NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM TEST ANSWERS FOR

Semester 1, AY2022/23

CS2040 – Data Structures and Algorithms

1 October 2022 Time allowed: 1.5 hours

STUDENT **NO. :**

A	N	S						

INSTRUCTIONS TO CANDIDATES

1. **Do NOT flip / turn over the test paper until you are told to do so**
2. Shade your **student number** in page 1 of the answer sheet. Do **NOT** write your name!
3. **COMPLETELY shade** the bubble for each answer using a fairly **dark pencil**, except for open-ended Q5, in which you may write legibly in either pen or pencil
4. **Do NOT rearrange, add/remove staples or add/remove pages** from the answer sheet. **Submit only the ENTIRE answer sheet** at the end of the test. It is your responsibility to ensure that you have submitted it, and submitted the correct answer sheet
5. If you fail to submit the correct answer sheet, fail to provide **correct particulars** or prevent the options from being **automatically detected** by software, we will consider it as if you did not submit your answers. In the best case, **marks will be deducted**
6. No extra time will be given at the end of the test for you to write your particulars, to shade or to fill in the answer sheet. You must do it **before** the end of the test
7. This paper consists of five (**5**) questions. Q1-4 are "MCQ", shade **at most one option per grid**. Write your code for Q5 in the allotted answer sheet. The question paper comprises ten (**10**) printed pages including this front page. The answer sheet comprises two (**2**) printed pages
8. This is an open-hardcopy-notes examination but **WITHOUT** electronic materials
9. Marks allocated to each question are indicated. Total marks for the paper is **45**
10. The use of electronic **calculator** is **NOT** allowed

Qn	***Max***	***Topic***
Q1ab	04	sort
Q2abcdefg	14	search vs sort AL vs LL time complexity
Q3ab	10	stack queue
Q4	09	recursion
Q5	08	ADT, LL
Total	**45**	

Legend
Easy
Medium
Difficult

Q1 [4 marks == 2 x 2]

A Java class X has **only one** comparison method properly **implemented** – **`boolean`** `less(X other)` – which returns whether the current object compares less than another. We want to sort an `ArrayList<X>` **`L`** containing **non-distinct elements** in **descending** (largest to smallest) order. The implementation of class X cannot be modified

Q1a. We can write our own comparison-based sorting function *in Java* that runs in O(n log n) time, so as to sort **L** correctly

◯ True	◯ False

Q1b. We can utilize *Java API's* `sort` function to sort **L** correctly

◯ True	◯ False

Yes, suppose there are 2 X objects a and b, wherever there is:

- a < b, the comparison can be implemented as a.less(b)
- a > b, the comparison can implemented as b.less(a)
- a <= b, the comparison can be implemented as !b.less(a)
- a >= b, the comparison can implemented as !a.less(b)

Although we cannot change the implementation of class X, we can write a class that implements Comparator<X>, and pass an object of that comparator into Collections.sort()

Q2 [14 marks == 7 x 2]

Computer memory can be visualized as in the diagram on the right, with the top row from left to right being the first 4 spaces, the middle row being the next 4 spaces, … and so on. There are only 12 spaces in the example on the right

Suppose there are:

- D0, D1, D2,… Dn - a large amount of data we are interested in, each Di having the same size
- P - other data that needs to remain untouched in memory
  as well as
- (blank) - unused space

The data stored in one space (e.g. the contents within D2) may be very large, but count a read/write of one space, or comparison of two spaces, as taking O(1) time

Data can be stored in **contiguous** fashion (illustrated below left), or as a **linked list** (illustrated below right) with nodes storing a **variable number of data spaces** in sequence within the node

Contiguous Data *Example*

	7	D0	D1	D2	D3	D4	D5
D6			P	P	P	P	P
P	P	P	P	P			
		P	P			P	
		P			P		

Linked List *Example*

7	3	D0	D1	D2	25		P
P	P	2	D5	D6	-1	P	P
P				P	P	P	P
P	2	D3	D4	10	P	P	P
P							

In this illustration, the first node shows there are a total of 7 data spaces across all nodes:

