NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM TEST FOR

Semester 1, AY2020/21

CS2040 – Data Structures and Algorithms

28 September 2020 Time allowed: 1.5 hours

STUDENT **NO.** :	A	0	1						

INSTRUCTIONS TO CANDIDATES

1. Copy the `ans_blank.txt` text file and rename it to `<your NUSNET ID>.txt` e.g. `e0123456.txt`. Write your **student number** and **NUSNET ID** within the appropriate tags of the copied file. Ensure you **answer question 0** as instructed. Failure to do so will prevent your submission from being graded
2. This is an open-hardcopy-notes examination but **WITHOUT** electronic materials
3. It is your responsibility to ensure that you have submitted the correct file with the correct particulars and format. If you submit the wrong file, name the file incorrectly, fail to provide correct particulars or change the contents of the file such that it cannot be parsed, we will consider it as if you did not submit your answers. In the best case, marks will be deducted
4. No extra time will be given at the end of the test for you to write your particulars. You must do it **before** the end of the test. However, you may upload your file after the end of the test
5. This paper consists of **5** inline questions including Q0, and **1** multiline question. It comprises ten (**10**) printed pages including this front page
6. Answer all questions within the text file. **Inline** answers should be appended to the end of each `#Qx`**:** part on the **same line**. Multiline answers should be **written between** the appropriate tags with **proper indentation** and adhering to the line limit. Avoid using the `#` character in both cases. Do NOT add, modify, remove any tag
7. Marks allocated to each question are indicated. Total marks for the paper is **45**
8. The use of electronic **calculator** is **NOT** allowed

Question	***Max***	***Marks***
Q0	Min -45	
Q1abcd	8	
Q2abc	6	
Q3abc	9	
Q4	6	
Q5ab	16	
Total	**45**	

Question 0 **[0 for ENTIRE PAPER if not done satisfactorily!]**

Please read the following NUS Code of Student Conduct (in particular the part on Academic, Professional and Personal Integrity), as well as items B and C below.

(A) **I am aware of, and will abide by the NUS Code of Student Conduct (in particular the part on Academic, Professional and Personal Integrity as shown below) when attempting this assessment.**

- Academic, Professional and Personal Integrity
  1. The University is committed to nurturing an environment conducive for the exchange of ideas, advancement of knowledge and intellectual development. Academic honesty and integrity are essential conditions for the pursuit and acquisition of knowledge, and the University expects each student to maintain and uphold the highest standards of integrity and academic honesty at all times.
  2. The University takes a strict view of cheating in any form, deceptive fabrication, plagiarism and violation of intellectual property and copyright laws. Any student who is found to have engaged in such misconduct will be subject to disciplinary action by the University.
  3. It is important to note that all students share the responsibility of protecting the academic standards and reputation of the University. This responsibility can extend beyond each student's own conduct, and can include reporting incidents of suspected academic dishonesty through the appropriate channels. Students who have reasonable grounds to suspect academic dishonesty should raise their concerns directly to the relevant Head of Department, Dean of Faculty, Registrar, Vice Provost or Provost.

(B) **I have read and understood the rules of the assessments as stated below.**

1. Students should attempt the assessments on their own. There should be no discussions or communications, via face to face or communication devices, with any other person during the assessment.
2. Students should not reproduce any assessment materials, e.g. by photography, videography, screenshots, or copying down of questions, etc.

(C) **I understand that by breaching any of the rules above, I would have committed offences under clause 3(l) of the NUS Statute 6, Discipline with Respect to Students which is punishable with disciplinary action under clause 10 or clause 11 of the said statute.**

3) Any student who is alleged to have committed or attempted to commit, or caused or attempted to cause any other person to commit any of the following offences, may be subject to disciplinary proceedings:

l) plagiarism, giving or receiving unauthorised assistance in academic work, or other forms of academic dishonesty.

To answer Q0, type either your student number or NUSNET ID to declare that you have **read and will abide** by the NUS Code of Student Conduct (in particular, (a) Academic, Professional and Personal Integrity), (b) and (c).

