02092

THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 221: MIDTERM EXAMINATION – October 19, 2016

Last Name: Liu First Name: Zhuo Lun (Ryan)

Signature: [signature] UBC Student #: 3 0 3 2 8 1 4 1

Important notes about this examination

1. You have **90 minutes** to solve the 8 problems on this exam.
2. A total of **75 marks** is available. You may want to complete what you consider to be the easiest questions first!
3. Ensure that you clearly indicate a legible answer for each question.
4. You are allowed one sheet of paper (US Letter size or metric A4 size) of notes. Besides this, no notes, aides, or electronic equipment are allowed.
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:

Question 1:	13
Question 2:	4
Question 3:	5
Question 4:	8
Question 5:	3
Question 6:	4
Question 7:	3
Question 8:	8

0 236366 453321

02092

This page intentionally left blank.

If you put solutions here or anywhere other than the blank provided for each solution, you must *clearly* indicate which problem the solution goes with and also indicate where the solution is at the designated area for that problem's solution.

1 Asymptotic Relationships [15 marks]

02092

Circle the statements that are correct.

$n^2 \in O(n^4)$	$n^3 + 100n \in \Theta(n^3)$	$\sqrt{n} \in \Omega(\lg n)$
$n \log n \in \Omega(n^{1.01})$	$\log n \in \Theta\left(\frac{\log_3(n)}{\lg n}\right)$	$n^{100} \in O(3^n)$
$n \log^3 n \in \Omega(n^2 \log n)$	$3n^5 + 17 \in \Theta(17n^5 + 3)$	$2^n \in \Theta(2^{2n})$
$\log(n!) \in O(n)$	$n^3 \in \Omega(10n^3 + 100n^2 + 1000n)$	$\log \log n \in \Theta(\log^2 n)$
$\sum_{i=1}^{n}(2i+1) \in \Theta(n^2)$	$1,000,000 \in O(n)$	$2^{2^n} \in \Omega(3^n)$

Remarks:

- $\log^3 n = (\log n)^3$
- $\log_b a = \dfrac{\log_c a}{\log_c b}$
- You can assume that $\log n = \log_{10} n$ (although it does not really matter).

2 Big-O Proof [10 marks]

0209

For this problem, you must **formally prove** that

$$\text{if } f(n) \in O(h(n)) \text{ and } g(n) \in O(h(n)) \text{ then } f(n) + g(n) \in O(h(n)).$$

Since this is a formal proof, it will be based on the definition of big-O notation (that uses c and n_0).

Hint:

(1) Write down what $f(n) \in O(h(n))$ means according to the definition.

(2) Write down what $g(n) \in O(h(n))$ means according to the definition (you should call constants c and n_0 something slightly different, as it's very unlikely, they are same as in (1)).

(3) Do some algebra to show that $f(n) + g(n) \in O(h(n))$.

Gen. form. → By the def'n of Big-O, we must find positive constants c and n_0 st $\forall n \geq n_0$, $T(n) \leq c\, f(n)$.

1. If $f(n) \in O(h(n))$, then using Big-O def'n, $\exists$ positive constants c and n_0 st $\forall n \geq n_0$, $f(n) \leq c \cdot h(n)$

2. If $g(n) \in O(h(n))$, then using Big-O def'n, $\exists$ positive constants d and n_1 st $\forall n \geq n_1$, $g(n) \leq d \cdot h(n)$

3. If $f(n) \in O(h(n))$ and $g(n) \in O(h(n))$, then we can show $f(n) + g(n) \in O(h(n))$.

This is what you need to prove don't use it here

Since $f(n)$ and $g(n) \in O(h(n))$, the sum will also $\in O(h(n))$. ???

We can see this if we use additive property st. $T(n) = f(n) + g(n)$, if $f(n)$ and $g(n) \in O(h(n))$, addition can be at most $O(2 \cdot h(n))$ → since 2 is a constant, we can bring it out and then $T(n) = f(n) + g(n) \in O(h(n))$.

4

3 Nth Power [10 marks]

The following code computes the n-th power of x using repeated squaring.

```
// Calculcate x^n (n is a non-negative integer)
double Pow(double x,int n) {
  double result = 1.0;
  double p = x;
  int i = n;
  while( i > 0 ) {
    // Invariant: result * p^i = x^n
    if( i%2 == 1 ) result *= p;
    p *= p;
    i = i/2; // integer division, so i=floor(i/2)
  }
  return result;
}
```

1. Describe the worst-case time complexity of the function `Pow` using Θ-notation. You do not need to prove that your answer is correct.

