National University of Singapore

CS2040S—Data Structures and Algorithms

MIDTERM ASSESSMENT

AY2024/2025, Semester 2

Time allowed: 1 hour 45 mins

INSTRUCTIONS TO STUDENTS

1. Do not open the midterm until you are directed to do so.
2. Read **all** the instructions first.
3. The quiz is closed book. You may bring one double-sided sheet of A4 paper to the quiz. (You must not bring any magnification equipment!) You must NOT use a calculator, your mobile phone, or any other electronic device.
4. The assessment paper contains **SEVEN (7) questions** and comprises **TWELVE (12) pages** including this cover page.
5. The time allowed for solving this test is **1 hour 45 mins**.
6. The maximum score of this test is **60 marks**. The weight of each question is given in square brackets beside the question number.
7. All questions must be answered in the space provided in the **ANSWER SHEET**; no extra sheets will be accepted as answers.
8. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
9. The questions are provided in the question booklet.
10. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
11. Unless otherwise stated in the question, when we ask for the worst-case big-O running time of an algorithm we always mean to give the tightest possible answer.
12. Unless otherwise stated in the question, we will assume that operators behave as per Java (e.g., 5/2 will evaluate to 2). However, pseudocode does not necessarily satisfy Java syntax (unless stated otherwise) and things that do not compile as legal Java are not necessarily bugs (as long as they are clear pseudocode).
13. Unless otherwise stated in the question, we are not concerned with overflow errors, e.g., storing the value $2^{245,546,983}$ in an integer.
14. Pseudo-code is fine, we are not concerned with Java compilability. By default, we are using 1-based indexing, **when given the option, please state explicitly if you are using 0-based instead**.
15. When considering efficiency, we will **consider worst case running time** unless otherwise stated.

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: Basic Bounds [6 marks]

For each of the following functions $T(n)$, pick the function f_i with the smallest index i among the following options, for which $T(n) = O(f_i(n))$:

1. $f_1(n) = O(1)$
2. $f_2(n) = \log(\log(n))$
3. $f_3(n) = \log^{100}(n)$
4. $f_4(n) = \sqrt{n}$
5. $f_5(n) = n$
6. $f_6(n) = n\log(n)$

For example: If you think that the function $T(n)$ is $O(\log(\log(n)))$ but not $O(1)$, then pick $O(\log(\log(n)))$.

A.

$$T(n) = 3\log(n^{10}) + \log^3(n)$$

[2 marks]

B.

$$T(n) = 2^{\log(n)+100} + n^{\frac{300}{400}}$$

[2 marks]

C.

$$T(n) = \sum_{i=1}^{\log(n)} i$$

[2 marks]

Question 2: To bound recurrences you must know how to bound recurrences... [9 marks]

For each of the following functions $T(n)$, pick the function f_i with the smallest index i among the following options, for which $T(n) = O(f_i(n))$:

1. $f_1(n) = O(1)$
2. $f_3(n) = \log^{100}(n)$
3. $f_4(n) = \sqrt{n}$
4. $f_5(n) = n$
5. $f_6(n) = n\log(n)$
6. $f_6(n) = 2^n$

For example: If you think that the function $T(n)$ is $O(\log(\log(n)))$ but not $O(1)$, then pick $O(\log(\log(n)))$.

A.

$$T(n) = \begin{cases} T(n-1) + T(n-1) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

[3 marks]

B.

$$T(n) = \begin{cases} T(n/2) + \log(n) & n \geq 2 \\ 1 & n = 1 \end{cases}$$

[3 marks]

C.

$$T(n) = \begin{cases} T(n-1) + n^2 & n \geq 2 \\ 1 & n = 1 \end{cases}$$

[3 marks]

Question 3: Peak Sorting [9 marks]

You are given as input an array $A[1 \ldots n]$ of n **distinct** integers. The array is guaranteed to be in such a way that the integers are arranged in a "$\wedge$" shape. I.e. there exists an index $1 < i < n$ for which:

1. $\forall 1 \leq j < i (A[j] < A[i])$
2. $\forall i < j \leq n (A[j] > A[i])$

An example of such an array is as follows:

$$[1, 5, 7, 10, 11, 12, 20, 9, 8, 3, 2]$$

A. Describe an algorithm that sorts these types of arrays as efficiently (in terms of worst-case asymptotic running time) as possible. Pseudo-code is fine, and a high level description of the algorithm will suffice. You need not write the actual for loops and so forth. Just a few lines description of your algorithm should suffice.

