THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 221: MIDTERM EXAMINATION – February 5, 2020

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. You must submit all pages of this booklet, including any detached pages.
4. In case of uncertainty, clearly write down your assumptions.
5. 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 2019W2: Midterm Exam 1

February 5, 2020

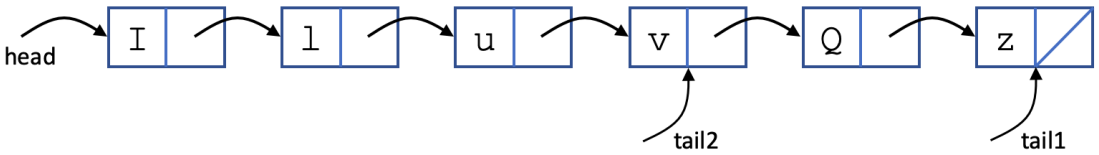
1 Who gets the marks? [1 mark] Write your 4 or 5 digit CSID here MMMMM

NOTE: You may find the following formulas useful:

- $\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1}$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

2 Two-tailed Queues [12-marks]

Suppose we implement a queue using a modification of a Singly-Linked-List, as shown in the illustration below. Our innovative list maintains one tail pointer whose target is the last node, and an additional tail pointer pointing to the node two *before* the last.



(a) **[4-marks]** If we implement `enqueue()` to occur at the head of the list, and `dequeue()` to occur at the tail, what can we say about the worst case running times of the best implementations of the functions? In the problems below, please assume that the data element in a node **cannot** be changed.

Fill in the bubble if the given function runs in the given asymptotic time.

	$\Omega(1)$	$O(1)$	$\Theta(1)$	$\Omega(n)$	$O(n)$	$\Theta(n)$
`enqueue`	●	●	●	○	●	○
`dequeue`	●	○	○	●	●	●

(b) **[4-marks]** If we implement `dequeue()` to occur at the head of the list, and `enqueue()` to occur at the tail, what can we say about the worst case running times of the best implementations of the functions? In the problems below, please assume that the data element in a node **cannot** be changed.

Fill in the bubble if the given function runs in the given asymptotic time.

	$\Omega(1)$	$O(1)$	$\Theta(1)$	$\Omega(n)$	$O(n)$	$\Theta(n)$
`enqueue`	●	●	●	○	●	○
`dequeue`	●	●	●	○	●	○

3 Love letters [14-marks]

For each code fragment, enter the exact number of lines printed as a function of n. Then, select the appropriate asymptotic running time from options A. to F. below. Finally, in the boxes at the bottom of the page, enter the code fragment numbers ordered by running time, fastest to slowest (breaking ties arbitrarily).

You may assume n is a power of 2, and that $n \geq 1$. Further, assume that function `lg(n)` returns $\log_2 n$, and that function `pow(a,b)` returns a^b.

A. $\Theta(1)$ B. $\Theta(\log n)$ C. $\Theta(n)$ D. $\Theta(n \log n)$ E. $\Theta(n^2)$ F. $\Theta(n^3)$

	Code fragment	# Lines printed	Asymptotic
1	`for(int i=pow(n,3); i>1; i=i/2)` `  cout << "1" << endl;`	$3 \log_2 n$	B

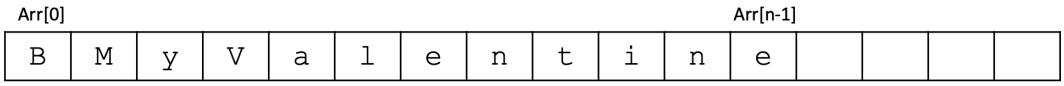
2	`for(int i=2; i<n; i++)`
	`  for(int j=i-1; j>0; j--)`
	`  cout << "2" << endl;`
		$\frac{(n-1)(n-2)}{2}$	E

3	`int s=0;`
	`while(s <= lg(n)) {`
	`  for (int i=0; i<pow(2,s); i++){`
	`  cout << "3" << endl;`
	`  }`
	`  s++;`
	`}`
		$2n - 1$	C

Write the code numbers from (asymptotically) fastest to slowest: 1 3 2

4 Dynamite! [12-marks]

Suppose we are designing a class to represent a string of characters, and that we implement the class using a dynamically sized contiguous block of memory like an array or vector. The array resizes by doubling when it is full, and by halving when it less than $\frac{1}{4}$ full. It also always stores the first character of the string in position 0, and the last character of the string in position $n-1$, where n is the length of the string.