2

worst-case: $\Theta(\lg n)$

$T(n) = C + T(n/2)$

$2C + T(n/4)$

$kC + T(n/2^k)$

$\log_2 n = k$

$2^k = n$

$T(1) + C \cdot \log_2 n$

$\Theta(\log_2 n)$

2. Use induction to prove the loop invariant (stated in the code). For the inductive step, consider two cases depending on whether i is even or odd.

2

loop invariant = $(result)(p^i) = x^n$

Base case:

$n = 0 \rightarrow x^0 = 1$

Code returns 1. X

Inductive step, rule holds for $k \leq n$:

Prove for $k = n+1$ → for both odd and even $i = n+1$.

even: even % 2 = 0 thus it skips and calculates $p = p * p$ and then $i = i/2$ → resultant i is still even. keeps on going until we reach $i = 2 \rightarrow 2/2 = 1$ so we hit if statement to return result $*= p$. Since the whole progress is true via induction step, then it is true.

odd: even % 2 = 1 → goes into if to have result $*= p = p$ if result = 1.

For $i = i/2$ → since i is int, truncation occurs and i becomes even. Follow evens flow until $i = 2 \rightarrow 2/2 = 1$ to return result $*= p$. [result = p in this case to account for truncation]

3. Show that if the loop invariant is correct, then the function Pow computes x^n.

If loop invariant is correct, that

result $\cdot p^i = x^n$, then:

By def'n, at the end, when $i = 2 \rightarrow 2/2 = 1$

and we have result $*$ p $\rightarrow$ where p = p * p i times.

Thus we have result $* p^i = x^n$

show i=0.

Details on prev. question.

even runs p = p * p i times, then does result (1.0) $* p^i \rightarrow$ by invariant is equal to x^n

odd runs p = p * p i-1 times, but b/c result is set to be p, when we do result $* p^{i-1} \rightarrow p \cdot p^{i-1} = p^i \rightarrow$ by invariant is equal to x^n

4 k-Way Mergesort [10 marks]

The **Mergesort3** algorithm is like **Mergesort** except that it splits the input array into three (approximately equal sized) parts, rather than just two. It recursively sorts the three parts and then merges the three parts together into one sorted array. For questions 1, 2, and 3, assume that n is a power of 3 and that **Merge3**(B,C,D) takes $|B| + |C| + |D|$ steps (where $|X|$ means the size of array X).

Mergesort3($A[1 \ldots n]$)
  if ($n \leq 1$) return A
  $B =$ **Mergesort3**($A[1 \ldots n/3]$)
  $C =$ **Mergesort3**($A[n/3 + 1 \ldots (2n)/3]$)
  $D =$ **Mergesort3**($A[(2n)/3 + 1 \ldots n]$)
  return **Merge3**(B,C,D)

1. Let $M(n)$ be the number of steps taken by **Mergesort3** on an input of size n. What is a recurrence equation for $M(n)$? (Don't forget the base cases.)

$M(n)$ = steps →

For $M(0)$ and $M(1)$ → return A → c_1

$M(n) = 3M(n/3) + cn$

$= 9M(n/9) + 3 \cdot cn/3 + cn \rightarrow 2cn$

$= 27M(n/27) + 3cn$

$\leq 3^k M(n/3^k) + kcn.$

→ $3^k M(n/3^k) + kcn$

Let $n = 3^k$ → $k = \log_3 n$

→ $n \cdot M(1) + cn \cdot \log_3 n$

$T(n) \in O(n \log_3 n)$

2. What is $M(n)$ as a function of n? Your solution should not be a recurrence or contain a summation.

$M(n) = 3^k \cdot M(n/3^k) + kcn$

$k = \log_3 k \rightarrow n = 3^k$

$= n \cdot M(1) + \log_3 n \cdot cn.$

Did work above by accident.

3. Find the simplest function $g(n)$ such that $M(n) \in \Theta(g(n))$. (You do not need to prove anything.) Is this asymptotic running time (i.e., $\Theta(g(n))$) different from that of regular 2-way **Mergesort** (yes or no)?

Refer to Q1. It is different.

✓ → $\Theta(n \cdot \log_3 n)$ 2-way mergesort is $\Theta(n \log_2 n)$.

The higher the base for log, the smaller the results are. So there exists a small difference.

4. We can imagine splitting the input array into k (approximately equal-sized) parts, for any integer k, and recursively sorting those k parts and merging them together into one sorted array. What is the asymptotic running time of this k-way **Mergesort** assuming that k is a constant (not depending on n)? (You do not need to prove anything.) You may assume that n is a power of k.

running time of K-way mergesort = $\Theta(n \cdot \log_K n)$

✓

5. What is the asymptotic running time of n-way **Mergesort**?

Be careful. You should describe how to perform the n-way **Merge** step of the algorithm. How much time does this step take?

n = # of elements → n-way Mergesort = sort element by element ie comparisons.

