CS2040S — Data Structures and Algorithms
School of Computing
National University of Singapore

Midterm Assessment

03 Mar 2025 **Time allowed:** 1.5 hours

Instructions — please read carefully:

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 may not bring any magnification equipment!) You may NOT use a calculator, your mobile phone, or any other electronic device.
4. The **QUESTION SET** comprises **SIX (6) questions** and **NINE (9) pages**, and the **ANSWER SHEET** comprises of **TEN (10) pages**.
5. The time allowed for solving this test is **1.5 hours**.
6. The maximum score of this test is **33 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, 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.
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**.

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: Basic Bounds [3 marks]

Give the tightest possible upper-bound in terms of Big-Oh for the following recurrences:

A.

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

. [1 mark]

B.

$$T(n) = 2^{\log(n)+100} + n^{\frac{5}{4}}$$

. [1 mark]

C.

$$T(n) = 2^{\log^2(n)} + (\log(n))^{\log(n)}$$

. [1 mark]

Question 2: To bound recurrences you must know how to bound recurrences... [3 marks]

Give the tightest possible upper-bound in terms of Big-Oh for the following recurrences:

A.

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

[1 mark]

B.

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

[1 mark]

C.

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

[1 mark]

Question 3: Peak Sorting [4 marks]

You are given as input an array $A[1\dots n]$ 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. $\forall 1 \le j < i (A[j] < A[i])$
2. $\forall i < j \le n (A[j] < A[i])$

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 asymptotic runnning time) as possible. **If you wish to use 0-based indexing, please state explicitly that you are doing so.** [3 marks]

B. Give the tightest possible running time bound of your algorithm. [1 mark]

Question 4: Invariants [4 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.

```
FindValue(A, target){
if(A[1] == target) {
    return true
}
upper_bound_idx = 1
while(A[upper_bound_idx] < target){
    upper_bound_idx = upper_bound_idx * 2
}

lower_bound_idx = upper_bound_idx / 2
while(lower_bound_idx + 1 < upper_bound_idx) {
    mid = (upper_bound_idx - lower_bound_idx) / 2 + lower_bound_idx
    if(A[mid] < target) {
        lower_bound_idx = mid
    } else {
        upper_bound_idx = mid
    }
}
return (A[upper_bound_idx] == target)
```

(i) at the **beginning** of any iteration, A[upper_bound_idx] < target

(ii) at the **end** of any iteration, A[upper_bound_idx] < target

(iii) at the **beginning** of any iteration, all elements x in A[1..upper_bound_idx / 2] are such that $x < target$

(iv) at the **end** of any iteration, all elements x in A[1..upper_bound_idx / 2] are such that $x < target$

(A) i only.

(B) ii only.

(C) i, iii only.

(D) i, iii, iv only.

(E) i, ii, iii, iv.

(F) None of the above options A, B, C, D, nor E are correct.

A. Which of the following options above are invariants for the loop for line 6? An iteration of this loop comprises line(s) 7. [2 marks]

(i) At every iteration, A[low] < target

(ii) At every iteration, A[low] $\leq$ target

(iii) At every iteration, A[high] > target

(iv) At every iteration, A[low] $\geq$ target

(A) i, iii only.

(B) i, iv only.

(C) ii, iii only.

(D) ii, iv only.

(E) i, ii, iii, iv.

(F) None of the above options A, B, C, D, are correct.

B. Which of the options above are invariants for the loop for line 11? An iteration of this loop comprises line(s) 12-17. [2 marks]

Question 5: Mountain Climbing [6 marks]

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

And are promised that the first value is smaller than the second 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 $k < A[n]$, outputs an index i such that $A[i] = k$.

We are assuming 1-based indexing.

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

Hint: Think about what `low`, `high` represent, and our termination condition is stated. Based on that, how should we update the variables, and what should we return?

A. Write the single line of code that belongs as `Snippet X` (on line 6.). [2 marks]

B. Write the single line of code that belongs as `Snippet Y` (on line 8.). [2 marks]

C. Write the single line of code that belongs as `Snippet Z` (on line 11.). [2 marks]

Question 6: Middle of the Pack [13 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.

Your goal is to help Nodle maintain 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.

We are assuming 1-based indexing. Please state explicitly if you are using 0-based indexing in your code.

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.
3. The balanced binary search tree used is an AVL tree that is **not** augmented with weights. As a consequence, the binary search tree used only provides insert, delete, lookup, search, findMin, findMax operations.

Example 1

1. `AddHusky`(1, 25)
2. `MedianHusky`() $\rightarrow$ 1
3. `MinHusky`() $\rightarrow$ 1
4. `MaxHusky`() $\rightarrow$ 1

Explanation: There is only one husky, and so it is the median, minimum, and maximum sized husky.

Example 2

1. `AddHusky`(2, 25)
2. `MedianHusky`() $\rightarrow$ 2
3. `AddHusky`(1, 50)
4. `MinHusky`() $\rightarrow$ 2
5. `MaxHusky`() $\rightarrow$ 1
6. `AddHusky`(3, 75)
7. `MedianHusky`() $\rightarrow$ 1

Explanation: Initially there is only one husky (the husky with id 2) so it is the median value. Then, after we add another husky (with ID 1), the minimum sized husky is of ID 2, whereas the maximum sized husky is of ID 1. Lastly, after the third husky is added, there are 3 huskies, $[25, 50, 75]$. The median husky is of ID 1 and has size 50.

