works correctly.

```
1  int bigSum(vector<int> & A) {
2    vector<int> bestAt(A.size());
3    bestAt[0]=A[0];
4    int best=0;
5    for( int i=1; i<A.size(); i++ ) {
6      if( bestAt[i-1] < 0 ) {
7        bestAt[i] = A[i];
8      } else {
9        bestAt[i] = bestAt[i-1]+A[i];
10     }
11     if( best < bestAt[i] ) best = bestAt[i];
12   }
13   return best;
14 }
```

(a) **[3-marks]** Show the contents of the `bestAt` vector at the end of the call to `bigSum` on the following input array `A`:

	0	1	2	3	4	5	6	7	8
A	-3	1	3	-2	4	6	-4	5	-1
bestAt	-3	1	4	2	6	12	8	13	12

(b) **[2-marks]** Fill in the blanks for the following loop invariant:

For all $i \in \{1, 2, \ldots, n\}$, where $n =$ `A.size()`, at the start of iteration i,

(A) `bestAt[i-1]` is the largest sum of contiguous elements in `A` ending at position $i - 1$

(B) `best` is the largest sum of contiguous elements in `A[0..i-1]`

(c) **[2-marks]** Is the base case true? ○ Yes ● No

If not, explain how to modify the code to make it true:

After line 4 add: `if( A[0] > 0 ) best = A[0];`

(d) **[2-marks]** Fill in the blanks of the following **key fact** about sums of contiguous sequences ending at position i.

The contiguous sequence with largest sum ending at position i is either:

1) the contiguous sequence with largest sum ending at position $i-1$

followed by $A[i]$, or

2) just $A[i]$

(e) **[2-marks]** Suppose that at the start of iteration i the loop invariant (from part (b)) is true (inductive hypothesis). We want to show that at the start of iteration $i+1$ (i.e., the end of iteration i) the loop invariant is true. Fill in the blanks below to explain how to use the key fact to show that invariant (A) holds for iteration $i+1$.

If `bestAt[i-1] < 0` then the inductive hypothesis and the key fact imply that `bestAt[i] =` `A[i]`.

Otherwise, the inductive hypothesis and the key fact imply that `bestAt[i] =` `bestAt[i-1]+A[i]`.

(f) **[1-marks]** What line of code ensures that invariant (B) hold for iteration $i+1$? 11

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.