Ans:

Question 1 **[8 marks, 4 x 2]**

Ivan tried to sort an array of **N** numbers, but you don't agree with how he does it:

```
int[] arr = ...; // N numbers here
ArrayList<Integer> list = new ArrayList<>();

// missing code, see (a) - (d)

for (int idx = 0; idx < list.size(); idx++)
      arr[idx] = list.get(idx); // copy back from AL to array
```

"Why does this guy keep sorting the data so many times??" you wonder... ¯_(ツ)_/¯

The missing code snippet uses either insertion sort or **improved** bubble sort, but modified to work with **ArrayList** instead of an array. This description should be intuitive enough, but for your convenience:

```
void insertionSort(ArrayList<Integer> a){
      for (int i = 1; i < a.size(); i++) {
            int next = a.get(i);
            int j;
            for (j = i - 1; j >= 0 && a.get(j) > next; j--)
                  a.set(j+1, a.get(j));
            a.set(j+1, next);
      }
}
void bubbleSort(ArrayList<Integer> a) {
      for (int i = 1; i < a.size(); i++) {
            boolean is_sorted = true;
            for (int j = 0; j < a.size() - i; j++) { // left to right pass
                  if (a.get(j) > a.get(j+1)) {
                        swap(a, j, j+1);
                        is_sorted = false;
                  }
            }
            if (is_sorted)
                  return;
      }
}
```

What is the time complexity of each of these cases? Fill in the big-O time complexity (need tightest bound) of each case **without justification**:

a)
```
for (int idx = 0; idx < arr.length; idx++) {
      list.add(0, arr[idx]); // add to front of list
      insertionSort(list);
}

O(                    )
```

b)
```
for (int idx = 0; idx < arr.length; idx++) {
      list.add(arr[idx]); // add to back of list
      insertionSort(list);
}

O(                    )
```

c)
```
for (int idx = 0; idx < arr.length; idx++) {
      list.add(0, arr[idx]); // add to front of list
      bubbleSort(list); // improved bubble sort
}

O(                    )
```

d)
```
for (int idx = 0; idx < arr.length; idx++) {
      list.add(arr[idx]); // add to back of list
      bubbleSort(list); // improved bubble sort
}

O(                    )
```

The symbol for exponentiation is ^ and the symbol for factorial is !

Question 2 **[6 marks, 3 x 2]**

In cryptography, it is common to compute:

(**a** ^ **b**) modulo **M** i.e. the remainder from (a^b / **M**)

where **a**, **b**, and **M** are positive integers and **M** > 1

One efficient (and correct) way to calculate this result is to calculate its "square root" recursively:

```
int modExp(int a, int b, int M){ // rec. calls underlined for convenience
    if (b == 0)
        return 1;

    if (b % 2 == 0) {
        int sqrt = modExp(a, b / 2, M);
        return (sqrt * sqrt) % M; // Statement S1
    }

    int less = modExp(a, b - 1, M);
    return (a * less) % M; // Statement S2
}
```

Currently, statements **S1** and **S2** run in O(1) time. However, Uncle Grandpa made mistakes while implementing the function =O... Help Uncle Grandpa compute the big-O time complexity (need tightest bound) of modExp(**a**, **b**, **M**) in terms of **a**, **b** and/or **M**. Remember to use the **correct variable(s)**. Do **NOT provide justification**:

a) The mistake is that statements **S1** and **S2** are replaced by statements that each run in O(**a**) time

O ( )

b) The mistake is that statements **S1** and **S2** are replaced by statements that each run in O(log **b**) time

O ( )

c) The mistake is that statements **S1** and **S2** are replaced by statements that each run in O(**b** log **b**) time

O ( )

The symbol for exponentiation is ^ and the symbol for factorial is !