Using above def'n : n-way = $\Theta(n \cdot \log_n n)$

✗ $\log_n n = 1$ → $\Theta(n)$

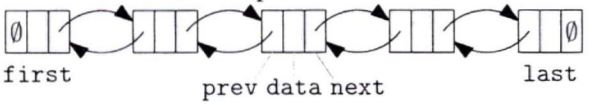
If n-way, then the merge step:

takes in n elements (each elem 0- array) and merges them together based on order given. (sorted)

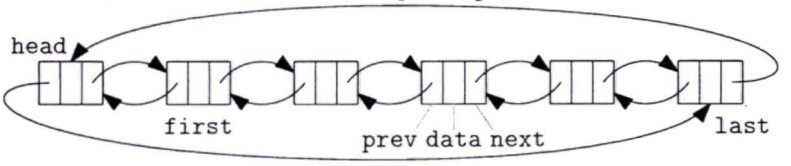
→ takes $\Theta(n)$ time.

5 Circular Doubly-Linked Lists [6 marks]

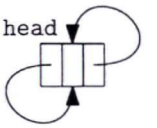
A non-empty regular doubly-linked list has a *first* and *last* node (which might be the same node), where the `prev` pointer of the first node is `NULL` and `next` pointer of the last node is `NULL`.



A **circular** doubly-linked list is a doubly-linked list that contains one special node called the *head* of the list. The `next` pointer of the head points to the first node in the doubly-linked list and the `prev` pointer of the head points to the last node in the doubly-linked list. In addtion, both the `prev` pointer of the first node and the `next` pointer of the last node point to the head. The head node acts as a dummy node that "closes the loop" to make the doubly-linked list circular. An **empty** circular doubly-linked list is just a head node that points to itself with both `next` and `prev` pointers.



A circular doubly-linked list.



An empty circular doubly-linked list.

```
struct Node { int data; Node * prev; Node * next; };
```

Your job is to write code to join two circular doubly-linked lists to create one circular doubly-linked list. The joined list contains all of the nodes from the first list (accessed via `headA`) followed by all of the nodes from the second list (accessed via `headB`) except **it contains only the first head** (`*headA`), which becomes the head of the joined list. You must not create any new nodes. Your code must run in **constant time and space**. (1) Don't forget to `delete headB`. (2) Either list might be empty, but neither `headA` nor `headB` are `NULL`.

```
void join_lists(Node * headA, Node * headB) {
\\ headA and headB are pointers to the head nodes of two
\\ circular doubly-linked lists.
\\ Join the two lists so that *headA is the head of the merged list.

\\ YOUR CODE GOES HERE
need check here
Node * last_A = headA -> prev;


last_A -> next = headB -> next;
head_A -> prev = headB -> prev;


delete headB;
```

6 nevePrint [6 marks]

Write a **recursive** C++ function, `nevePrint`, that given a pointer to the first entry of a singly-linked list prints the data in even positions of the list in reverse order. For example, `nevePrint` should print `8 6 4 2` when given `first` → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → ∅.

Keep your code short, simple, and **recursive**.

```
struct Node { int data; Node *next; };
void nevePrint( Node *first ) {
  Node * curr = first -> next;

  if ( curr -> next != NULL && curr -> next -> next != NULL ) {
    curr = curr -> next -> next;
    nevePrint ( curr );
  }

  std:: cout << curr -> data << " ";
}
```

-1 seg fault

-1 jumping too many nodes

4

7 Short Answers [8 marks]

02092

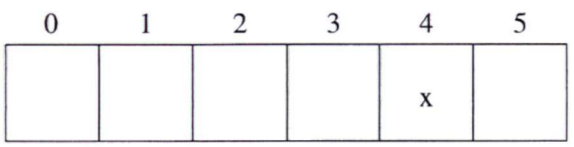
1. Consider the following incomplete statement:

   > Assume algorithm `foo` and algorithm `bar` solve the same problem (so they take the same set of inputs). If `foo` has worst-case running time [A] (n^2) and `bar` has worst-case running time [B] (n^3) on inputs of size n, then `foo` is faster than `bar` on [C] inputs of size n for sufficiently large n.

   By filling in boxes [A], [B], and [C], we can complete the statement and make it True or False. Assume [A] and [B] must be: [O] or [Ω]. Assume [C] is [all] or [some]. List all ways of filling in the boxes that result in the statement being True. Each row that you fill in should specify the three choices that make the statement true.

