NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM TEST FOR

Semester 1, AY2021/22

CS2040 – Data Structures and Algorithms

6 October 2021 Time allowed: 1.5 hours

STUDENT **NO. :**

A	0	1						

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 pages 1 AND 3 of the answer sheet. Do **NOT** write your name!
3. **Answer question 0** as instructed, or your submission will NOT be graded
4. Do NOT rearrange the answer sheet, or add/remove staples. **COMPLETELY shade** the bubble for each answer using a fairly **dark pencil**
5. **Submit only the 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
6. 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**
7. No extra time will be given at the end of the test for you to write your particulars or shade the answer sheet. You must do it **before** the end of the test
8. This paper consists of **21** questions including Q0. Not more than one option should be shaded per grid. The question paper comprises fourteen (**14**) printed pages including this front page. The answer sheet comprises three (**3**) printed pages.
9. This is an open-hardcopy-notes examination but **WITHOUT** electronic materials
10. Marks allocated to each question are indicated. Total marks for the paper is **45**
11. The use of electronic **calculator** is **NOT** allowed

Sect Qn	***Max***	***Marks***
Q0	Min -45	
S1 Q1 - 8	12	
S2 Q9 - 14 (a, b)s	18	
S3 Q15 - 16	6	
S4 Q17-19, Q20a-f	9	
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.**

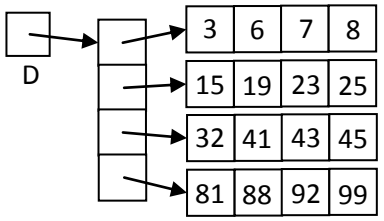
3) Any student who is alleged to have committed or attempted to commit, or caused or attempted to cause any other person to commit any of the following offences, may be subject to disciplinary proceedings:

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

To answer Q0, shade the option on the answer sheet 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: Read and AGREE

Section 1 - To represent a square matrix, the data structure **D** is used (an **example** in the *abstraction* on the right). **D** is an ArrayList<ArrayList<**Big**Integer>> containing **N** distinct integers (**N** is a positive even perfect square)



In this *example*
N = 16 (and NOT 4)

The integers will be in ascending order moving from left to right, this property wrapping around to the leftmost element of each lower row

While elements being BigInteger means each element can be EXTREMELY large, treat *comparisons* and *additions* between two BigIntegers as O(1) operations

Q1-4: We want to find an integer **X** among the **N** integers in **D** **[4 marks == 4 x 1]**
(**X** is guaranteed to exist in **D** for simplicity)

Which of these statements is/are possible approach(es) having the correct time complexities (tightly bounded):

Q1. T/F Can modify binary search to account for the 2D grid, it should take O(log **N**) time if properly implemented

Q2. T/F Can binary search the first column to select a row, then binary search that row to find **X**, it should take total of O(log(**N**)) time if properly implemented

Q3. T/F Can linear search the first column to select a row, then linear search that row, it should take total of $O(\sqrt{\mathbf{N}})$ time if properly implemented

Q4. T/F Can brute force search every element in the 2D grid, it should take $O(\mathbf{N}^2)$ if properly implemented

-= NOTHING TO SEE HERE =-

THIS SPACE IS INTENTIONALLY

LEFT BLANK

-= SCROLL DOWN FOR Q5 =-

For each question in Q5-14, options A-R refer to:

A $O(\log(\log(N))$	B $O(\log(N))$	C $O(\sqrt{N})$	D $O(N)$	E $O(N \log(N))$	F $O(N^{1.5})$	G $O(N^{1.5} \log(N))$
H $O(N^2)$	J $O(N^2 \log(N))$	K $O(N^3)$	L $O(N^4)$	M $O(2^N)$	P $O(N!)$	R $O(N^N)$

Your answer should just be **ONE character** corresponding to the time complexity you choose. It is possible that the same option can be the answer for more than one question, but a question will only have **exactly one answer**

Worked Example

The best algorithm to sum up all the elements in **D** will take [D] time *i.e. O(N) time*

