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.

$$T(n) = n + 20 \cdot \log^{100}(n) + 20$$

[2 marks]

B.

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

[2 marks]

C.

$$T(n) = \sqrt{n}\log(n) + n$$

[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. $f_2(n) = \log(\log(n))$
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.

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

[3 marks]

B.

$$T(n) = \begin{cases} T(n/10) + 10n & n > 10 \\ 1 & n \leq 10 \end{cases}$$

[3 marks]

C.

$$T(n) = \begin{cases} 100000000 & n > 100000000 \\ T(n-1) + T(n-2) & n \leq 100000000 \end{cases}$$

[3 marks]

Question 3: Jagged Sorting [9 marks]

You are given as input an array $A[1 \ldots n]$ of n **distinct** integers, (where n is **even**). The array is guaranteed to be in such a way that:

$$\forall 1 \leq i \leq n-2, A[i] < A[i+2] \tag{1}$$

An example of such an array is as follows:

$$[7, 1, 8, 9, 17, 10, 20, 12]$$

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 array A of n integers, we want to find the smallest positive (non-zero) integer that is missing from the array. You are promised that $\forall 1 \leq i \leq n$, $1 \leq A[i] \leq n$.

For example, given an array $A[1..n] = [6,3,1,2,6,4,6]$, smallest missing integer is 5. On the other hand, given an array $A[1...n] = [3,6,5,1,4,2]$, the smallest missing integer is 7.

```
1  FindFirstMissing(A[1..n]){
2      for(int index1 = 1; index1 <= n; ++index1){
3          int next = A[index1];
4          while(next != A[next]) {
5              int original_value = A[next];
6              A[next] = next;
7              next = original_value;
8          }
9      }
10
11     int check_index = 1;
12     while(check_index <= n && A[check_index] == check_index){
13          ++check_index;
14     }
15     return check_index;
16 }
```

The first iteration of any loop is when $j = 1$, and last iteration of any loop is when $j = n$. State for the following invariants, whether it is **True** or **False**.

At the end of the j^{th} iteration (lines 3-8) for the loop (on line 2):

A. For all values $1 \leq i \leq j, A[i] = i$. [1 mark]

B. For all values $1 \leq x \leq j$ where x is in $A[1..n]$, $A[x] = x$. [1 mark]

C. For all values $1 \leq i \leq j, A[A[i]] = A[i]$. [1 mark]

D. The smallest j values of $A[1..n]$ are in $A[1..j]$. [1 mark]

At the end of the j^{th} iteration (line 13) for the loop at line 12:

E. For all values $1 \leq i \leq j, A[i] = i$. [1 mark]

F. For all values $1 \leq x \leq j$ where x is in $A[1..n]$, $A[x] = x$. [1 mark]

G. For all values $1 \leq i \leq j, A[A[i]] = A[i]$. [1 mark]

H. The smallest j values of $A[1..n]$ are in $A[1..j]$. [1 mark]

Question 5: Finding Irregularities [6 marks]

Consider any array $A[1..n]$ of $n \geq 2$ **distinct** integers such that it is piecewise-sorted. In other words, there exists an index $1 \leq j < n$ such that:

1. The sub-array $A[1..j]$ is sorted.
2. The sub-array $A[(j+1)..n]$ is sorted.
3. The elements in the sub-array $A[1..j]$ are greater than the elements in the sub-array $A[(j+1)..n]$.

Here is an example of an array where $j = 2$:

$$A[1..6] = [9, 10, 1, 4, 5, 7]$$

Here is a snippet of some code that tries to find value j.