If you wish to use 0-based indexing, please state explicitly that you are doing so.

[7 marks]

B. Give the tightest possible running time bound of your algorithm. [2 marks]

Question 4: Invariants [8 marks]

Given an infinitely large, and sorted array A, we wish to find out whether a certain value k is in the array or not. Here is an algorithm that does it in $O(\log(n))$ time, where n is either the position of the first value that is $\geq k$. You may assume that `BinarySearch(A, ℓ, r, k)` returns `true` if k exists in $A[\ell..r]$ (inclusive).

We are assuming 1-based indexing.

```
1  FindValue(A, target){
2      if(A[1] >= target) {
3          return (A[1] == target)
4      }
5      upper_bound_idx = 1
6      while(A[upper_bound_idx] < target){
7          upper_bound_idx = upper_bound_idx * 2
8      }
9
10     lower_bound_idx = upper_bound_idx / 2
11     while(lower_bound_idx + 1 < upper_bound_idx) {
12         mid = (upper_bound_idx - lower_bound_idx) / 2 + lower_bound_idx
13         if(A[mid] < target) {
14             lower_bound_idx = mid
15         } else {
16             upper_bound_idx = mid
17         }
18     }
19     return (A[upper_bound_idx] == target)
```

State for the following invariants, whether it is **True** or **False**.

At the end of the j^{th} iteration (line 7) for the loop (on line 6):

A. For all values i such that $i < upper\_bound\_idx$ it holds that $A[i] < target$. [1 mark]

B. For all values i such that $i < upper\_bound\_idx/2$ it holds that $A[i] < target$. [1 mark]

C. Let t be the smallest index for which $A[t] > target$. Then $upper\_bound\_idx < t$. [1 mark]

D. It holds that $upper\_bound\_idx/2 < target$. [1 mark]

At the end of the j^{th} iteration (line 12-17) for the loop at line 11:

E. For all values $i < lower\_bound\_idx$, $A[i] < target$. [1 mark]

F. For all values $i < upper\_bound\_idx$, $A[i] < target$. [1 mark]

G. For all values $lower\_bound\_idx \leq i < upper\_bound\_idx$, $A[i] < target$. [1 mark]

H. $A[lower\_bound\_idx] < target$. [1 mark]

Question 5: Mountain Climbing [6 marks]

You are given an array $A[1..n]$ of $n \geq 2$ integers where adjacent elements differ by value 1. I.e. $\forall i[|A[i]-|A[i+1]|| \leq 1$.

And are promised that the first value is smaller than the last value. i.e. $A[1] < A[n]$.

Here is a snippet of some code that given some value k such that $A[1] < k$ and $k \leq A[n]$, outputs an index i such that $A[i] = k$.

We are assuming 1-based indexing.

```
1  BinarySearch(A, k){
2      int low = 1;
3      int high = n;
4      while(low + 1 < high) {
5          int mid = (high - low) / 2 + low;
6          if(A[mid] < k) {
7              /* snippet X */
8          } else {
9              /* snippet Y */
10         }
11     }
12     return /* snippet Z */
13 }
```

Hint: Notice that we are starting off with $A[1] > A[n]$. First think about what kind of invariant we can use, and how we should maintain it.

A. Write the code that belongs as `Snippet X` (on line 7.). [2 marks]

B. Write the code that belongs as `Snippet Y` (on line 9.). [2 marks]

C. Write the code that belongs as `Snippet Z` (on line 12.). [2 marks]

Question 6: Middle of the Pack [22 marks]

Nodle wants to adopt huskies but is quite particular about their size. A husky can be represented by a pair of integers (i, s_i), where i is the unique ID for the husky, and s_i is the size of the Husky.

Every time Nodle sees a Husky, he wants to keep track of its size, and every once in a while, he wants to know the IDs of the huskies with the median size, or max size, or minimum size.