- The first node itself has 3 data spaces, storing data D0 D1 D2, and the node references address 25 which is the bottom-most node
- The second node has 2 data spaces, stores data D3 D4, and the node references address 10 which is the node situated in the middle
- The third and last node has 2 data spaces, stores data D5 D6, and the node does not reference any node hence storing -1

Notes:

- The sizes of memory, number of data spaces D0, D1, D2, … Dn, the locations of other unmovable data (P) and unused space (blank) illustrated here are just an **example**, and may not always be as such
- Every node will have at least 1 data space, there will be no "empty" nodes
- Algo using any space outside of a node counts as taking up additional space
- You may safely ignore << O(N) space taken up by system's call-stack if you are using a recursive algo
- You may claim the average/expected time for quick sort if pivot selection is randomized

For each question in **Q2a-g**, choose the time complexity of the best algorithm needed to solve the problem:

◯ O(log(log(N))	◯ O(log(N))	◯ $O(\sqrt{N})$	◯ $O(\sqrt{N}\log(N))$	◯ O(N)	◯ O(N log(N))
◯ $O(N^{1.5})$	◯ $O(N^{1.5}\log(N))$	◯ $O(N^2)$	◯ $O(N^2\log(N))$	◯ $O(2^N)$	◯ O(N!)

Suppose our data is stored in a **linked list**, and you are limited to using **O(1) additional space**

Q2a. if the data is unsorted, sorting D0 D1 D2 D3… Dn requires __O(N log(N))___ time
Quicksort with random pivot when partitioning can still be used

partition() does not require random access, the ends of the 2 partitions just need to advance sequentially (whether within the same node or to the next node) while keeping track of the size of the left partition, from which the pivot index and partition sizes can be computed

When recursing, instead of keeping track of indexes in the whole array, we can just keep track of the first index of a partition and the number of data spaces in it

Q2b. If there are **at least n-1 nodes** (there are n+1 data spaces 0..n) and the data is unsorted, sorting D0 D1 D2 D3… Dn requires _O(N log(N))_ time
Can do the same as Q2a

Alternatively, can just use the quicksort algorithm (with random pivot) as if each list node had 1 element each. If a node has more than 1 element, just use the first element. At most 2 nodes will have 1 or 2 extra elements, which we can shift into the correct place one at a time in O(N) each

Q2c. If there are **exactly 3 nodes** and the data is unsorted, sorting D0 D1 D2 D3… Dn requires _O(N log(N))_ time

Can do the same as Q2a

Alternatively, can first find the first data space and size of each of the 3 nodes. Then, perform quicksort on an array as usual, but each array access translates to a node and an offset from that node's first data space

We still have our data stored in a **linked list**, but now **space is NOT restricted** - there is enough unused space for your algorithm to run:

Q2d. If there are **exactly 3 nodes** and the data is **unsorted**, sorting D0 D1 D2 D3… Dn requires _O(N log(N))_ time

Doesn't matter how many nodes there are, since there is no space restriction, copy all elements out and use any efficient comparison-based sort

Q2e. If there are **exactly 3 nodes**, and the data **WITHIN**/inside **each node is sorted**, sorting D0 D1 D2 D3… Dn requires ___ O(N) ___ time

Merge the first two arrays into a buffer, then merge the buffer and third array into yet another buffer. Copy the second buffer back into the 3 nodes

Q2f. If there are not more than $\sqrt{\mathbf{n}}$ nodes, and the **entire data D0 D1 D2 D3… Dn is sorted**, finding whether a given integer exists in D0 D1 D2 D3… Dn requires __O($\sqrt{N}$)__ time