Q5-6: **[4 marks == 2 x 2]**

Suppose *just for Q5-6* that **D** is a **Linked**List<ArrayList<BigInteger>> instead, with elements still in ascending order as before. We still want to find an integer **X** among the **N** integers in **D**:

Q5. The best solution involving binary search would then take [A-R] time

Q6. The best solution that does NOT involve binary search would then take [A-R] time

Q7-8: (**D** is back to an **Array**List<ArrayList<BigInteger>>) **[4 marks == 2 x 2]**

Now **N**/2 of the integers are randomly removed while keeping each row compact (no gaps between elements) but NOT shifting any elements from one row to another row

A function f(**D**, **A**) adds another **N**/2 integers into **D** (which now has only **N**/2 integers), from an **unsorted** array **A** of length **N**/2 (each new integer to be added will be distinct for simplicity, i.e. NOT be found within **D** and only found once within **A**). By the end of f, **D** has to end up being a 2D grid with dimensions same as before elements were removed, numbers in ascending order

Q7. Given O(**N**) extra space, the best implementation of f runs in [A-R] time

Q8. **-= Difficult question alert! =-** If f is implemented such that:

- it takes O(1) extra space aside from arraylist expansions
- elements from **A** (remember, **D** is sorted but **A** is unsorted) are sequentially inserted one at a time into the appropriate inner arraylist in **D** so that elements are still in ascending order
- after inserting all elements from **A** into **D**, the sizes of the inner arraylists in **D** are balanced by shifting elements between consecutive rows
  (e.g. element in row 4 has to pass through rows 3 and 2 to be shifted to row 1)

then the best implementation of f given these constraints/direction will run in [A-R] time

-= NOTHING TO SEE HERE =-

THIS SPACE IS INTENTIONALLY

LEFT BLANK

-= SCROLL DOWN FOR Q9 =-

Section 2 - Tom and Jerry have written 2 functions fSort and gSort to sort an array. For each function, what happens if some sorting-related algorithm is used within the function? For each question, independent of the other questions, answer whether a call to the function over the entire array can successfully sort the array, and the time complexity of the function (if exists). Q9-11 work on the entire array using fSort, while Q12-14 work on the entire array using gSort

Important NOTE: As long as the algorithm does not crash and terminates gracefully, you **should still state the time complexity** of the function **even if it does not successfully sort** the array

Q9-14 (Options A-R same as those on previous page) **[18 marks == 6 x (2 + 1)]**

A $O(\log(\log(N))$	B $O(\log(N))$	C $O(\sqrt{N})$	D $O(N)$	E $O(N \log(N))$	F $O(N^{1.5})$	G $O(N^{1.5} \log(N))$
H $O(N^2)$	J $O(N^2 \log(N))$	K $O(N^3)$	L $O(N^4)$	M $O(2^N)$	P $O(N!)$	R $O(N^N)$

```
fSort(arr, lo, hi) {
     if lo >= hi return
     mid = (lo + hi) / 2
     fSort(arr, lo, mid)
     hmmm(arr, 1 + mid, hi)
     merge(arr, lo, mid, hi) // merge as used in mergeSort
}
```

Q9. If **hmmm** is replaced with `fSort`, then the call to `fSort(arr, 0, arr.length - 1)`:

9a) [T/F] successfully sorts the array

9b) takes worst case [A-R/X] time (answer X if algorithm crashes or does not terminate)

Q10. If **hmmm** is replaced with `quickSort` as in lectures, then the call to `fSort(arr, 0, arr.length - 1)`:

10a) [T/F] successfully sorts the array

10b) takes worst case [A-R/X] time (answer X if algorithm crashes or does not terminate)

Q11. If **hmmm** is replaced with `partition` as used in quickSort in lectures, then `fSort(arr, 0, arr.length - 1)`:

11a) [T/F] successfully sorts the array

11b) takes worst case [A-R/X] time (answer X if algorithm crashes or does not terminate)

