MCQs [4 marks per question: 40 marks]

Q1. For the following function, which data structure would the system use internally to evaluate it (n > 1). You can assume that the system calculates `factorial (n)` before `factorial (n – 1)`.

```
static int factorial(int n) {
    if (n == 0 || n == 1)
       return 1;
    return n * factorial(n - 1);
}
```

a) Stack

b) Queue

c) Linked List

d) Array

e) Hash Table

Ans: Stack is the data structure used as recursion has a first in last out property.

Q2. What is the time complexity for the following loop (m > 1, n > 1)?

```
for (int i = 1; i < n; i++) {
    i *= m;
}
```

a) O (n)

b) O (m)

c) O ($\log_m n$)

d) O ($\log_n m$)

e) None of the listed

Ans: The value of i will go from 1, m, m^2, m^3, m^4, …. m^k. Where $m^k - 1 = n$.

We want to get the value of k and hence we take the log with respect to m.

$\log_m (m^k - 1) = \log_m n$ ➔ $k = O(\log_m n)$.

Q3. What is the space complexity of the following function (n > 1)?

```
int fibonacci (int n) {
        int a = 0, b = 1, c = 0;
        if (n == 0)
                return a;
        if (n == 1)
                return b;
        for (int i = 2; i <= n; ++i){
                c = a + b;
                a = b;
                b = c;
        }
        return c;
}
```

a) O (n)

b) O (1)

c) O (n^2)

d) O (log n)

e) None of the above

Ans: O (1) since the only space needed is for the four variables (a, b, c, and n) used.

Q4. Six objects, namely, A, B, C, D, E, and F are added to a stack in alphabetical order. The first four items are then removed and added to a separate queue after which two items from the queue are dequeued and added back to the stack. We then take the element from the top of the stack. Which element is this?

a) A
b) B
c) C
d) D
e) E
f) F

Ans: Order on stack is (from top to bottom): F – E – D – C – B – A. After taking the 1st 4 elements and adding to the queue, the stack becomes: B – A while the queue (from front to back) is: F – E – D – C. Dequeuing the 1st two elements will give us F and E in that order and after adding to the stack, the stack becomes, E – F – B – A. So, the item to then be popped is E.

Q5. What is the time complexity of the following piece of code (N > 1)?

```
total = 0;
for (int i = 1; i < N; i++)
    for (int j = 1, j < i * i; j++)
       for (int k = 0; k < j; k++)
            total++;
```

a) O (N^2)
b) O (N^3)
c) O (N^4)
d) O (N^5)
e) None of the listed

Ans: The innermost loop gets executed j times, which is O (N^2) as j goes from 1 to i^2 (N^2). The loop with counter j gets executed N^2 times and the outermost loop N times. So, in total we get N * N^2 * N^2 ➔ O (N^5).

Q6. Which of the following statement is false in regard to finding the most efficient data structure (from array, linked list, stack, queue, and hash table) for the purpose of solving the given problem?

a) To manage a collection of data of the same data type, we would use an array.
b) We wish to maintain a group of numbers, which can grow or shrink over time, so we would go with a Linked List.
c) At a juice shop, we want to keep track of customers coming in and their orders and hence it is best to go with a Hash Table.
d) To insert a new value in an unsorted list, we can go with either a Stack or a Queue as the ADT.
e) None of the listed.

Ans: In this case, we would need a queue, rather than a Hash Table.

Q7. What is the time complexity of the following function, runLoop (N) (N > 1)?

```
runLoop (N) {
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j)
      System.out.println(" * ");
    for (int a = 0; k < i; ++a) {
      System.out.println(" * ");
      for (int b = 0; b < 1000; ++b)
        System.out.println(" * ");
    }
  }
}
```

a) O (N)

b) O (N^3)

c) O (N log N)

d) O (N^4)

e) O (N^2)

Ans: The outer loop runs for N times and so does the loop with j, so that gives us O (N^2) there. For later, the loop with A runs for O (N), while b is a constant loop, so this would be O (1000 * N) → O (N). So, the loops with i, a and b would be O (N^2). Total is O(N^2)