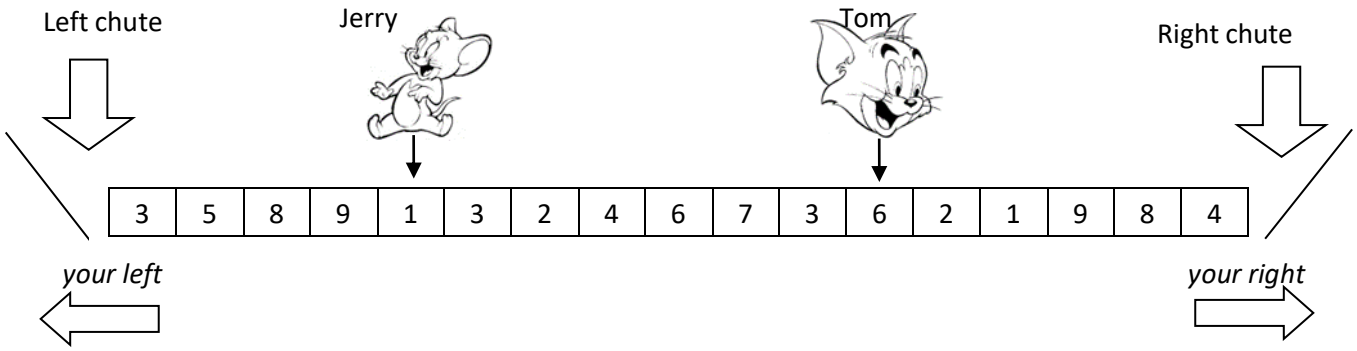
Can't do binary search on the entire data as linked list does not have random access, but can linear search (with 2 references) to find which node the data is in, then binary search that node in O(log N) time

Q2g. If there are not more than $\sqrt{\mathbf{n}}$ nodes, and the **entire data D0 D1 D2 D3… Dn is sorted**, finding the **k**th largest element for a given integer **k** requires __O($\sqrt{N}$)__ time

Requires linear traversal of all nodes (using 2 references) while accumulating the sizes of the nodes' data. Once the correct node is located, locating the correct data space takes O(1) time

Q3 [10 marks == 2 x 5]

Jerry and Tom[1] are both working in a factory, standing along a line of items, each item having a positive integer value. This is illustrated in the diagram below:



The item values and the length of the line shown here are just an **example**

You want to implement a program that models these real-life operations:

- Item of given value rolls down the **left chute** and joins the line
  if 9 rolls down the left chute on the illustration, then the line of items will be [9, 3, 5, 8, …]
- **Jerry** takes the item *to your immediate left* of the item directly in front of him if exists, and throws it over Tom's head into the **right chute**, the item joining the line there
  if called on the illustration, Jerry will take 9 out of the line and the 9 ends up as the rightmost element, the line of items will be [3, 5, 8, 1, 3, …, 9, 8, 4, 9]
- **Jerry** checks that there is at least 1 item on the line between him and **Tom** (excluding the ones in front of them), and if there is, **Jerry** removes the item directly in front of him, after which the item that was on *your immediate right* naturally closes the gap – **Output** the item **Jerry** removed, if any
  if called on the illustration, Jerry will remove 1 from the line and be standing behind 3. The line of items will be [3, 5, 8, 9, 3, 2, 4, …]. 1 will be output by this operation
- **Jerry** adds an item of given value *to your immediate right* of the item directly in front of him
  if 9 is added, the line of items will be [3, 5, 8, 9, 1, 9, 3, …]
- **Tom** takes the item in front of him if exists, throws it at Jerry – the item **leaves the system** and the item on *your immediate right* naturally closes the gap – If the operation does remove an item, **output** the item **Tom** threw, **AND also** the item **Jerry** is in front of
  if called on the illustration, 6 leaves the system, Tom will be standing behind 2, the line of items will be […, 6, 7, 3, 2, 1, 9, …]. "6 1" will be output by this operation
- **Output** the sum(sum of value of items to the left of **Jerry**, sum of value of those to the right of **Tom**)
  if called on the illustration, the answer and output is 49 (== 25 + 24)
- **Output** the sum of value of items from **Jerry** to **Tom** *inclusive*
  if called on the illustration, the answer and output is 32 (== sum([1, 3, 2, …, 7, 3, 6]))