We are assuming 1-based indexing.

```
1  BinarySearch(A){
2      int low = 1
3      int high = n
4      while(low + 1 < high) {
5          int mid = (high - low) / 2 + low
6          if(A[low] > A[mid]) {
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: Answering Students [22 marks]

After lectures, Nodle typically has quite a few students who will come up to him to ask questions. However, as there is only one Nodle, he wants to know who is the next student he needs to service.

A student S_i will stand up at some point and walk towards Nodle. The student is represented by a tuple of positive integers (t_i, i, d_i, r_i).

1. The value t_i represents the starting time of the student.
2. The value i represents the unique ID of the student.
3. The value d_i represents the starting distance of the student.
4. The value r_i represents the rate at which the student is approaching.

Nodle will answer the student when he reaches Nodle's location, i.e. d_i reaches 0.

Here is an example scenario:

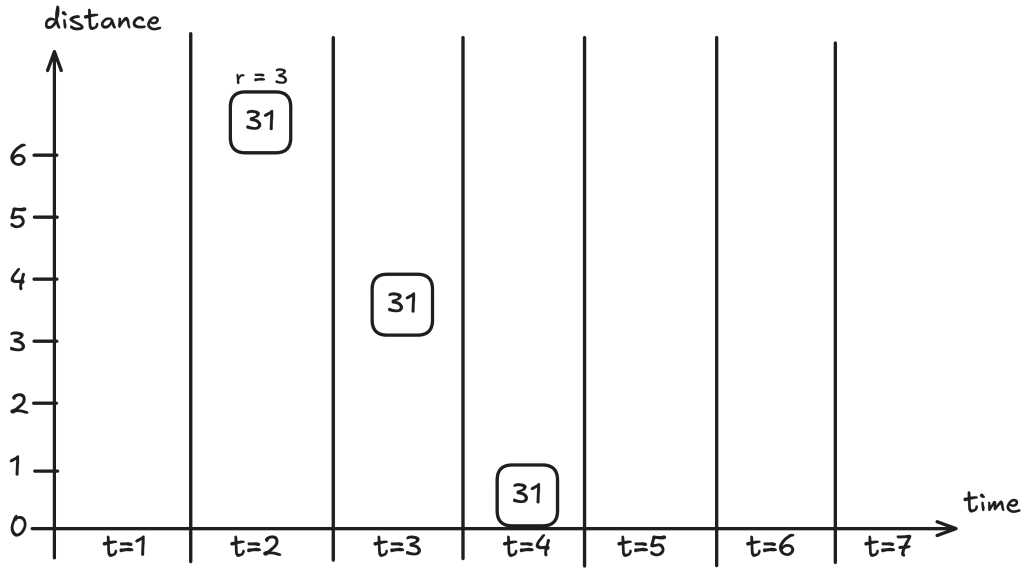


Figure 1: Tracking only student 31

Consider the scenario where a student 31, with rate $r = 3$, is added at distance $d = 6$ at time $t = 2$. Suppose that Nodle wants know who is the next student he has to answer at $t = 2$ onards. The answer would be 31. Similarly, if Nodle had checked with $t = 1, 2, 4$, the answer would also be 31. On the other hand, if Nodle wanted to know which student he had to answer at $t = 5$, the answer is -1.

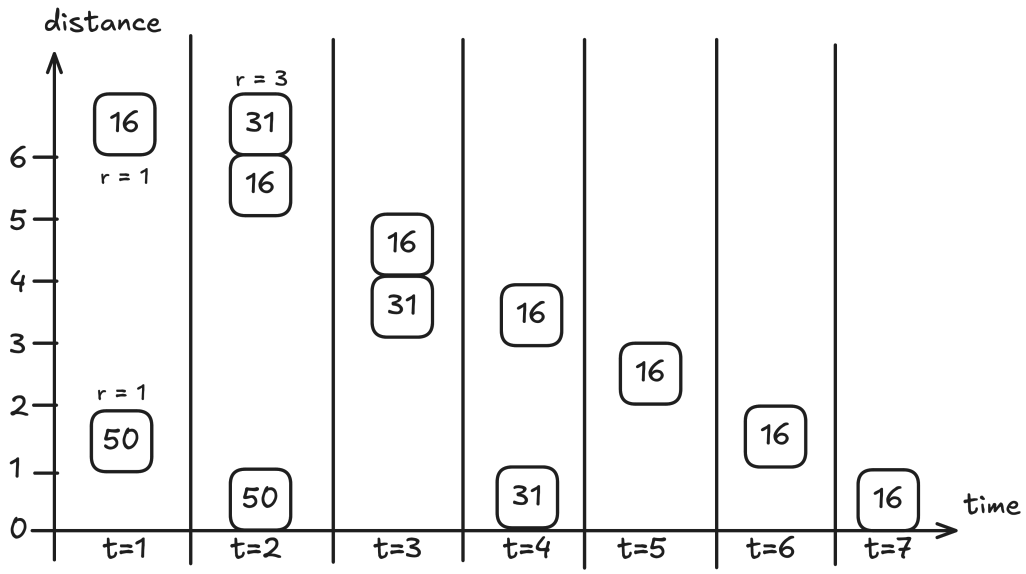


Figure 2: Tracking student 16 **and** 31 **and** 50

Now let's say that there are 2 new students: 16, and 50.

Student 16 starts at time $t = 1$, at distance $d = 6$, and at a rate of 1.

Student 50 starts at time $t = 1$, distance $d = 1$, and a rate of 1.

After we add these 2 students, then if at time $t = 1, 2$, if Nodle wants to know who he should answer next, the answer is 50. At time $t = 3, 4$, the answer is 31. And at time $t = 5, 6, 7$, the answer is 16.

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

1. `AddStudent`(t_i, i, d_i, r_i): Add a student i who starts approaching Nodle at minute t_i, from starting distance $d_i > 0$, at speed $r_i > 0$. Does not return any value.
2. `Answer`(t): Returns integer i which is the id of the next student (from all the added students) Nodle will answer earliest from time t onwards. Return -1 if no such student exists.

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

1. `AddStudent(2, 31, 6, 3)`
2. `Answer(1)` $\rightarrow 31$
3. `Answer(5)` $\rightarrow -1$
4. `AddStudent(1, 16, 6, 1)`
5. `AddStudent(1, 50, 1, 1)`
6. `Answer(1)` $\rightarrow 50$
7. `Answer(4)` $\rightarrow 31$
8. `Answer(5)` $\rightarrow 16$

You may assume that:

1. Every student eventually reaches Nodle at exactly distance 0 meters.