Here is an example scenario:

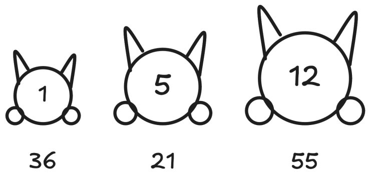

Figure 1: Added Husky with sizes 1, 5, and 12

Then, the minimum-sized husky has ID 36. The median-sized husky has ID 21. Then the maximum-size husky has ID 55.

New let's say that we add two new Huskies: 44 and 61.

Husky 44 has size 20, and Husky 61 has size 10.

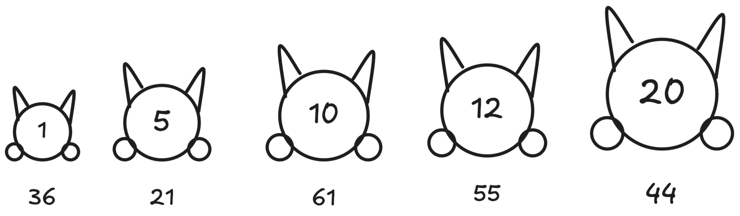

Figure 2: Two more huskies.

Then, the minimum-sized husky has ID 36. The median-sized husky has ID 61. Then the maximum-size husky has ID 44.

Your goal is to help Nodle design a data structure that supports the following operations:

1. `AddHusky`(i, s_i): Add a husky i that has size s_i.
2. `MedianHusky`(): Return the id of the husky with the median size.
3. `MinimumHusky`(): Return the id of the husky with the minimum size.
4. `MaximumHusky`(): Return the id of the husky with the maximum size.

So based on the example, here is an example sequence of calls:

1. `AddHusky`(36, 1)
2. `AddHusky`(21, 5)
3. `AddHusky`(55, 21)

4. `MinHusky`() $\rightarrow$ 36
5. `MedianHusky`() $\rightarrow$ 21
6. `MaxHusky`() $\rightarrow$ 55
7. `AddHusky`(61, 10)
8. `AddHusky`(44, 20)
9. `MinHusky`() $\rightarrow$ 36
10. `MedianHusky`() $\rightarrow$ 61
11. `MaxHusky`() $\rightarrow$ 44

You may assume that:

1. The husky sizes are unique integer values.
2. We will only find the median value when there is an odd number of huskies added.

A. Describe the data structure. Please specify any variables or data structures (e.g. trees, arrays, linked lists, stacks, queues) you might need.

Note: If you are using an AVL tree, it will be assumed that it is augmented to support height-balancing, and get min/max, lookup, delete, successor/predecessor operations are available in worst case $O(\log(n))$ time for a tree of size n. It is not assumed that the tree is able to support rank/select operations. [3 marks]

B. Describe your algorithm (in pseudocode) for `AddHusky`(i, s_i). [2 marks]

C. What is the running time of your algorithm for `AddHusky`? [1 mark]

D. Describe your algorithm (in pseudocode) for `MinHusky`(). [1 mark]

E. What is the running time of your algorithm for `MinHusky`? [1 mark]

F. Describe your algorithm (in pseudocode) for `MaxHusky`(). [1 mark]

G. What is the running time of your algorithm for `MaxHusky`? [1 mark]

H. Describe your algorithm (in pseudocode) for `MedianHusky`(). [1 mark]

I. What is the running time of your algorithm for `MedianHusky`? [1 mark]

As an extension, the Huskies now come in either a **blue** coat, or a **grey** coat. Nodle wants to know, for a certain size: what colour is the Husky, and how many huskies of the opposite colour are smaller than it.

- `CountLighter(`s`)`: Returns the colour of the husky of that size, and how many huskies with the opposite coat that is smaller than s.