-= NOTHING TO SEE HERE =-

THIS SPACE IS INTENTIONALLY

LEFT BLANK

-= SCROLL DOWN FOR Q12 =-

```
gSort(arr, lo, hi) {
  if lo >= hi return
  here = hmmzz(arr, lo, hi)
  gSort(arr, lo, here - 1)
  gSort(arr, 1 + here, hi)
}
```

Q12. If **hmmzz** is replaced with `partition` as used in quickSort in lectures, then `gSort(arr, 0, arr.length - 1)`:

12a) [T/F] successfully sorts the array

12b) takes worst case [A-R/X] time (answer X if algorithm crashes or does not terminate)

Q13. If **hmmzz** is replaced with `selectMax`, which swaps the largest element in the range [`lo`..`hi`] with the element at index `hi` and returns the **rightmost** index `hi`, then the call to `gSort(arr, 0, arr.length - 1)`:

13a) [T/F] successfully sorts the array

13b) takes worst case [A-R/X] time (answer X if algorithm crashes or does not terminate)

Q14. If **hmmzz** is replaced with `bubble`, which performs *one iteration of the outer loop* of bubbleSort (bubbling rightwards only, as in lectures) but on the range [`lo`..`hi`] and returns the **leftmost** index `lo`, then `gSort(arr, 0, arr.length - 1)`:

14a) [T/F] successfully sorts the array

14b) takes worst case [A-R/X] time (answer X if algorithm crashes or does not terminate)

Section 3 - You have stacks **S**, **R**, a queue **Q** and a List **A**. The List **A** starts off containing **N** elements (**N** is a positive even integer) while **S**, **R** and **Q** start off empty. Your intern has written some long-winded programs and you are to simplify them to produce the same end result

Each operation takes the form **XY**, where **X** != **Y**, which moves exactly **N**/2 elements from ADT **X** to ADT **Y** using:

If X or Y is a ___	then ___ from X	and ___ into Y
List	removeFirst	addLast
Stack	pop	push
Queue	dequeue	enqueue

Your answer should be a sequence of operations separated by just one space between two consecutive operations. For example `SR SR RQ` moves **N** elements from stack S to stack R, then moves **N**/2 elements from stack R to queue Q...

Important Notes

- Use as **few operations as possible** for full credit
- **You will get 0 marks** for a question if your answer is **malformed**, **incorrect**, or **takes more operations** than a direct translation of the code snippet
- **X** and **Y** must be different instances, but their data types may be the same, e.g. `SR` is ok but `AA` and `SS` are NOT

Worked Example

```
qExample(A) {
  while not A.isEmpty() do    Q.enqueue(A.removeFirst())
  while not Q.isEmpty() do    S.push(Q.dequeue())
  while not S.isEmpty() do    R.push(S.pop())
  while not R.isEmpty() do    A.addLast(R.pop())
}
```

Answering `AQ AQ QS QS SR SR RA RA` which is the direct translation of qExample will result in only 1 mark

Using more than 8 operations (the direct translation above takes 8 operations) will result in 0 marks

Answering (answer left blank) will result in 3 marks as it is a sequence which uses the **fewest number of operations** (0 operations in this case) to produce the same end result in all the ADTs **S**, **R**, **Q** and **A**

Q15-16 **[6 marks == 2 x 3]**

```
q15(A) {
  while not A.isEmpty() do    S.push(A.removeFirst())
  while S.size() != R.size() do    R.push(S.pop())
  while not R.isEmpty() do    A.addLast(R.pop())
  while not S.isEmpty() do    R.push(S.pop())
  while not R.isEmpty() do    A.addLast(R.pop())
}
```

Q15 operation: `SHADE EACH OP IN SEQUENCE`

```
q16(A) {
  while A.size() != S.size() do         S.push(A.removeFirst())
  while not A.isEmpty() do    R.push(A.removeFirst())
  while not R.isEmpty() do    S.push(R.pop())
  while not S.isEmpty() do    Q.enqueue(S.pop())
  while not Q.isEmpty() do    S.push(Q.dequeue())
  while not S.isEmpty() do    A.addLast(S.pop())
}
```

