Answer Sheet (31251, Autumn 2022)

Multiple-Choice Questions

Please type your choices (among A, B, C, D) into the space after each question number.

1	B	4	B	7	A	10	D	13	A B C
2	B	5	A D	8	A C D	11	C	14	A B C
3	A	6	B C D	9	A C D	12	D	15	A

Short-Answer Questions

Please type your answer into the space below each question number.

Question 16 (Word Limit: 100)

Here are four possible insertion order: (note that below are just some of the correct insertion orders; more correct insertions orders exist).

- `6, 3, 7, 2, 4, 9`
- `6, 3, 9, 2, 4, 7`
- `4, 2, 7, 3, 6, 9`
- `4, 3, 7, 2, 6, 9`

Question 17 (Word Limit: 100)

The code checks every node of a linked list. The return is the subtraction of

- The absolute of all non-zero data in all nodes

from

- *number* * the number of occurrences of 0 in the nodes' data

Question 18 (Word Limit: 100)

The code print outs the sum of all the numbers that form a path from the bottom-left corner to the top-right corner of a matrix.

Question 19 (Word Limit: 200)

The problem with the current insertion mechanism is that ints can only be stored in the diagonal of the two-dimensional array, which easily cause collisions and is a waste of space in the array.

Here are two possible solutions (other solutions may also exist):

Solution 1: Use two different hash functions for row and column, respectively.
Given an int number, we use

- One hash function to give the row index, say `X`.
- A second hash function to give the column index, say `Y`.
- The two hash functions must be different so that we can put numbers as evenly as possible to the matrix (rather than only putting them to the diagonal).

Collision handling: `// suppose X, Y are the hashing result of a number x.`

```
int a = X, b = Y, counter = 0;
while (counter < (n^2 - 1) && A[a][b] is occupied) {
```

```
        counter += 1;
        a = ( X + int((X + counter)/n) ) % n;
        b = ( Y + counter%n ) % n;
    }
    if (counter < (n^2 - 1))
        A[a][b] = x.
    else
        std::cout << "matrix is full" << std::endl;
```

Solution 2: Use the two-dimensional array as a one-dimensional array. As such, we need a mechanism to develop a one-to-one mapping between two-dimensional indices [0…n][0…n] and one dimensional indices [0…n^2-1]. Then, we only need one hash function with linear probing to insert ints to the one-dimensional indices and then convert every one-dimensional index to two-dimensional indices for completing the insertion.

Question 20 (Word Limit: 100)

We can use two layers to for-loops to check for the longest turbulence starting from every number.

Question 21 (Word Limit: 200)

We can reduce the computational cost during the computational process we provided in Question 20.
If a number is in the middle of a sequence, then it cannot be part of another sequence.

Bearing this in mind, we only need to check the possible turbulences from 1, 5, 7, 4, 3 when looking for the longest turbulence.