2. No two students will reach Nodle at the same time.
3. Every student has a unique id i.

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 rank/select, get min/max, lookup, delete, successor/predecessor operations are available in worst case $O(\log(n))$ time for a tree of size n. [3 marks]

B. Describe your algorithm (in pseudocode) for `AddStudent`(t_i, i, d_i, r_i). [2 marks]

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

D. Describe your algorithm (in pseudocode) for `Answer`(t). [2 marks]

E. What is the running time for your implementation of Answer? [1 mark]

As an extension, student i now also comes with a positive priority score p_i (in addition to all the previous information). Nodle wishes to support a new type of query:

- `AnswerPriority`(t): Returns integer i which is the id of the next student (from all the added students) with the highest priority p_i, among all the students that will reach Nodle at time t onwards. Returns -1 if no students will reach Nodle at time t onwards.

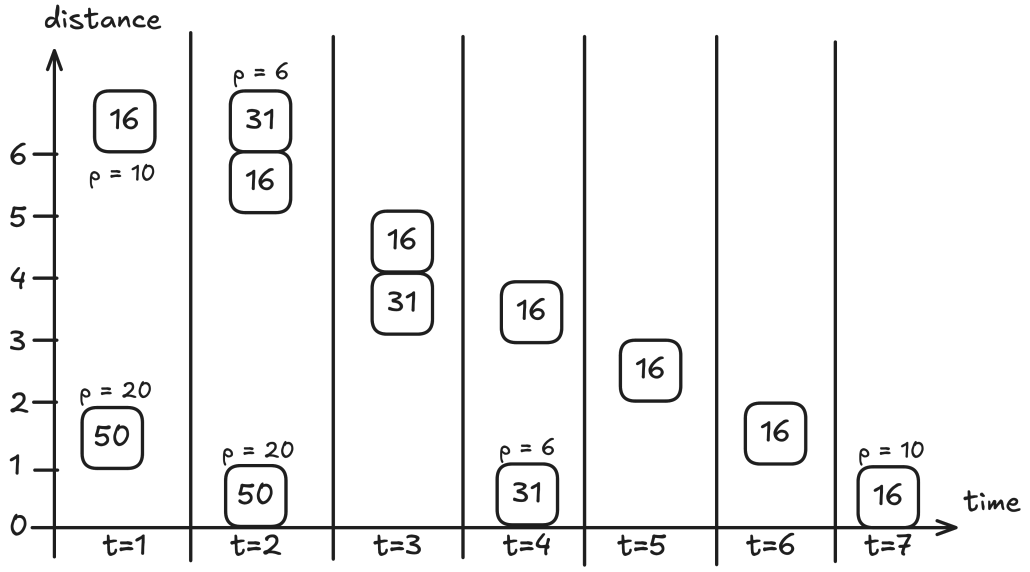


Figure 3: Now with priorities

For example, consider the same scenario again, but this time where student 16 has priority score 10, but this time where student 31 has priority score 6, but this time where student 50 has priority score 20.

Then for example, after the 3 students are added:

1. `AnswerPriority`(1) returns 50
2. `AnswerPriority`(2) returns 50
3. `AnswerPriority`(4) returns 16

F. Describe an augmentation for your data structure from part **A.** to handle this new query operation. [3 marks]

G. Describe (in pseudocode) how to maintain the augmented data during inserts, and AVL rotations. [4 marks]

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

I. Describe (in pseudocode) `AnswerPriority(t)` algorithm. High level ideas are enough. [4 marks]

J. What is the running time for your implementation of `AnswerPriority`? [1 mark]

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

Given a perfect circle, 3 points on the circumference of this circle are selected uniformly at random for us to place the legs of a chair. What is the probability that this chair is able to stand (and not flop over)? [0 mark]

— E N D O F P A P E R —

CS2040S—Data Structures and Algorithms

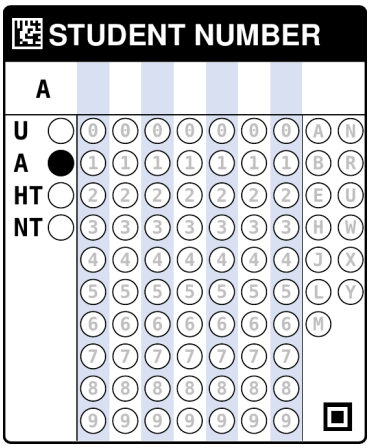
Midterm Assessment Answer Sheet

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 **SEVEN (7) 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.



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]

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

Question 2B

[3 marks]

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

Question 2C

[3 marks]

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

Question 3A Describe sorting algorithm. [7 marks]

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









}
```