Q16 operation: `SHADE EACH OP IN SEQUENCE`

-= NOTHING TO SEE HERE =-

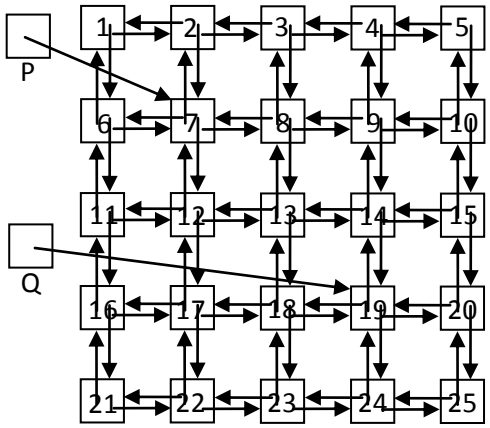
THIS SPACE IS INTENTIONALLY

LEFT BLANK

-= SCROLL DOWN FOR SECTION 4 =-

Section 4 - You have a rectangular grid which is abstracted in the diagram on the right. Each of the 25 rectangles are GridNode objects. **P** and **Q** are GridNode references. 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 1-100

}
```

Q17-19 **[6 marks == 1 + 2 + 3]**

3 code snippets, one for each question, are run *independently* on the original grid. Which diagram shows the resultant grid after the function completes execution?

Choose **exactly one option** for each question
In each option, the differences from the original grid are marked using *dashed lines* for convenience

-= NOTHING TO SEE HERE =-

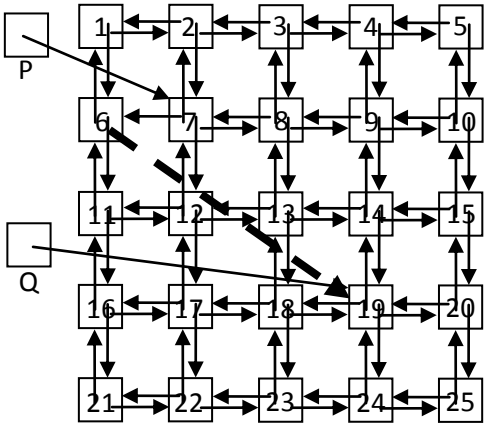
THIS SPACE IS INTENTIONALLY

LEFT BLANK

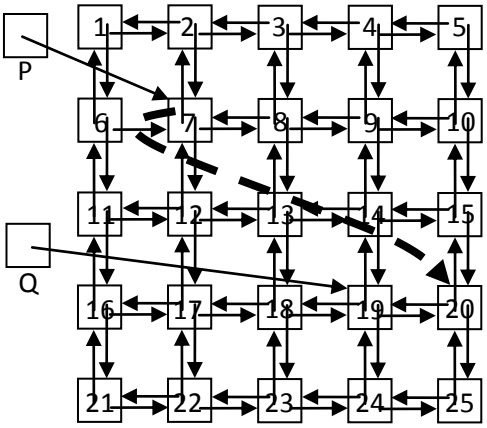
-= SCROLL DOWN FOR Q17 =-