Q8. Our data consists of many <key, value> pairs, which we wish to store into a Hash Table. The Hash function used is key % 10, where 10 is the total number of elements the table can have. What is the result of the Hash Table once the following keys are inserted in order: 23, 50, 13, 77, 43, 83. Assume quadratic probing is used for collision detection.

a)

50		43	23	13			77		83

b)

50		83	23	13			77	43	

c)

50	83		13	23			43	77	

d)

50		43	23	13			77		

e)

50			23	13			77	43	

Ans: h(23) → 3, h(50) → 0, h(13) → 3, but this is occupied so next location would be (h(13) + 1 * 1) % 10 → 4, h(77) → 7, h(43) → 3, but we have a collision here and then also at the next location which has 77. Hence, location after is: (h(43) + 3 * 3) % 10 → 2.

Similarly, for 83, we have 3 clashes and hence location after is (h(83) + 4 * 4) % 10 → 9.

Q9. Consider a Hash Table of size N (N > 1) to which 2 * N keys map to. This Hash Table uses 'Separate Chaining' for collision resolution. We are suddenly given more budget and are allowed to increase the size of our Hash Table to 2 * N to speed things up. However, this means rehashing values from the 1st table into a 2nd one. What is the worst-case time complexity for the most efficient algorithm to achieve this?

a) O (N)

b) O (N log N)

c) O (log N)

d (O N^2)

e) O (N^3)

Ans: The worst case happens if all keys are mapped to the same slot in both the old and new hash tables. In this case, it takes O (N) to look through the full list and also O (N) for the insert into the new table as each element will get added to the linked list due to collision. Hence, worst-case would be O (N^2).

Q10. You have your data stored in a Linked List (size of list = N, N > 1) and now you wish to sort that data. Which sorting algorithm would you use to ensure that your algorithm is efficient in its worst-case complexity? You should assume that no other data structures other than the Linked List are used

a) Mergesort

b) Bubblesort

c) Quicksort

d) Insertion sort

e) Selection sort

Ans: Mergesort. It gives us O(N log N). In lists due to pointer manipulation, we also do not need a lot of extra space. Quicksort has a worst-case complexity that is worse and also random access on Linked List is not easy.

Analysis Questions [12 marks]

[Q11/Q12.] Algorithm Z below is an algorithm that correctly sorts an array A of size n.

```
// Start with the largest gap and work down to a gap of 1
// Similar to insertion sort but instead of 1, gap is being used in each step
gaps = [10, 4, 1]

for each (gap in gaps) {
  // Do a gapped insertion sort for every elements in gaps
  // Each loop leaves A[0...gap-1] in gapped order
  for (i = gap; i < n; i += 1) {
    // save A[i] in temp and make a hole at position i
    temp = A[i]
    // shift earlier gap-sorted elements up until the correct location for A[i] is found
    for (j = i; (j >= gap) && (A[j - gap] > temp); j -= gap) {
      A[j] = A[j - gap]
    }
    // put temp (the original A[i]) in its correct location
    A[j] = temp
  }
}
```

Note: If gap =1 then Algorithm Z becomes Insertion Sort.

Claim: Algorithm Z is a stable sorting algorithm. (?m)

Q11 [2 marks] : Ans: False.

Q12 [4 marks]: A simple counterexample will suffice by considering the 2nd smallest gap of 4.

Let A = [5_a, 1, 5_b, 2, 3, 4]

For gap = 4, we consider every index which is a 0 and a multiple of 4, i.e. 0 and 4. Since A[4] less than A[0], we can insert A[4] = 3 before A[0] = 5_a and "shift" it back by 1 slot from 0 to 4. This results in A becoming [3, 1, 5_b, 2, 5_a, 4].

For gap = 1, the algorithm reduces to running the insertion sort algorithm. This results in the sorted output sequence of [1, 2, 3, 4, 5_b, 5_a].