In each scenario below, select the best asymptotic growth rate from the given list, and write its letter in the box:

A. $\Theta(1)$
B. $\Theta(\log n)$
C. $\Theta(n)$
D. $\Theta(n \log n)$
E. $\Theta(n^2)$
F. $\Theta(n^3)$

A	Return the character in position $\frac{n}{2}$ in the array, assuming the string has length n.
C	Maximum amount of memory used by the array to represent a string of length n.
C	Worst case time to perform n consecutive additions to the end of an originally empty string.
E	Worst case time to remove $\frac{n}{2}$ characters, one at a time, from the front of an n character string.
B	Maximum number of times the array is resized when performing n consecutive additions to the end of an originally empty string.
A	Maximum number of times the array is resized when performing $\frac{n}{2}$ consecutive additions to the end a string of length n.

5 Sweetarts hearts for you, my sweetheart [11-marks]

Valentine's day is coming soon! You want to send a candy gram to your secret crush, but it's late and the candy shop might be sold out of the best candies. To save yourself the embarrassment of being repeatedly denied sale of your top choices, you will instead purchase the k^{th}-ranked candy on your list, which is stored as an unordered **1-indexed** array (the first index is 1) whose values are candy ratings. A natural solution is to sort your array of candy by value, and then choose the k^{th} position easily, but here you will try a different algorithm which achieves the same result without (fully) sorting the array.

The algorithm is provided with a parameter k, and returns the element in the input collection that is the k^{th} **lowest** value (when sorted in increasing order).

The algorithm operates by choosing one element p, called the *pivot*, from the input array A and splits the input into elements which are Lower than the pivot value, and elements which are Higher than the pivot. Assume that your array does not contain duplicate values. Next the algorithm checks how many elements are in Lower, and Higher, and recurses on one of the pieces, possibly with a modified value of k.

(a) **[2-marks]** Suppose that $k = 5$ and we have completed one round of splitting, and the array is in the following arrangement with the pivot value highlighted.

piece	*Lower*								***p***	*Higher*				
value	13	7	18	24	5	23	11	21	**27**	32	51	42	48	37
index	1	2	3	4	5	6	7	8	**9**	10	11	12	13	14

On which piece of the array should we recurse? (*Lower* or *Higher*) What should be the parameter k for this recursive call? Enter your answers in the boxes.

Array piece: Lower k : 5

(b) **[2-marks]** Suppose that $k = 5$ and we have completed one round of splitting, and the array is in the following arrangement with the pivot value highlighted.

piece	*Lower*		***p***	*Higher*										
value	7	5	**11**	23	21	13	42	48	32	51	18	27	24	37
index	1	2	**3**	4	5	6	7	8	9	10	11	12	13	14

On which piece of the array should we recurse? (*Lower* or *Higher*) What should be the parameter k for this recursive call? Enter your answers in the boxes.

Array piece: Higher k : 2

(c) **[3-marks]** With your understanding of the recursive call and parameters from the above two questions, fill in the blanks to complete the pseudocode of the `SelectCandy` function below: (Note that in the pseudocode we denote the length of an array A by $|A|$.)

1. **SelectCandy**($A\,[1 \ldots n]$, k)
2. $p =$ **pivot**(A) // p is the index of the pivot element, randomly chosen from indices of A
3. $Lo =$ array of elements in A that are $< A\,[p]$
4. $Hi =$ array of elements in A that are $> A\,[p]$
5. if ($k = p$) return A[p]
6. else if ($|Lo| > k$)
7. return **SelectCandy**(Lo , k)
8. else if ($|Lo| < k$)
9. return **SelectCandy**(Hi , (k-p))

(d) **[2-marks]** Let $V(n)$ be the number of steps taken by **SelectCandy** on an input of size n, and suppose that the selection of p always divides the array evenly so that $|Lo| = |Hi|$. Assume that lines 2, 3, 4, 6, and 8 take a total of cn steps. What is the recurrence equation for $V(n)$ expressed using functions of c, and n?

$$V(0) = V(1) = 1$$

$$V(n) = \boxed{V(n/2) + cn} \quad \text{for } n > 1$$

(e) **[2-marks]** What is asymptotic value of $V(n)$?