```
q17(P, Q) {
     P.left = Q.right;
}
```

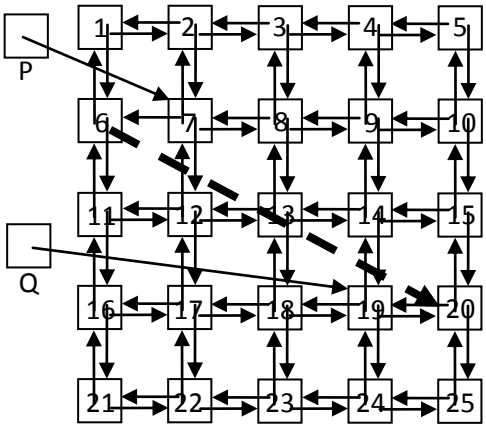
Option A



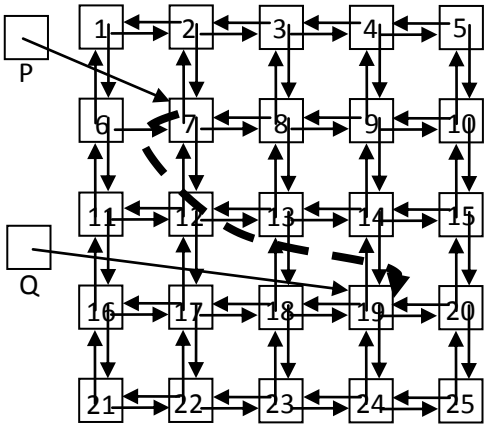
Option B



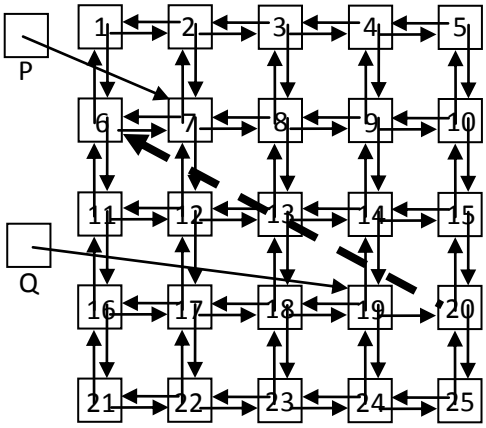
Option C



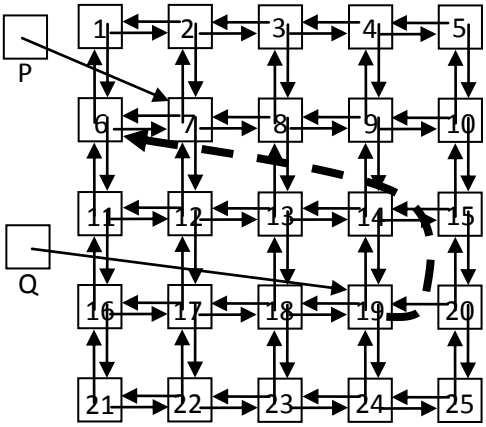
Option D



Option E



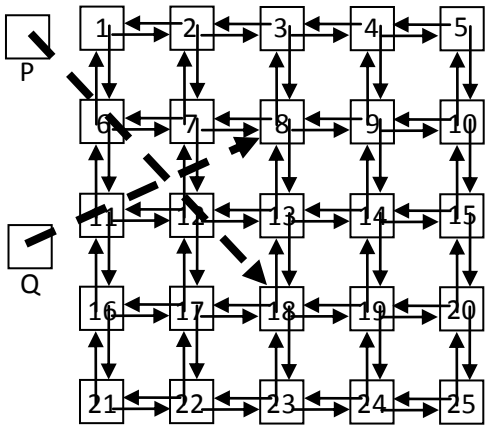
Option F



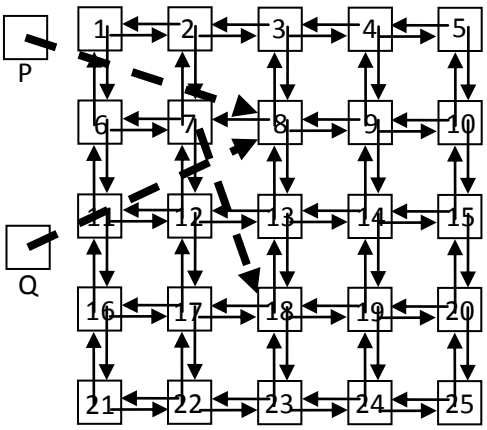
Q17 answer: A-F