Each operation must be done in **O(1)** time. Deferring the cost to another operation is NOT acceptable here

You have the **choice** to keep track, or NOT keep track, of anything that you are not required to output

[1] Adapted from https://coloringonly.com/images/imgcolor/1548379186-tom-and-jerry-coloring-pages-lovely-tom-and-jerry-thumbs-up-coloring-page-tom-and-jerry-coloring-pages-of-tom-and-jerry-coloring-pages.jpg

For **each** of the question in **Q3a-b INDEPENDENTLY**, choose the minimal/simplest data structure(s) you need, i.e. **as far left an option as possible**, aside from O(1) spaced variables

Q3a. Only linked list **WITHOUT** any extra iterators/pointers midway through

BLL – Basic Linked List

TLL – Tailed Linked List

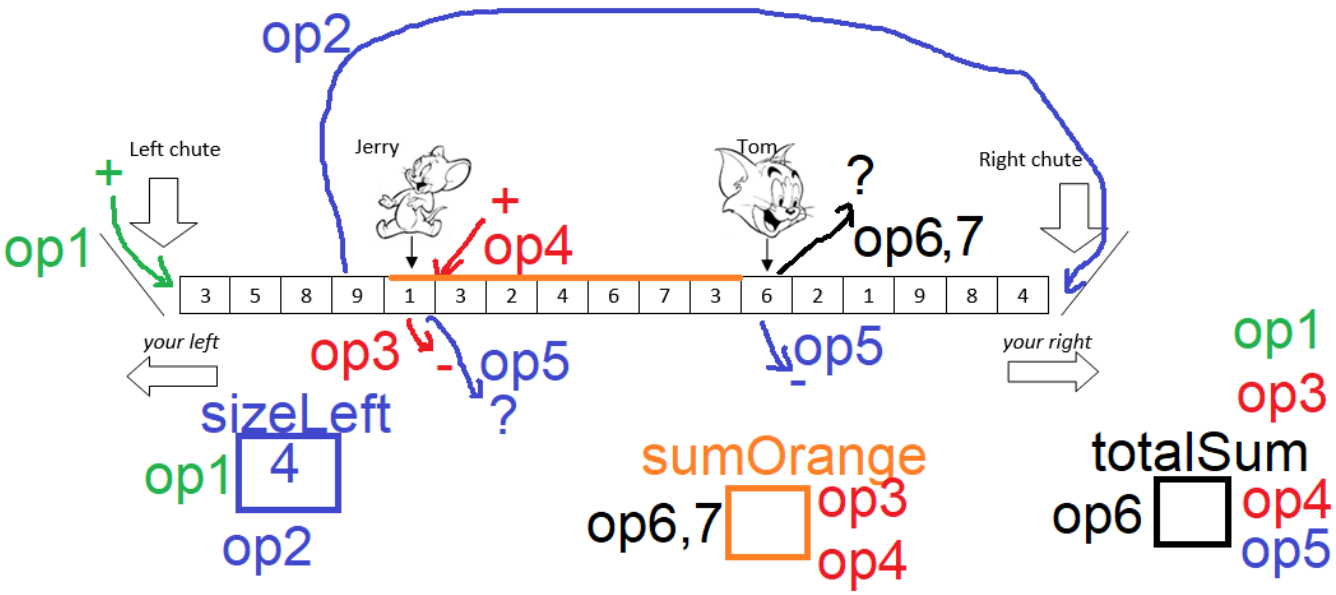
DLL – Doubly Linked List

◯ 2 BLLs	◯ 1 BLL + 1 TLL	◯ 2 TLLs	◯ 1 TLL + 1 DLL	◯ 2 DLLs	◯ 1 TLL + 2 DLLs

Q3b. Only Stack (S) and/or Queue (Q)

◯ 2 S	◯ 1 S + 1 Q	◯ 2 Q	◯ 2 S + 1 Q	◯ 1 S + 2 Q	◯ 2 S + 2 Q

Picture speaks a 1000 / 10,000 words



BLANK SPACE

TILL END OF PAGE

Q4 [9 marks]