A. Write down any variables or ADTs your data structure might need to use. If you are using a tree, state explicitly what is the type of the key, and what is the type of the value (if any). [1 mark]

B. Implement the `AddHusky`(i, s_i) method. [3 marks]

C. Implement the `MedianHusky`() method. [3 marks]

D. Implement the `MinHusky`() method. [1 mark]

E. Implement the `MaxHusky`() method. [1 mark]

F. What is the running time for your implementation of `AddHusky`(i, s_i)? [1 mark]

G. What is the running time for your implementation of `MedianHusky`()? [1 mark]

H. What is the running time for your implementation of `MinHusky`(i, s_i)? [1 mark]

I. What is the running time for your implementation of `MaxHusky`()? [1 mark]

— E N D   O F   P A P E R —

CS2040S — Algorithms and Data Structures
School of Computing
National University of Singapore

Midterm Assessment — Answer Sheet

2024/2025 Semester 2

Time allowed: 1.5 hours

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 **TEN (10) pages**, including this cover 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	/ 3
Q2	/ 3
Q3	/ 4
Q4	/ 4
Q5	/ 6
Q6	/ 13
Total	/ 33

This page is intentionally left blank.
Use it ONLY if you need extra space for your answers, and indicate the **question number clearly** as well as in the original answer box. **Do NOT** use it for your rough work.

Question 1A
[1 marks]

Question 1B
[1 marks]

Question 1C
[1 marks]

Question 2A
[1 marks]

Question 2B
[1 marks]

Question 2C
[1 marks]

Question 3A [3 marks]

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








}
```

Question 3B [1 marks]

Question 4A [2 marks]

Question 4B [2 marks]

Question 5A [2 marks]

Question 5B [2 marks]

Question 5C [2 marks]

Question 6A [1 marks]

Example solution 1:

```
    class DataStructure {
        AVL<int> small_tree;
        AVL<int> large_tree;
    }
```

Example solution 2:

```
    class DataStructure {
        An AVL tree storing integers as keys, integers as values, called
           small_tree;
        An AVL tree storing integers as keys, integers as values, called
           large_tree;
    }
```

Example solution 3:

```
    class DataStructure {
        An AVL tree storing Husky sizes as keys, Husky IDs as values, called
           small_tree;
        An AVL tree storing Husky sizes as keys, Husky IDs as values, called
           large_tree;
    }
```

Either solution is acceptable for us, solution 3 is a little clearer than solution 2. Here is an example of a solution that is unclear and risks losing marks:

Negative Example solution 3:

```
    class DataStructure {
        An AVL Tree storing huskies. // What does it mean to store a husky? What
           is the key?
        An AVL Tree that stores the median values. // What does it mean to do
           this?
        AVL tree // Again, it is unclear to us what your keys and values are.
           Graders will have to try to infer across your other answers.
    }
```

Question 6B [3 marks]

Example Solution 1:

```
class DataStructure {
    void AddHusky(int id, int size) {
        if(small_tree.size() == large_tree.size()){
            large_tree.insert(size, id);
            int smallest_id = large_tree.get_minimum_value();
            int smallest_size = large_tree.get_minimum_key();
            small_tree.insert(smallest_size, smallest_id);
        } else {
            small_tree.insert(size, id);
            int largest_id = small_tree.get_minimum_value();
            int largest_size = small_tree.get_minimum_key();
            large_tree.insert(largest_size, largest_id);
        }
    }
}
```

Example Solution 2:

```
class DataStructure {
    void AddHusky(int id, int size) {
        if both trees are the same size, insert size as key, mapping to id as
            value into large_tree.
            then get and remove the key value pair (with smallest key) out of the
                 large_tree, and insert it into the small_tree.

        otherwise the trees are not the same size, insert size as key, mapping to
             id as value into small_tree,
            then get and remove the key value pair (with largest key) out of the
                small_tree, and insert it into the large_tree.

    }
}
```

Both solutions are acceptable. To the reader we know what is being done and we can clearly picture what is the change being made to the data structures. Here is a negative example of an unclear solution.

Negative Example Solution:

```
class DataStructure {
    void AddHusky(int id, int size) {
        insert the husky into the tree, if the tree is bigger than the other tree
            , otherwise insert the husky into the other tree.
    }
}
```

In the negative example it is very unclear on what is going on and the grader is unable to tell what change is being made to the data structures, what is being inserted, and so on.

Question 6C

[3 marks]

Example Solution 1:

```
class DataStructure {
    void AddHusky(int id, int size) {
        return small_tree.max_value(); // returns the value of the maximum key
            stored in the small_tree
    }
}
```

Example Solution 2:

```
class DataStructure {
    void AddHusky(int id, int size) {
        get the id of the largest husky stored in small_tree, return it
    }
}
```

Both solutions are acceptable. Notice here that again, the grader can clearly tell what is being returned.

Negative Example Solution:

```
class DataStructure {
    void AddHusky(int id, int size) {
        just return the median
    }
}
```

Based on our example so far and what we have stored, this is unclear on where or how we obtain this value.

Question 6D

[1 marks]

```
    class DataStructure {
        int MinHusky() {
            /* Your (pseudo)code here */













        }
    }
```

Question 6E

[1 marks]

```
    class DataStructure {
        int MaxHusky() {
            /* Your (pseudo)code here */













        }
    }
```

Question 6F

[1 marks]

Question 6G [1 marks]

Question 6H [1 marks]

Question 6I [1 marks]

This page is intentionally left blank.
Use it ONLY if you need extra space for your answers, and indicate the **question number clearly** as well as in the original answer box. **Do NOT** use it for your rough work.

— END OF ANSWER SHEET —