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

Question 4A For all values $1 \leq i \leq j, A[i] = i$. [1 marks]

◯ True ◯ False

Question 4B For all values $1 \leq x \leq j$ where x is in $A[1..n]$, $A[x] = x$. [1 marks]

◯ True ◯ False

Question 4C For all values $1 \leq i \leq j, A[A[i]] = A[i]$. [1 marks]

◯ True ◯ False

Question 4D The smallest j values of $A[1..n]$ are in $A[1..j]$. [1 marks]

◯ True ◯ False

Question 4E For all values $1 \leq i \leq j$, $A[i] = i$. [1 marks]

◯ True ◯ False

Question 4F For all values $1 \leq x \leq j$ where x is in $A[1..n]$, $A[x] = x$. [1 marks]

◯ True ◯ False

Question 4G For all values $1 \leq i \leq j$, $A[A[i]] = A[i]$. [1 marks]

◯ True ◯ False

Question 4H The smallest j values of $A[1..n]$ are in $A[1..j]$. [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 AddStudent. [2 marks]

```
class DataStructure {
    void AddStudent(int time, int id, int distance, int rate){
        /* Your pseudocode here */




    }
}
```

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

Question 6D Describe your algorithm (in pseudocode) for Answer. [2 marks]

```
class DataStructure {
    int Answer(int time){
        /* Your pseudocode here */




    }
}
```

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

Question 6F Describe an augmentation for your data structure from part A to handle this new query operation. [3 marks]

Question 6G Describe how to maintain the augmented data during inserts, and AVL rotations. [4 marks]

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

Question 6I Describe (in pseudocode) AnswerPriority(t) algorithm. High level ideas are enough. [4 marks]

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

Question 7 Probability that chair can stand. [0 marks]

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]

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

Question 2B

[3 marks]

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

Question 2C

[3 marks]

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

Question 3A Describe sorting algorithm. [7 marks]

```
void JaggedSorting(int[] arr, int n){
    1. split the array based on even vs odd indices into
       two temp arrays.
    2. run the merge step of the mergesort algorithm on the
       two temp arrays into a new array.
    3. copy it back into arr.
}
```

Marks are awarded based on correctness and running time of algorithm:

1. 7 marks for correct $O(n)$ solution
2. 4 marks for correct $O(n\log(n))$ solution
3. 2 marks for correct $O(n^2)$ solution
4. 1 mark for correct Bucket/Radix Sort
5. 0 mark for incorrect algo

1 mark is penalized if array.remove() calls that causes the algo to be $O(n^2)$ are used.