Question 3 **[9 marks, 3 x 3]**

It is now the year 2040 and robots run the library. Books are arranged in very long piles vertically, since robots can move much faster and vertically higher than humans =X ... Next to some piles of books, there are empty spaces, and/or robot-controlled dropships that **one at a time** loads books from the top but drops books out from the bottom

A **pile** of books can be modeled as a **stack**, with the book nearest the ceiling being at the top of the stack
An empty **space** can be modeled as a **stack**, books are (of course) added to the top
A **dropship** can be modeled as a **queue**, with the tail being the top where books are added, and the front of the queue being the bottom of the ship where the books are dropped from

Robots can only transfer one book at a time, but are given commands in the form of:

```
<from><to>(<num>)
```

As an example:

```
AB(n/2)
```

indicates that the robot will transfer (**n**/2) books, one at a time, from location **A** to location **B**

[Edit:] Announced that can't command the robot to transfer books from a dropship back to itself, i.e. no QQ(x)

Example

Say pile **P** has **n** books, you want to flip **P** EXCEPT for the bottom **b** books which should remain untouched, and you are given 2 Stacks **A** and **B**. One of the 2 possible programs (without more movements than necessary) would be:

```
PA(n-b)AB(n-b)BP(n-b)
```

You may notice, the program to give the robot MUST:

- be a series of commands WITHOUT any separator in between commands
- have NO space(s) both within and outside the parentheses

You will definitely get marks deducted, or even get 0, if you do not **follow this format STRICTLY**

Your Tasks

A pile **P** has **n** books. Move the top **c** (**c** ≤ **n**)books to the bottom of **P** such that the relative positions of the books within the top section is still the same, and the relative positions of books within the bottom sections is still the same

E.g. **n** = 7, **c** = 3, T U V W X Y Z are the 7 books in **P**

P before:	Ground [T, U, V, W, **X, Y, Z**] Ceiling
P after:	Ground [**X, Y, Z**, T, U, V, W] Ceiling

There are 3 different independent scenarios. In each scenario, write a program for the robot to run (without performing more movements than necessary) using the **format** shown above, **without comments/justification**:

a) You have 2 empty spaces **A** and **B**, but NO dropship

Program:

b) You have only ONE empty space **A** and ONE dropship **Q**

Program:

c) You have only ONE dropship **Q** but NO empty space

Program:

Question 4 **[6 marks]**

Nothing interesting about this question. Boo hoo =(

Examine the code snippet below:

```
private static void rec(int x, ArrayList<Integer> a) {
      if (x < 0) return;
      if (x == 0) {
            System.out.print(a);
            return;
      }
      for (int i = 1; i <= x; i++) {
            ArrayList<Integer> copy = new ArrayList<>(a);
            copy.add(i);
            rec(x - i, copy);
      }
}
public static void rec(int x) { // pre-cond : x > 0
      rec(x, new ArrayList<Integer>());
}
```

For your convenience, the Java API of the toString() method of java.util.ArrayList is pasted here:

`public String toString()`

Returns a string representation of this collection. The string representation consists of a list of the collection's elements in the order they are returned by its iterator, enclosed in square brackets (`"[]"`). Adjacent elements are separated by the characters `", "` (comma and space). Elements are converted to strings as by `String.valueOf(Object)`.

Overrides:

`toString in class Object`

Returns:

`a string representation of this collection`

Source: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Write the output that is printed after the call to [Edit:] rec(5) completes

(This is an **INLINE** question) Ans:

INLINE
ANSWER

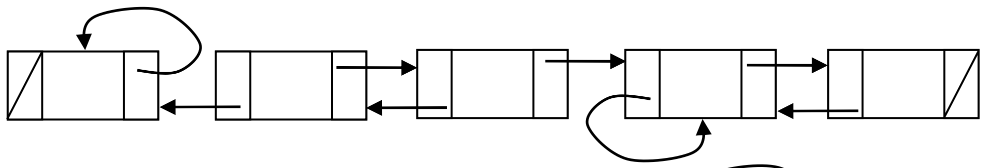
Question 5 **[16 marks, 6 + 10]**

Did you know that linked list can be used in files (and even in older file systems)? A large file need not be stored in one contiguous location on the disk. It can be stored as chunks of data (Nodes) that reference the next chunk. Consider a doubly-linked implementation that is NOT circular:

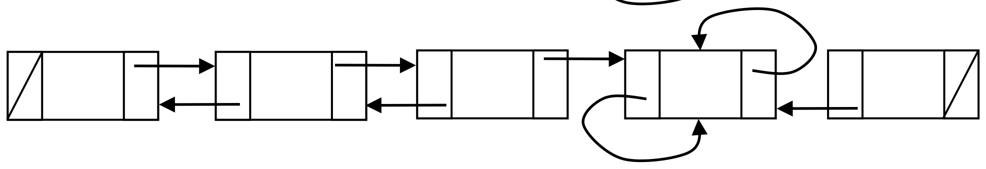
```
     class Node {
          public Node next, prev;
          public YourFileData data;

          // omitted constructors - doesn't concern you in this question
          // accessors(getters), mutators(setters)
     }
     class YourFile {
          private Node head, tail;
          private int numNodes;

          // omitted some other usual methods to copy-construct,
          //   append, remove, clear, search
          //   which don't concern you in this question

          // now some other methods which DO concern you in this question
          public int countForwardValidRefs() ...
          public int countBackwardValidRefs () ...
          public boolean canBeFixed() ... // for part (a)
          public void repair() ... // for part (b)
     }
```

For convenience, the Node attributes have been made public, so that you may either use getters/setters in your code, or access the attributes directly (without penalty)

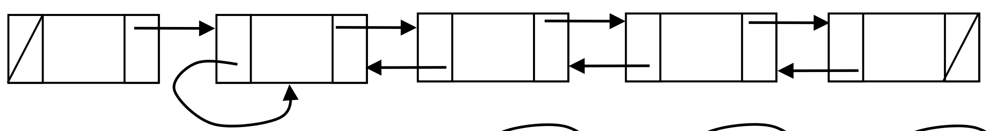
In this fictitious scenario, due to a previous bug while editing the file, the prev and/or next reference(s) of 0, 1 or more Nodes were corrupted, in that they point back to the same Node. Oh no!
The head, tail, and numNodes attributes still store the correct number of nodes though



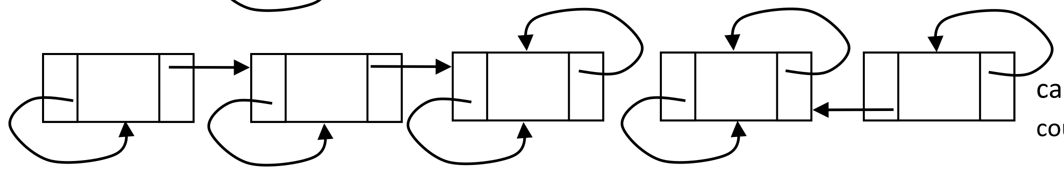
canBeFixed() -> false



canBeFixed() -> true
countForwardValidRefs() -> 3



canBeFixed() -> true
countForwardValidRefs() -> 5
countBackwardValidRefs() -> 3



canBeFixed() -> true
countForwardValidRefs() -> 2

Fortunately, Yuan Bin has written 2 methods, one for each direction. For a **given direction**, the method helps count the **number of consecutive nodes that have good links** in that direction. You **do NOT** need to understand the internal working of each method, but here it is:

```
public int countForwardValidRefs() {
      int valid = 0;
      for (Node curr = head; // some mind-boggling loop =D
            valid < numNodes && curr.next != curr;
            valid++, curr = curr.next);
      return valid;
}
public int countBackwardValidRefs(){
      int valid = 0;
      for (Node curr = tail; // some mind-boggling loop =D
            valid < numNodes && curr.prev != curr;
            valid++, curr = curr.prev);
      return valid;
}
```

"Maybe these methods might come in handy later...", you note

a) Write Java code to complete the canBeFixed() method which returns whether the linked list can be repaired, such that the prev and next pointers are all restored to how a linked list should look like. If there is no error, canBeFixed() should return true

[Line limit **24**] Ans:

MULTILINE
ANSWER

b) Write Java code to complete the repair() method which repairs the linked list, if it can be fixed. For bonus credit (+3 marks), this should be done in O(**N**) time

[Line limit **40**] Ans:

MULTILINE
ANSWER

- **End of Paper** -