```
q18(P, Q) {
      P = P.right;
      P.left.right = Q.right.left.left;
      Q = P;
      P = P.left.right;
      Q.left.right = Q;
}
```

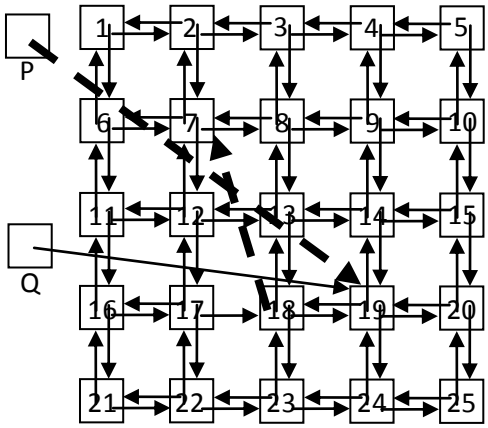
Option A



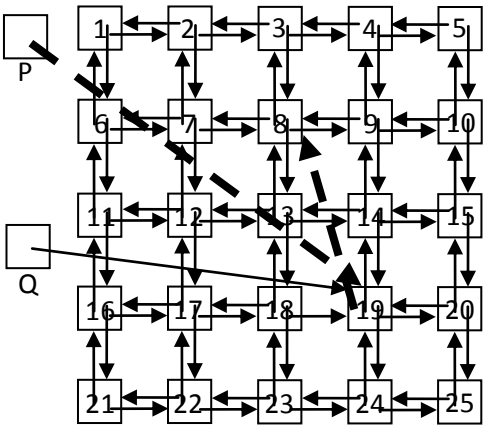
Option B



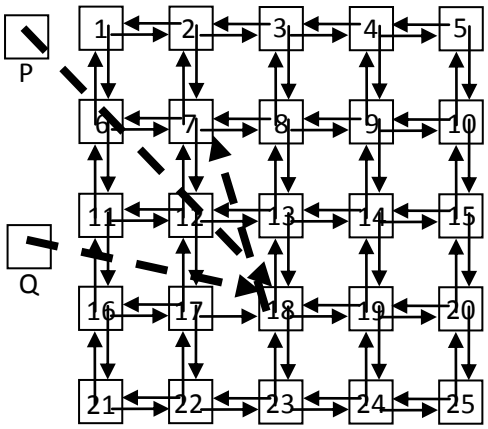
Option C



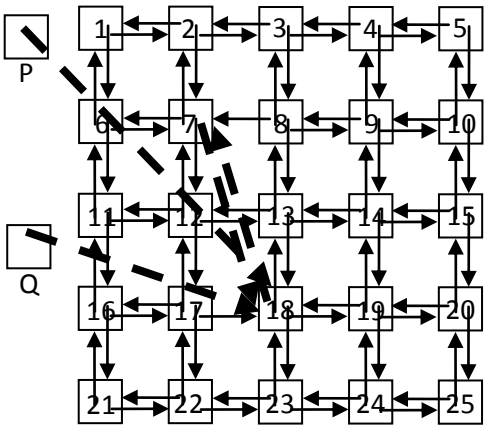
Option D



Option E



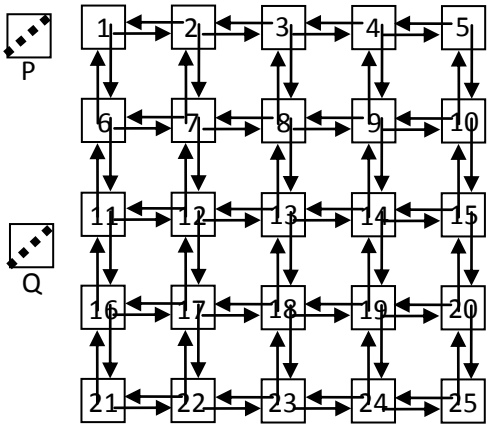
Option F



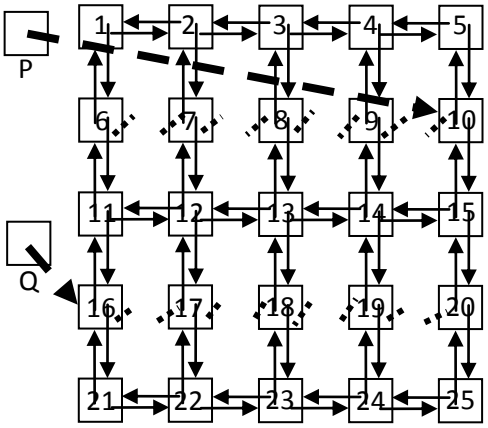
Q18 answer: A-F