J. Describe an augmentation for your data structure from part **A.** to handle this new query operation. [2 marks]

K. Describe (in pseudocode) how to maintain the augmented data during inserts, and AVL rotations (if any). [3 marks]

L. What is the running time for your new implementation of `AddHusky`? [1 mark]

M. Describe (in pseudocode) $CountLighter(s)$ algorithm. High level ideas are enough. [3 marks]

N. What is the running time for your new implementation of `CountLighter`? [1 mark]

Question 7: Bonus Question, Optional: [0 marks]

A man is stuck in a perfectly circular arena with a lion. The man can move as fast as the lion. Is it possible for the man to survive? (Assume each has infinite strength so they can both continue to move indefinitely if needed)? [0 mark]

— END OF PAPER —

CS2040S—Data Structures and Algorithms

Sample Midterm Assessment Answer Sheet

AY2024/2025, Semester 2

Time allowed: 1 hour 45 mins

Instructions (please read carefully):

1. Write down your **student number** on the right and using ink or pencil, shade the corresponding circle in the grid for each digit or letter. DO NOT WRITE YOUR NAME!
2. This answer booklet comprises **EIGHT (8) pages**, including this cover page, not including the blank page.
3. All questions must be answered in the space provided; no extra sheets will be accepted as answers. You may use the extra page behind this cover page if you need more space for your answers.
4. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
5. The questions are provided in the question booklet, and if there is any inconsistency in the question, please refer to the question booklet. If there is any inconsistency in the answers, refer to the answer sheet.
6. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
7. Please fill in the bubbles properly. **Marks may be deducted** for unclear answers.

STUDENT NUMBER

A									
U ○	0	0	0	0	0	0	0	A	N
A ●	1	1	1	1	1	1	1	B	R
HT ○	2	2	2	2	2	2	2	E	U
NT ○	3	3	3	3	3	3	3	H	W
	4	4	4	4	4	4	4	J	X
	5	5	5	5	5	5	5	L	Y
	6	6	6	6	6	6	6	M	
	7	7	7	7	7	7	7		
	8	8	8	8	8	8	8		
	9	9	9	9	9	9	9		

For Examiner's Use Only

Question	Marks
Q1	/ 6
Q2	/ 9
Q3	/ 9
Q4	/ 8
Q5	/ 6
Q6	/ 22
Q7	/ 0
Total	/ 60

Question 1A [2 marks]

○ $O(1)$	○ $O(\log(\log(n)))$	○ $O(\log^{100}(n))$
○ $O(\sqrt{n})$	○ $O(n)$	○ $O(n\log(n))$

Question 1B [2 marks]

○ $O(1)$	○ $O(\log(\log(n)))$	○ $O(\log^{100}(n))$
○ $O(\sqrt{n})$	○ $O(n)$	○ $O(n\log(n))$

Question 1C [2 marks]

○ $O(1)$	○ $O(\log(\log(n)))$	○ $O(\log^{100}(n))$
○ $O(\sqrt{n})$	○ $O(n)$	○ $O(n\log(n))$

Question 2A [3 marks]

○ $O(1)$	○ $O(\log^{100}(n))$	○ $O(\sqrt{n})$
○ $O(n)$	○ $O(n\log(n))$	○ $O(2^n)$

Question 2B [3 marks]

○ $O(1)$	○ $O(\log^{100}(n))$	○ $O(\sqrt{n})$
○ $O(n)$	○ $O(n\log(n))$	○ $O(2^n)$

Question 2C [3 marks]

○ $O(1)$	○ $O(\log^{100}(n))$	○ $O(\sqrt{n})$
○ $O(n)$	○ $O(n\log(n))$	○ $O(2^n)$

Question 3A Describe sorting algorithm. [7 marks]

```
void PeakSorting(int[] arr, int n){
    /* Your pseudocode here */
    
}
```

Question 3B Tightest possible running time bound. [2 marks]

Question 4A [1 marks]

○ True ○ False

Question 4B [1 marks]

○ True ○ False

Question 4C [1 marks]

○ True ○ False

Question 4D [1 marks]

○ True ○ False

Question 4E [1 marks]

○ True ○ False

Question 4F [1 marks]

○ True ○ False

Question 4G [1 marks]

○ True ○ False

Question 4H [1 marks]

○ True ○ False

Question 5A Snippet X (on line 7.) [2 marks]

Question 5B Snippet Y (on line 9.) [2 marks]

Question 5C Snippet Z (on line 12.) [2 marks]