Match the following 4 code snippets to the 3 problems which are **independent** of one another. A problem will be correctly solved by 0 or 1 code snippet, while a code snippet will solve 0 or 1 problem

P1: Given the row and column indexes of the bottom-rightmost cell of a rectangular grid, find the shortest number of jumps from the bottom-rightmost cell to the top-leftmost cell where (r, c) == (0, 0). Each jump can only move you one cell leftward or upward. Additionally, *when along the principal diagonal* (diagonal that will reach (0, 0) if extended top-left-ward) you can also choose to jump one cell top-left-ward

P2: Given 2 lists/arrays of integer sequences, find the smallest number of terms that have to be (replaced, removed or added) one-at-a-time, to transform the first sequence into the second sequence

P3: Given the row and column indexes of the bottom-rightmost cell of a rectangular grid, find the shortest number of jumps moving from the bottom-rightmost cell to reach the top-leftmost where (r, c) == (0, 0). Each jump can only move you one cell leftward, upward, rightward or downward from/to a cell that has a value of `true`. Both bottom-rightmost and top-leftmost cells have the value of `true`

For each of code snippets **Q4A-D** in **pseudocode**, shade the problem it **correctly solves** if any:

◯ P1	◯ P2	◯ P3

Marks will be awarded for the entire Q4 instead of for individual parts. Shading option(s) when the code snippet does NOT solve those problem(s) *will be penalized*

BLANK SPACE

TILL END OF PAGE

Code Snippet **Q4A**

◯ P1	◯ P2	◯ P3

(None)

```
f(a, b) {
   if (a == -1) return +INF;
   if (b == -1) return +INF;
   x = 1 + min(f(a-1, b), f(a, b-1)); // +INF + 1 -> +INF
   if (a == b) x = min(x, 1 + f(a-1, b-1));
   return x;
}
ans = f(a, b);
```

Associated with P1 but will always return +INF due to missing base case

Code Snippet **Q4B**

◯ P1	◯ P2	◯ P3

(None)

```
f(a, b, c, d) {
   if (c < 0 || d < 0 || c > b || d > b || a[c][d]==false) return +INF;
   if (c == 0 && d == 0) return 0;
   return 1 + min(f(a, b, c-1, d), f(a, b, c, d-1), // +INF + 1 -> +INF
                  f(a, b, c+1, d), f(a, b, c, d+1));
}
ans = f(a, b, a, b);
```

Associated with P3 but will often get stuck in inf recursion as problem doesn't shrink
(If fixed, very similar to tutorial flood fill example)

Code Snippet **Q4C**

◯ P1	◯ P2	◯ P3

```
f(a, b, c, d) {
   if (c < 0) return 1+d;
   if (d < 0) return 1+c;
   if (a[c] == b[d]) return f(a, b, c-1, d-1);
   return 1 + min(f(a, b, c-1, d-1), f(a, b, c-1, d), f(a, b, c, d-1));
}
ans = f(a, b, length(a) - 1, length(b) - 1);
```

This problem is known as edit distance. Where an element matches, we just look at the previous element. Otherwise, we make 1 edit out of 3 possibilities and take edit with the minimum distance, either replacing the current element, deleting it, or adding it, respectively
(The algorithm can be improved to polynomial time instead of exponential time by using dynamic programming (DP), which is out of CS2040 syllabus)

Code Snippet **Q4D**

◯ P1	◯ P2	◯ P3

```
f(a, b) {
   if (a == b) return a;
   if (a == 0) return b;
   if (b == 0) return a;
   return 1 + min(f(a-1, b), f(a, b-1));
}
ans = f(a, b);
```