A	B	C
Ω	Ω	all

2. Assume that the ADT Queue is implemented using a circular array with size 6. The current state of this data structure is as follows:



`front = 4 and back = 5`

Assume enque at front and deque at back

- enque Arr[front+1]
- deque Arr[back-1]

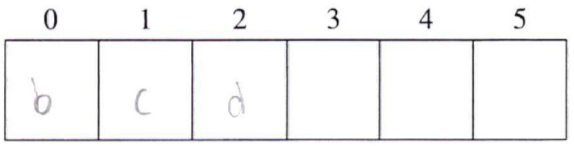
Note that this means that the queue contains one element "x".

Draw the final state of the data structure after the following sequence of operations:

```
enqueue(a); enqueue(b); enqueue(c); deque(); deque(); enqueue(d);
```

x a x a b x a b c a b c b c b c d

Final state:



`front = 2 and back = 1`

1 2 3 4 5
1 1 2 3 5

02092

3. Recall the first version of `fib` function from the lecture:

```
  int fib(int n) {
1.   if (n <= 2) return 1;
2.   int a = fib(n-1);
3.   int b = fib(n-2);
4.   return a+b;
  }
```

n = 3 2 1
a = fib(4) → fib(3) + fib(2)
b = fib(3) →

At one point in the execution of `fib(5)`, the call stack looks like this:

Fill in the missing line! ⟶

Line 1, n = 2, return 1
Line 2, n: 3, a=fib(2), b=fib(1)
Line 3, n = 5, a = 3, b = fib(3)

4. Consider a d-heap with n elements. Circle the correct answers below:

- The height of this d-heap is approximately:

  A. $\lg n$ B. $\lg d$ C. $\log_d n$ D. $d \lg n$

- Assume the height of this d-heap is H. The `swapDown` function will take at most:

  A. H B. $H \lg d$ C. dH D. $d + H$ steps.

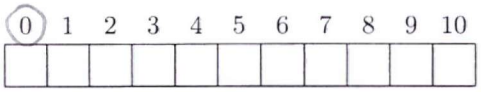
unique elem.

replace heap 7/41

02092

8 Two-dimensional Heaps [10 marks]

1. The following figure represents an array that stores a binary Min-Heap with 11 elements. I haven't shown the keys for these elements.

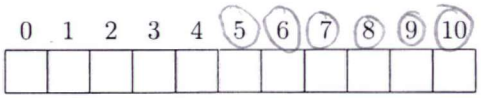


2

Assume follows property of heap order.

Circle the locations where the element with the **minimum** key in the heap might lie.

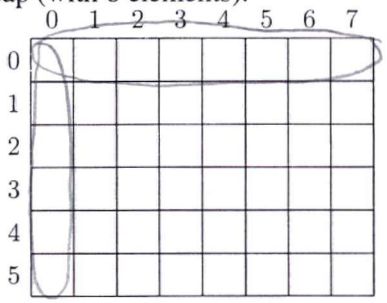
2. The following figure represents an array that stores a binary Min-Heap with 11 elements. I haven't shown the keys for these elements.



2

Circle the locations where the element with the **maximum** key in the heap might lie.

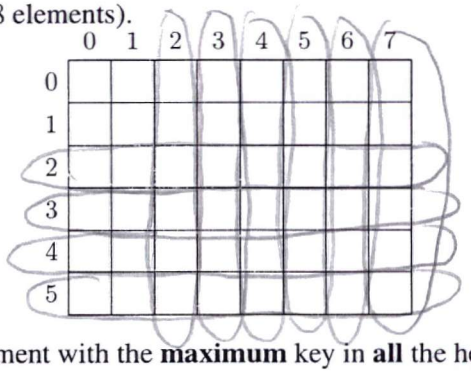
3. The following figure represents a two-dimensional array in which every row and every column is a binary Min-Heap. Notice that each element in the two-dimensional array is a member of two heaps: its column heap (with 6 elements) and its row heap (with 8 elements).



1

Circle the locations where the element with the **minimum** key in **all** the heaps might lie.

4. The figure represents a two-dimensional array in which every row and every column is a binary Min-Heap. Notice that each element in the two-dimensional array is a member of two heaps: its column heap (with 6 elements) and its row heap (with 8 elements).



1

Circle the locations where the element with the **maximum** key in **all** the heaps might lie.

5. Is  the only two-dimensional array with keys 1, 2, 3, and 4 in which every row and every column is a binary Min-Heap? Yes or No.

1 2 3
3 4 4

No. Can have

2

1
2

1 1 2 3
2 3 4 4

Still works.

This page intentionally left blank.

If you put solutions here or anywhere other than the blank provided for each solution, you must *clearly* indicate which problem the solution goes with and also indicate where the solution is at the designated area for that problem's solution.

This page intentionally left blank.

If you put solutions here or anywhere other than the blank provided for each solution, you must *clearly* indicate which problem the solution goes with and also indicate where the solution is at the designated area for that problem's solution.