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

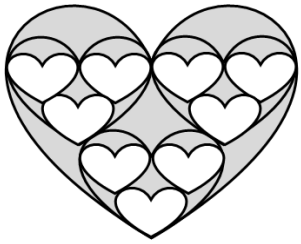
6 Put a little love in your heart [7-marks]

It's time to reserve little portions of your heart for all of the people you secretly admire!

Carefully study the diagrams below. Let $H(n)$ be the number of enclosed, non-heart-shaped regions. These regions have been shaded for clarity. The value of $H(n)$ for small values of n is shown below:

$H(1) = 0$

$H(2) = 5$



$H(3) = 20$

$H(4) =$ ☐

(a) **[1-marks]** Fill in the value of $H(4)$. Observe how many new enclosed regions will be created and added to the existing regions.

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

$$H(1) = 0$$

$$H(n) = \boxed{3}\, H(\boxed{n-1}) + \boxed{5} \quad \text{for } n \geq 1$$

(c) **[3-marks]** What is $H(n)$ 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 $H(5) = 200$.)

$$\frac{5}{2}(3^{n-1} - 1)$$

7 Ms. Americana [16-marks]

Suppose you want to rearrange a music playlist so that every other song you hear is by Taylor Swift, and that no two songs in a row are by the same artist. In this problem you will formulate, solve, prove correct, and analyze an algorithm to achieve this task.

The data for the problem is simply a `vector<char>` whose entries each represent a song, denoted by a single character indicating its artist. The letter `T` represents Taylor Swift. So, for example, the array `PLbad = [T, T, A, B]` consists of two songs by Taylor, followed by one whose artist is denoted by 'A' followed by a fourth whose artist is denoted by 'B'. That play list does not satisfy our constraints, though, because the songs do not alternate between Taylor Swift and other artists. The play lists represented by `PLgood1 = [T, A, T, B]`, `PLgood2 = [B, T, A, T]`, and `PLgood3 = [T, A, T, A]`, are good.

For simplicity, you may assume that the playlist is non-empty, that it has even length, and that exactly half of the songs are by Taylor Swift. Also, please use n to represent `A.size()`.

Consider the following helper function `int findNext(vector<char> & A, int start, bool Tay)`:

```
int findNext(vector<char> & A, int start, bool Tay) {
    int curr = start;
    while ((A[curr] == 'T') == Tay){
        curr++;
    }
    return curr;
}
```

Assume also that you are given a function `void swap(char & a, char & b)` that can be used to exchange the data in a vector. In your arguments for parts (e) through (g), please refer to this assumption as **(fact A)**.

Finally, our complete algorithm for solving the playlist problem is given here:

```
void playList(vector<char> & A) {
    bool Tay = (A[0] == 'T');
    for (int i = 1; i < A.size(); i++){
        int next = findNext(A, i, Tay);
        swap(A[next],A[i]);
        Tay = !Tay;
    }
}
```

(a) **[2-marks]** Show the state of the `A` vector at the start of iteration $i = 5$.

	0	1	2	3	4	5	6	7
A	T	A	B	C	T	T	D	T
A at $i = 5$	T	A	T	C	T	B	D	T

(b) **[2-marks]** What does function `findNext` do? (Fill in the blanks to answer.) In your arguments for parts (e) through (g), refer to these observations as **(fact B)**.

If `Tay == True`: `findNext` returns [index of first non-'T'] in `A[` [i] ... [n-1] `]`.

If `Tay == False`: `findNext` returns [index of first 'T'] in `A[` [i] ... [n-1] `]`.