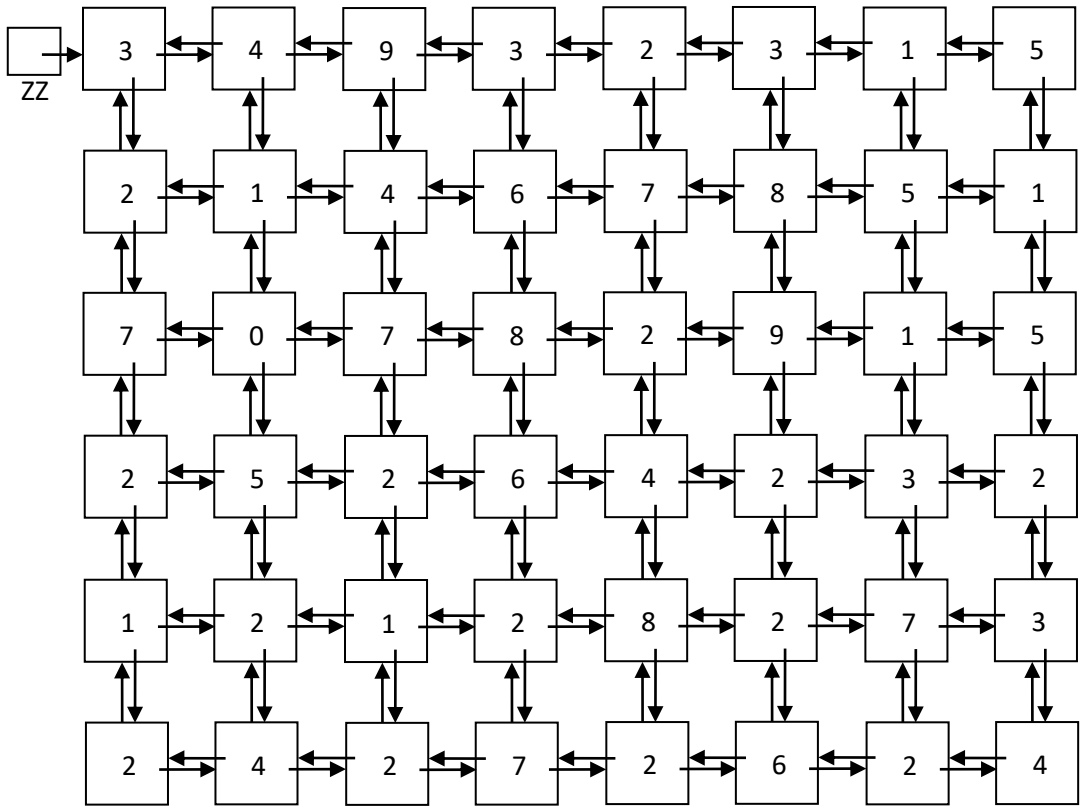
You may or may not get confused because the base case are shortcuts instead of what was described in the question (This problem can actually be solved in O(1) time, recursion not needed =X)

Q5 [8 marks + 3?]

You have a rectangular **N** x **M** Grid, of which a smaller version is abstracted in the diagram below. Each rectangle is a GridNode objects. The GridNode class is defined below the diagram

ZZ is a GridNode reference (will not be null) pointing to the top-left node of the grid. Each GridNode object has references pointing to nodes in the respective directions

Before




```
class GridNode {
     public GridNode left, right, up, down;
     public int value; // any number from 0 to 2^24-1 inclusive
}
```

Ivan is interested in swapping the place of two **K** x **L** rectangles for $1 \leq \mathbf{K} \leq \mathbf{N} - 2$ and $1 \leq \mathbf{L} \leq \mathbf{M} - 2$. It is guaranteed that the two rectangles will:

- NOT intersect each other
- NOT be on the border of the grid
- NOT border each other on any of the 4 sides

If you perform this operation, which may be performed repeatedly, correctly in:

- O(**N**+**M**) time, you will get the full 8 marks and 3 bonus marks **(WARNING: time consuming)**
- O(**N**+**M**+**KL**) time with O(1) space, you will get the full 8 marks
- O(**N**+**M**+**KL**) time with O(**KL**) space, you will get 5 marks
- O(**KL**(**N**+**M**)) time with O(1) space, you will get 3 marks