This is in fact the Shellsort algorithm.

[Q13/Q14.] Suppose you have a random list of people standing in a queue. Each person is described by a pair of integers [h, k], where h is the height of the person (in feet) and k is the number of people in front of this person who have a height greater than or equal to h. We also call the second number an index. Given an unordered list of pairs, Ying has been tasked to reconstruct the queue. Note: *assume that the input data is correct*.

Example (some people are tall, say basketball players at 7 feet):

Input:
[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]

Output:
[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]

In the output list, [6,1] means the person is 6 feet tall and 1 is the number of taller or equally tall people standing in front of that person in the queue (which is the [7,0] person). [5,2] means there are 2 equally or taller people in front of that person.

After some thinking, Ying comes up with the following algorithm:

Step 1: Sort the people based on heights in descending order. If they have the same heights, sort them based on their index in ascending order.

In the example, after sorting, we have:
[7,0], [7,1], [6,1], [5,0], [5,2], [4,4]

Step 2: Add each person into the output list one by one. Specifically, insert them into the (growing) list at position k. Note, the index in the output list starts from 0.

Example:
[7,**0**]
[7,0], [7,**1**]
[7,0], [6,**1**], [7,1]
...

Q13 [2 marks]: Claim: Ying's algorithms solves this problem: True / False

Q14 [4 marks]: Your rationale for your True/False answer is:

a. Ying's algorithm does not solve this problem because the same index k may appear multiple times.
b. Ying's algorithm solves this problem, but it is not the most efficient solution. There exists a better solution that doesn't require sorting.

c. Ying's algorithm does not solve this problem because an index may be out-of-bounds.
d. Ying's algorithm solves this problem because it is exactly similar to adding the tallest group first, and then the second tallest group (we are adding them one by one in each group) and so on. k is the index where each person should be at the time of insertion.
e. Ying's algorithm does not solve this problem because an index may be in the right position at first, but with each new insertion the indices to the right become incorrect.

Structured Questions [48 marks]

[Q15/Q16:] You have a singly-linked list of numbers. We want to rotate the linked list in a circular fashion by shifting n elements to the right.

Consider the following linked list; 1 -> 2 -> 3 -> 4 -> 5 -> null

After shifting 2 elements to the right circularly we get, 4 -> 5 -> 1 -> 2 -> 3 -> null

Q15 [marks 10]: Design an algorithm to circularly right-shift n elements of a given linked list L. You may assume that n < length(L).

Ans:

```
1. If head is NULL or head.next is NULL: # base case
2.     return head
3. current = head
4. length = 1
5. While current.next is not NULL: # find length and last node of L
6.     current = current.next
7.     length = length + 1
8. current.next = head # Connect the last element to the head, making it circular
9. For i from 1 to length – n - 1: # find the new start
10.        current = current.next
11. new_head = current.next # designate the new start
12. current.next = null # remove connection from new tail
13. return new_head
```

Q16 [marks 6]: We can now make use of the above circular-right-shift-by-n algorithm to reverse a linked list as shown below.

```
1.  If head is NULL or head.next is NULL:
2.      Return head
3.   original_head = head
4.   Reversed_head = null
5.  For i from 1 to length:
6.      head = circular-right-shift-by-n(head, 1) # Right-shift by 1
7.      new_node = Node(head.value)
8.      new_node.next = reversed_head
9.      reversed_head = new_node
10.     head = head.next
11. Return reversed_head
```

The expected behaviour is to reverse a given linked list and create a new linked list with elements in the reversed order.

Example, input 1->2->3-> null should provide 3 -> 2-> 1 -> null as the output.

Does this algorithm work as expected? Explain.
What is the run time of this algorithm?

Ans: The algorithm produces a linkedlist in the same order as the original linkedlist. This is because, right shift by one before picking the head always ensures the last element is picked. And we keep adding that to the head of our result. Giving a list in the same ordering.
Runtime of the algorithm is O(n^2)

[Q17/Q18.] Oizne Mak is the librarian of The Great Magic Library of Cossun. He has to manage a total of *n* books in the library. Each book has a title and a magic number. The magic number of each book is **unique** and in the range 1 to *n*.