```
q19(P, Q) {
  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;

    P.up.down = P.down; P.down.up = P.up;
    Q.up.down = Q.down; Q.down.up = Q.up;

    P = P.right; Q = Q.left;
  }
  P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
}
```

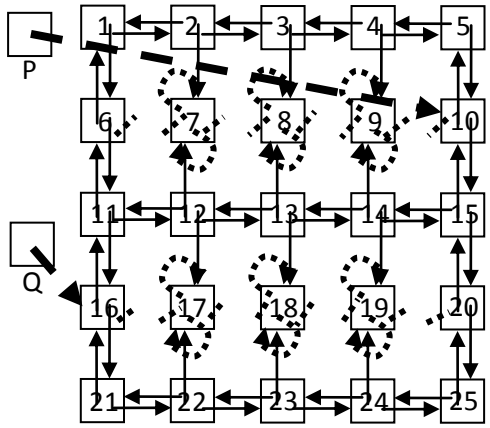
Option A



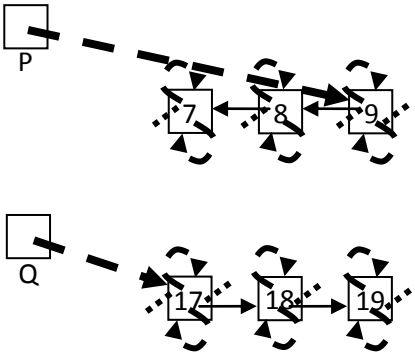
Option B



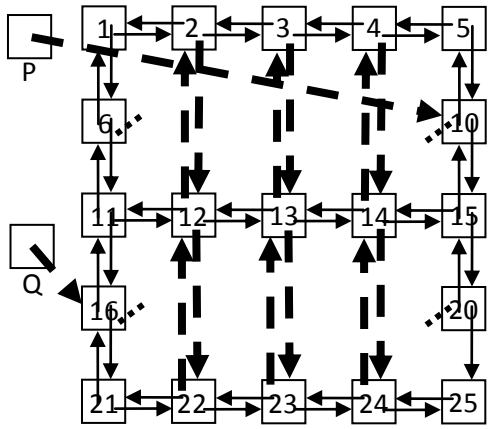
Option C



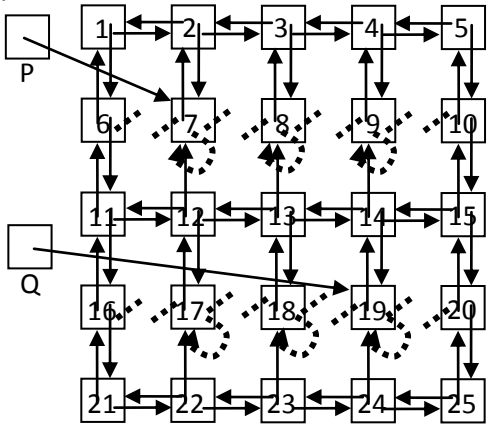
Option D



Option E



Option F



Q19 answer: A-F

Q20 **-= Time consuming question alert! =-** **[3 marks]**

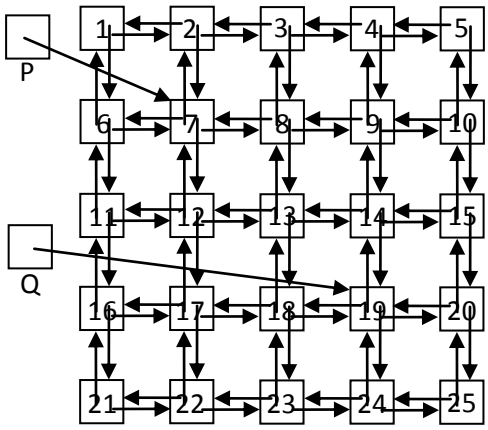
Which code snippet(s) below, when run on the original grid shown below-*left*, cause(s) the grid to look like the diagram shown below-*right*? After running, the nodes need not remain in the exact same locations in computer memory as before

You are given 6 options. **Zero, one or more of the options** may be true
Marks will be *awarded for the entire Q20* instead of for each option. **Marking scheme**:

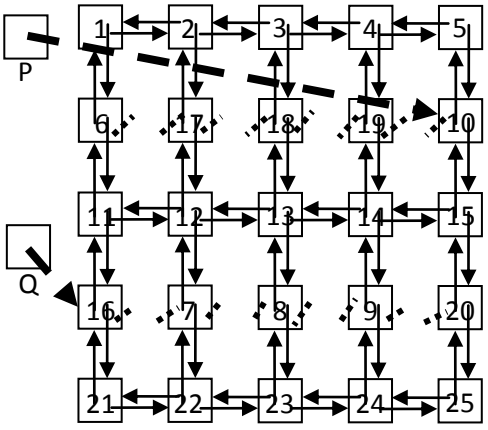
- At least 1 True and 1 False option have to be *correctly identified*, otherwise 0 marks
- 1 mark will be deducted for each wrong/invalid/blank option

-= Time consuming question alert! =-

Before (original grid)



After running your option



The changes to the grid after, as compared to before, are marked using *dashed lines* for convenience
Also, note the positions of 7, 8, 9, as well as 17, 18, 19

[Post-exam note: The intention was for B and F to be true, but forgot to swap the positions of 7 and 9 =( ]