[If you have no time left, you might want to go for a lower-tier but correct solution to cut loss]

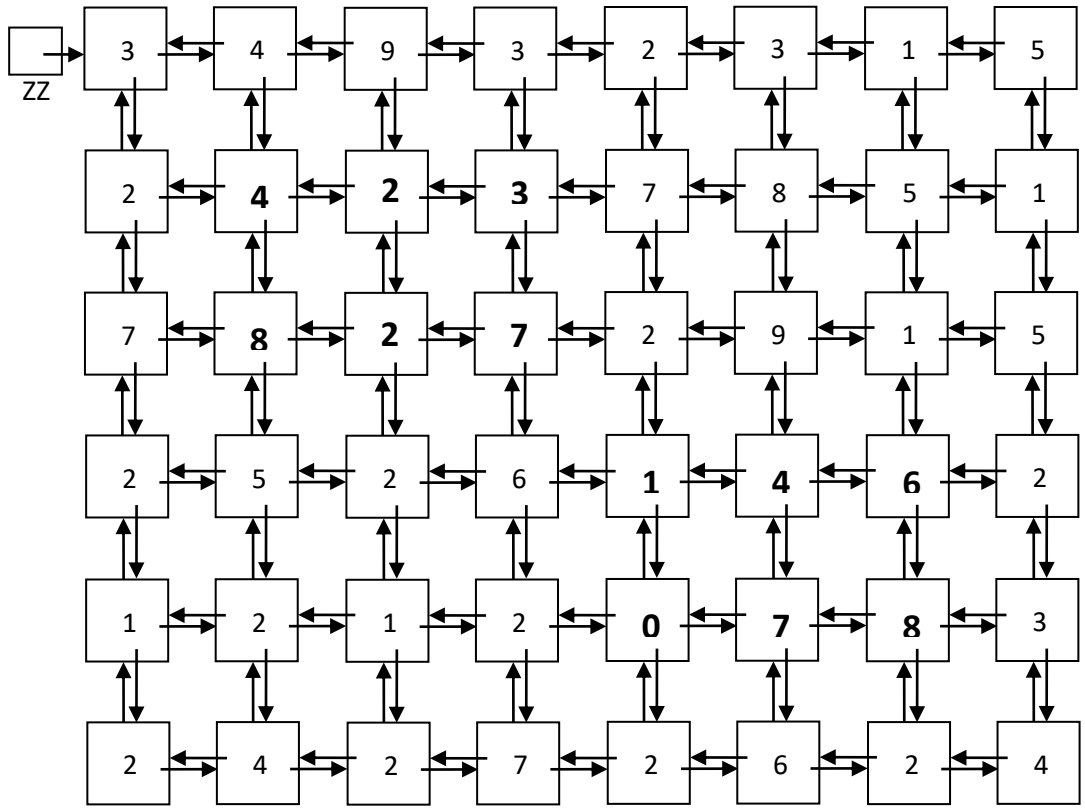
In the `exchange` method of the `Grid` class, you are given **K**, **L**, the (valid) top-left row and column indexes (0-based) of each rectangle **R1**, **C1** and **R2**, **C2** that are to be swapped. You are also given some other instance methods (that **MAY** or may **NOT** help you solve the problem?)

Your task is to **implement the `exchange` method** correctly and efficiently. You may implement other method(s) in the Grid class if it helps you

```
class Grid {
   GridNode ZZ; // (row, col) = (Zero, Zero)

   public void exchange(int K, int L, int R1, int C1, int R2, int C2) {
      // your Q5 answer here
   }

   void swap(GridNode left, GridNode right) {
      int newLeftValue = right.value;
      right.value = left.value;
      left.value = newLeftValue;
   }

   void cutRight(GridNode from, int cuts) {
      GridNode above = from.up;
      while (cuts-- > 0) {
         from.up = null; above.down = null;
         from = from.right; above = above.right;
      }
   }
   void cutDown(GridNode from, int cuts) {
      GridNode beside = from.right;
      while (cuts-- > 0) {
         from.right = null; beside.left = null;
         from = from.down; beside = beside.down;
      }
   }

   void stitchRight(GridNode bot, GridNode top, int stitches) {
      while (stitches-- > 0) {
         bot.up = top; top.down = bot;
         bot = bot.right; top = top.right;
      }
   }
   void stitchDown(GridNode left, GridNode right, int stitches) {
      while (stitches-- > 0) {
         left.right = right; right.left = left;
         left = left.down; right = right.down;
      }
   }
}
```