(c) **[2-marks]** Fill in the blanks for the following loop invariant for `playList`:
For all $i \in \{1, 2, \ldots, n\}$, at the start of iteration i,

(inv A) the play list represented by `A[` [0] ... [i-1] `]` consists of Taylor Swift songs in every other location, and no two consecutive songs are by the same artist, and

(inv B) variable `Tay` is `True` if and only if [A[i-1]=='T']. (Write an expression here–no words!)

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

For the inductive step of the correctness proof, we assume that the loop invariant is true at the beginning of iteration i and show that it is true at the beginning of iteration $i+1$. In your reasoning below, please refer to the inductive assumptions as **inv A(i)** and **inv B(i)**, corresponding to their descriptions in part (c) above. You may also use **fact A** and **fact B**.

(e) **[2-marks]** Which three invariants, facts, or lines of code, guarantee that at the beginning of iteration $i+1$, `A[i-1]`$\neq$ `A[i]`?

[INV B] [FACT B] [FACT A]

(f) **[1-marks]** `A[i-1]`$\neq$ `A[i]` together with one other observation, restores invariant A for the start of iteration $i+1$. What is that observation? (Answer with an invariant, fact, or line of code.)

[INV A]

(g) **[1-marks]** Finally, invariant B is restored for the start of iteration $i+1$ by one line of code from function `playList`. Which one?

[LINE 6]

(h) **[2-marks]** Our problem is completely solved if we observe that at the start of iteration $i =$ [n], invariant [A] gives us what we want! (Choose part A or B of the invariant.)

(i) **[2-marks]** The worst case asymptotic running time of `playList` is:

○ $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ● $\Theta(n^2)$

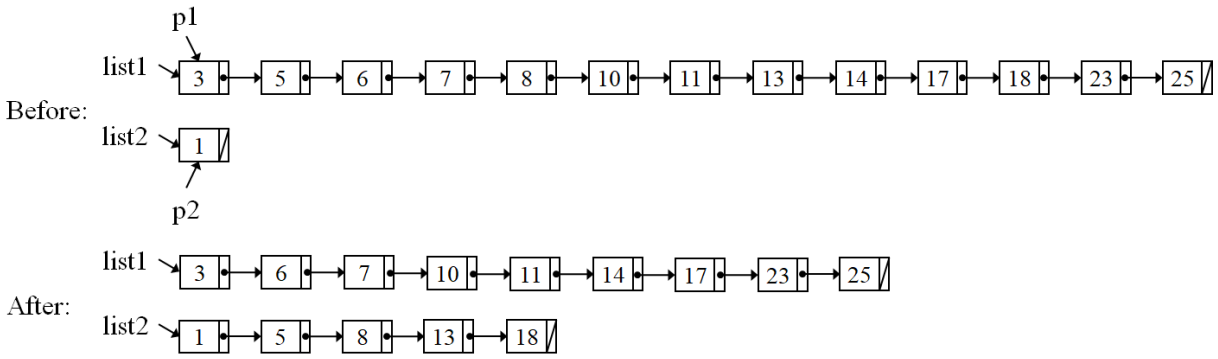
8 Kiss on my list [7-marks]

In the problem below, we have given you "before and after" models of linked lists. Your task is to transform the "before" into the "after" using simple pointer manipulations on the list nodes. Refer to the elements of the list nodes using the `Node` class below. Your solution should follow these guidelines:

- Any variables listed in the picture have already been declared and can be used in your solution.
- You may write loops to simplify your solutions, but your answers don't need to be general... they just need to work on the given lists. (Don't worry about even/odd length, or empty lists, for example.)
- You may **not** create or destroy any Node objects.

```
class Node {
   public:
      int data;
      Node * next;
      Node(int e): data(e), next(NULL) {} };
```

Perform the necessary transformation, without declaring any additional local pointer variables, without referring to the `list1` or `list2` variables, and without referring to the `data` attribute. Pointers `p1` and `p2` should be set to `NULL` when your code finishes.



line #	
1	while (p1->next != NULL) {
2	p2->next = p1->next; // required first move.
3	p2 = p2->next; // could be in different order if next line is different choice
4	p1->next = p2->next; // could be p1->next = p1->next->next;
5	p2->next = NULL; // not necessary until end
6	p1 = p1->next->next;
7	}
8	p1=p2=NULL;
9	
10	

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.