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

$O(n)$ **Explanation:** Notice that the algorithm ends when **left_idx** = $n-1$ and **right_idx** = n, and each iteration, exactly one of the two indices increases by 1. This means that there are at most n iterations in total, each iteration costs $O(1)$ in total for the comparisons and copying.

Question 4A For all values $1 \leq i \leq j, A[i] = i$. [1 marks]

◯ True ● False

Question 4B For all values $1 \leq x \leq j$ where x is in $A[1..n]$, $A[x] = x$. [1 marks]

◯ True ● False

Question 4C For all values $1 \leq i \leq j, A[A[i]] = A[i]$. [1 marks]

● True ◯ False

Question 4D The smallest j values of $A[1..n]$ are in $A[1..j]$. [1 marks]

◯ True ● False

Question 4E For all values $1 \leq i \leq j$, $A[i] = i$. [1 marks]

● True ○ False

Question 4F For all values $1 \leq x \leq j$ where x is in $A[1..n]$, $A[x] = x$. [1 marks]

● True ○ False

Question 4G For all values $1 \leq i \leq j$, $A[A[i]] = A[i]$. [1 marks]

● True ○ False

Question 4H The smallest j values of $A[1..n]$ are in $A[1..j]$. [1 marks]

○ True ● False

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

high = mid

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

low = mid

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

low

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

```
class DataStructure {
    // Declare any data structures or variables here
    // This AVL tree maps arrival times to student IDs
    Tree<int, int> tree;


}
```

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

```
class DataStructure {
    void AddStudent(int time, int id, int distance, int rate){
        /* Your pseudocode here */
        // tree.insert(time + distance / rate, id);
        1. compute the arrival time using the formula
        2. time + distance / rate.
        3. Insert it as the key into the tree, and id is the value
    }
}
```

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

$O(\log(n))$ time for inserting into an AVL tree.

Question 6D Describe your algorithm (in pseudocode) for Answer. [2 marks]

```
class DataStructure {
    int Answer(int time){
        /* Your pseudocode here */
        if the maximum key of the tree is < time, then
                return -1.
        Otherwise, get the successor of key from the tree,
            and return the respective ID that it maps to.
    }
}
```

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

$O(\log(n))$ for the successor operation.

Question 6F Describe an augmentation for your data structure from part A to handle this new query operation. [3 marks]

Augment each node to also store 3 things: (1) the priority score of the node, (2) the maximum priority score that is in the entire sub-tree rooted at that node; and (2) a reference/pointer to node with that respective maximum priority score.

Question 6G Describe how to maintain the augmented data during inserts, and AVL rotations. [4 marks]

When adding a student to the data structure with priority score p_i, the node is initially set so that the node's priority score is p_i, max priority score is p_i, and it points to itself.

Then, traverse back up from the inserted leaf to the root, update the max priority score in the following way:

`max_priority = max(self.priority, left_subtree.max_priority, right_subtree.max_priority)`.

Update the references/pointers similarly.

The same formula applies during AVL rotations.

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

$O(\log(n))$

Question 6I Describe (in pseudocode) AnswerPriority(t) algorithm. High level ideas are enough. [4 marks]

Find the successor node of t in the tree. Then start the walk back up the tree. As you walk up the tree, check the current node and the right child of the node (starting with the successor node).

For any nodes that have an arrival time of $\geq t$, we update the max node we have seen so far accordingly, using the `max_priority` data to see if there is a new maximum, and `max_reference` to know which node has that priority.

After we are done walking back up to the root, we have the correct node and can return its ID.

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

$O(\log(n))$

Question 7 Probability that chair can stand. [0 marks]