The earlier grid would look like the diagram below after the execution of `exchange(2, 3, 3, 4, 1, 1)` is completed

After



- End of paper -

For bonus marks, need to locate 16 references first (4 inside and 4 outside for each of the 2 rectangles), then cut the 2 rectangles out and stitch the swapped rectangles back

For full marks, just be careful to create new references before traversing left and right. Swap node data instead of attempting to cut and stitch nodes together

O(KL) space but good time – Copying one or both rectangles out before writing to swap rectangles

Bad time complexity but O(1) space – Writing a method to get to a cell, but calling that method repeatedly instead of advancing references

Copying the entire grid out is **NOT acceptable** as it takes too much space (against the grading bands given)

```
GridNode move(GridNode from, int row, int col) {
   while (row-- > 0) from = from.down;
   while (col-- > 0) from = from.right;
   return from;
}

// O(N+M) with bonus marks
GridNode[] getCorners(int K, int L, int R, int C) { // CCW except bot
right
   return new GridNode[]{
      move(ZZ, R, C),
      move(ZZ, R+K-1, C),
      move(ZZ, R, C+L-1)
   };
}
public void exchange(int K, int L, int R1, int C1, int R2, int C2) {
   GridNode[] r1I = getCorners(K, L, R1, C1), // inside
      r2I = getCorners(K, L, R2, C2),
      r1O = {r1I[0].up, r1I[0].left, r1I[1].down, r1I[2].right}, //
outside, CCW
      r2O = {r2I[0].up, r2I[0].left, r2I[1].down, r2I[2].right};
   cutRight(r1O[2], L); cutRight(r1I[0], L);
   cutDown(r1O[1], K); cutDown(r1I[2], K);
   cutRight(r2O[2], L); cutRight(r2I[0], L);
   cutDown(r2O[1], K); cutDown(r2I[2], K);

   stitchRight(r1O[2], r2I[1], L); stitchRight(r1I[0], r2O[0], L);
   stitchDown(r1O[1], r2I[0], K); stitchDown(r1I[2], r2O[3], K);

   stitchRight(r2O[2], r1I[1], L); stitchRight(r2I[0], r1O[0], L);
   stitchDown(r2O[1], r1I[0], K); stitchDown(r2I[2], r1O[3], K);
}

// O(N+M+KL) "full" marks
public void exchange(int K, int L, int R1, int C1, int R2, int C2) {
   GridNode head1 = move(ZZ, R1, C1), head2 = move(ZZ, R2, C2);
   for (int r = 0; r < K; r++) {
      GridNode curr1 = head1, curr2 = head2;
      for (int c = 0; c < L; c++) {
         swap(curr1, curr2);
         curr1 = curr1.right; curr2 = curr2.right;
      }
      head1 = head1.down; head2 = head2.down;
   }
}
```

Marking Scheme

	A	B
Q1a	**2**	
Q1b	**2**	

	A	B	C	D	E	F	G	H	I	J	K	L
Q2a						**2**			1			
Q2b						**2**						
Q2c						**2**						
Q2d						**2**						
Q2e					**2**							
Q2f		1	**2**									
Q2g			**2**									

	A	B	C	D	E	F
Q3a		**5**	3	2	1	
Q3b		**5**			3	

	A	B	C	(NA)
Q4A	-2	-3	-3	**+3**
Q4B	-3	-3	-2	**+3**
Q4C	-3	**+3**	-3	-3
Q4D	**+3**	-3	-3	-3

Q4 has min 0, max 9