```
runOptionA(P, Q) {
  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
    P.value = Q.value; Q.value = P.value;
    P = P.right; Q = Q.left;
  }
  P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
}
```

20a answer: T/F

-= Time consuming question alert! =-

-= Time consuming question alert! =-

```
runOptionB(P, Q) {
  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;

    P.value+=Q.value; Q.value=P.value-Q.value; P.value-=Q.value;//swap?

    P = P.right; Q = Q.left;
  }
  P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
}
```

20b answer: T/F

```
runOptionC(P, Q) {
  for (int i = 0; i < 3 - 1; i++)
    Q = Q.left;
  R = Q.left; // R is a new GridNode reference

  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.left.right=null; Q.left=null;

    P.value+=Q.value; Q.value=P.value-Q.value; P.value-=Q.value;//swap?

    P = P.right; Q = Q.right;
  }
  P.left.right=null; P.left=null; Q.left.right=null; Q.left=null;

  Q = R;
}
```

20c answer: T/F

```
runOptionD(P, Q) {
  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;

    P.up = Q.up; Q.up = P.up; P.down = Q.down; Q.down = P.down;
    P.up.down = P.down.up = Q; Q.up.down = Q.down.up = P;

    P = P.right; Q = Q.left;
  }
  P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
}
```

20d answer: T/F

-= Time consuming question alert! =-

-= Time consuming question alert! =-

```
runOptionE(P, Q) {
  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;

    P.up.down=P.down.up=Q; Q.up.down=Q.down.up=P; // reordered from ...
    P.up=Q.up; Q.up=P.up; P.down=Q.down; Q.down=P.down; // option D

    P = P.right; Q = Q.left;
  }
  P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
}
```

20e answer: T/F

```
runOptionF(P, Q) {
  for (int i = 0; i < 3; i++) {
    P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;

    P = P.up; P.down.up = Q.up; Q.up = P;
    P = P.down; P.up.down = P; Q.up.down = Q;

    P = P.down; P.up.down = Q.down; Q.down = P;
    P = P.up; P.down.up = P; Q.down.up = Q;

    P = P.right; Q = Q.left;
  }
  P.left.right=null; P.left=null; Q.right.left=null; Q.right=null;
}
```

20f answer: T/F

-= Time consuming question alert! =-

- End of paper -