Question 6A Declare your variable or data structures. [3 marks]

```
class DataStructure {
     // Declare any data structures or variables here








}
```

Question 6B Describe your algorithm (in pseudocode) for AddHusky [2 marks]

```
class DataStructure {
    void AddHusky(int id, int size){
        /* Your pseudocode here */






    }
}
```

Question 6C What is the running time for your implementation of AddHusky? [1 marks]

Question 6D Describe your algorithm (in pseudocode) for MinHusky [1 marks]

```
class DataStructure {
    void MinHusky(){
        /* Your pseudocode here */





    }
}
```

Question 6E What is the running time for your implementation of MinHusky? [1 marks]

Question 6F Describe your algorithm (in pseudocode) for MaxHusky [1 marks]

```
class DataStructure {
    void MaxHusky(){
        /* Your pseudocode here */





    }
}
```

Question 6G What is the running time for your implementation of MaxHusky? [1 marks]

Question 6H Describe your algorithm (in pseudocode) for MedianHusky [1 marks]

```
class DataStructure {
    void MedianHusky(){
        /* Your pseudocode here */





    }
}
```

Question 6I What is the running time for your implementation of MedianHusky?[1 marks]

Question 6J Describe an augmentation for your data structure from part A to handle this new query operation. [2 marks]

Question 6K Describe how to maintain the augmented data during inserts, and AVL rotations (if any). [3 marks]

Question 6L What is the running time for your new implementation of AddHusky? [1 marks]

Question 6M Describe (in pseudocode) CountLighter(s) algorithm. High level ideas are enough. [3 marks]

Question 6N What is the running time for your new implementation of CountLighter? [1 marks]

Question 7 [0 marks]

This page is intentionally left blank.

It may be used as scratch paper.

This page is intentionally left blank.

It may be used as scratch paper.

— END OF ANSWER SHEET —

Question 1A [2 marks]

○ $O(1)$	○ $O(\log(\log(n)))$	[✓ $O(\log^{100}(n))$]
○ $O(\sqrt{n})$	○ $O(n)$	○ $O(n\log(n))$

Question 1B [2 marks]

○ $O(1)$	○ $O(\log(\log(n)))$	○ $O(\log^{100}(n))$
○ $O(\sqrt{n})$	[✓ $O(n)$]	○ $O(n\log(n))$

Question 1C [2 marks]

○ $O(1)$	○ $O(\log(\log(n)))$	[✓ $O(\log^{100}(n))$]
○ $O(\sqrt{n})$	○ $O(n)$	○ $O(n\log(n))$

Question 2A [3 marks]

○ $O(1)$	○ $O(\log^{100}(n))$	○ $O(\sqrt{n})$
○ $O(n)$	○ $O(n\log(n))$	[✓ $O(2^n)$]

Question 2B [3 marks]

○ $O(1)$	[✓ $O(\log^{100}(n))$]	○ $O(\sqrt{n})$
○ $O(n)$	○ $O(n\log(n))$	○ $O(2^n)$

Question 2C [3 marks]

○ $O(1)$	○ $O(\log^{100}(n))$	○ $O(\sqrt{n})$
○ $O(n)$	○ $O(n\log(n))$	[✓ $O(2^n)$]

Question 3A Describe sorting algorithm. [7 marks]

Here's an example high level description of the optimal algorithm:

```
void PeakSorting(int[] arr, int n){
    1. Find the index j that is larger than its neighbours.
        This can be done with a simple loop to check each element.
    2. Copy out the first j elements into an array.
        Copy out the remaining n - j elements into another array,
        and then reverse it.
    3. Run the merge step on the two arrays.
}
```

Partial marks also can be awarded if certain steps are seen. Depending on the algorithm itself.

Question 3B Tightest possible running time bound. [2 marks]

$O(n)$

Question 4A [1 marks]

True [✓ False]

Question 4B [1 marks]

[✓ True] False

Question 4C [1 marks]

True [✓ False]

Question 4D [1 marks]

True [✓ False]

Question 4E [1 marks]

[✓ True] False

Question 4F [1 marks]

True [✓ False]

Question 4G [1 marks]

True [✓ False]

Question 4H [1 marks]

[✓ True] False

Question 5A Snippet X (on line 7.) [2 marks]

low = mid

Question 5B Snippet Y (on line 9.) [2 marks]

high = mid

Question 5C Snippet Z (on line 12.) [2 marks]

high

Question 6A Declare your variable or data structures. [3 marks]

```
class DataStructure {
    // Declare any data structures or variables here
    two AVL trees, each is keyed on husky sizes and maps to
    husky IDs as values.

    call the first tree as small_tree, the second tree as large_tree.
}
```

Question 6B Describe your algorithm (in pseudocode) for AddHusky [2 marks]

```
class DataStructure {
    void AddHusky(int id, int size){
        /* Your pseudocode here */
        if the two trees are the same size,
        then insert the husky into the small_tree
        then remove the largest husky from small_tree,
        and insert the removed husky into the large_tree.

        else, if small_tree has one more husky than the large_tree,
        then insert the husky into the large_tree
        then remove the smallest husky from the large_tree,
        and insert the removed husky into the small_tree.
    }
}
```

Question 6C What is the running time for your implementation of AddHusky? [1 marks]

$O(\log(n))$ time for inserting and deleting from an AVL tree twice.

Question 6D Describe your algorithm (in pseudocode) for MinHusky [1 marks]

```
class DataStructure {
    void MinHusky(){
        /* Your pseudocode here */

        return the smallest husky from small_tree


    }
}
```

Question 6E What is the running time for your implementation of MinHusky? [1 marks]

$O(\log(n))$ time for obtaining the minimum Husky.

Question 6F Describe your algorithm (in pseudocode) for MaxHusky [1 marks]

```
class DataStructure {
    void MaxHusky(){
        /* Your pseudocode here */

        return the largest husky from large_tree


    }
}
```

Question 6G What is the running time for your implementation of MaxHusky? [1 marks]

$O(\log(n))$ time for obtaining the maximum Husky.

Question 6H Describe your algorithm (in pseudocode) for MedianHusky [1 marks]

```
class DataStructure {
    void MedianHusky(){
        /* Your pseudocode here */

        return the largest husky from small_tree


    }
}
```

Question 6I What is the running time for your implementation of MedianHusky?[1 marks]

$O(\log(n))$ time for obtaining the maximum Husky from small_tree.

Question 6J Describe an augmentation for your data structure from part A to handle this new query operation. [2 marks]

Augment each node to also store: (1) the colour of the coat of the husky at this node, (2) the count of huskies of grey coats in the subtree rooted at this node (2) the count of huskies of blue coats in the subtree rooted at this node

Question 6K Describe how to maintain the augmented data during inserts, and AVL rotations (if any). [3 marks]

When adding a husky into the tree, the node is initially a leaf node. If the husky being inserted as a grey coat, set the colour to grey, otherwise, to blue. Similarly, set the count of the grey coats to be 1 if the husky being inserted has a grey coat, 0 otherwise. Also, set the count of the blue coats to be 1 if the husky being inserted has a blue coat, 0 otherwise.

Then walk back up to the root node, and each both counts are updated similarly as the sum of the left counts + right counts. Also if the husky is grey, + 1 to the grey counts, otherwise + 1 to the blue counts.

The same formula applies during AVL rotations.

Question 6L What is the running time for your new implementation of AddHusky? [1 marks]

$O(\log(n))$

Question 6M Describe (in pseudocode) CountLighter(s) algorithm. High level ideas are enough. [3 marks]

The algorithm for both colours work similarly. Find the husky of size s in either tree first to find its color, call it c. Call the opposite colour d. Then, we need to count how many huskies have the opposite colour in both trees that have size $< s$.

To do this, firstly, maintain a counter called $total = 0$. Start from the node s, and add the total count of d-coloured huskies in the left subtree of s to $total$. Then as we walk up to the root node, if we were on the right child of the parent node p, add the total count of d-coloured huskies in the left subtree of p to $total$. Then set p as our current node, and repeat.

Question 6N What is the running time for your new implementation of CountLighter? [1 marks]

$O(\log(n))$

Question 7 [0 marks]