At the end of each day, Oizne Mak must rearrange all the books in the library onto the shelves in ascending order of their magic numbers. He must then write the titles of each book in that order onto a piece of paper to be kept for record-keeping. He has requested your help to find a way to do this more quickly so he can go home earlier.

You may assume that the books are initially stored in an array, and that the record-keeping (i.e. printing of book titles) can only be done after the books are rearranged.

Q17 [marks 3]: (a) Design an algorithm to rearrange the books and complete the record-keeping.

Ans: Sort the books by magic number with any sorting algorithm covered in class other than radix sort (values of *n* are not bounded). Then iterate through the array in the sorted

order to print the titles of the books. The algorithm has a time complexity of $O(n \log n)$ or $O(n^2)$ depending on the sorting algorithm used.

Alternatively, we can use a hash table, storing the (magic number, book) as the key-value pairs and use this to do the sorting. This gives us an average time complexity of $O(n)$.

Q18 [marks 12]: (b) The playful magician Ecneics likes to play pranks on Oizne Mak. This time, he has casted a spell that makes him forgetful. Now, Oizne Mak is unable to remember things that happened even just a minute ago!

Design an algorithm to rearrange the books and complete the record-keeping, but with a time complexity of $O(n)$ and space complexity of $O(1)$.

Ans: We consider the idea of a hash function where the hash value of each book will be its magic number itself, since all the magic numbers are unique values in the range 1 to n.

```
12. Let A[1...n] denote the books in the library.
13. for i in 1:n
14.     Let j = magic number of book at A[i]
15.    while i != j
16.         swap A[i] with A[j]
17.         Let j = magic number of book at A[i] (after the swap)
18.    end
19. end
```

Then iterate through the array in the sorted order to print the titles of the books.

The above algorithm works because the magic numbers are unique, thus each book must be placed at only one unique location in the array after it is sorted (i.e. there should be no collisions). When we stop swapping at the end of the while loop, A[i] will have a book with magic number i.

[Q19/Q20.] After spending a lot of time in The Great Magic Library of Cossun, Oizne Mak has managed to learn some magic of his own. He has learnt a spell called REVERSE[*a, b*], and he wishes to practice it at home in his room. He lays out *n* cards, each labelled with an integer, on the table in a straight line.

Each invocation of the spell requires him to decide on 2 values *a* and *b*, where $a \leq b$. The 2 values form an interval [*a*, *b*], which means the cards will be rearranged for the positions $a, a-1, \ldots, b-1, b$ in **reverse** order. An example is shown below.

A = [1, 3, 10, 8, 9, 6, 2]

Oizne Mak casts REVERSE[2, 3]
Oizne Mak casts REVERSE[4, 6]

A is rearranged into [1, **10, 3, 6, 9, 8**, 2]

You may assume that *n*, *k*, *a*, *b* are all positive integers and satisfy the inequality $1 \leq a \leq b \leq n$.

It is possible for multiple casts of the REVERSE[*a, b*] spell to have overlapping intervals. When two intervals overlap, they will merge into a larger interval. Without loss of generality, consider two intervals [*a*, *b*] and [*c*, *d*] where $a \leq c$.

- If $d \leq b$, then we merge the intervals to form the new interval [*a*, *b*], i.e. the interval [*c*, *d*] is completely contained in the interval [*a*, *b*].
- If $c \leq b$, then we merge the intervals to form the new interval [*a*, *d*].
- Otherwise, the intervals do not overlap. We keep both [*a, b*] and [*c, d*]

A = [1, 3, 10, 8, 9, 6, 2, 7, 13, 5]

Oizne Mak casts REVERSE[3, 6]
Oizne Mak casts REVERSE[4, 8] -> merges with [3, 6] into [3, 8]
Oizne Mak casts REVERSE[1, 2]
Oizne Mak casts REVERSE[5, 6] -> merges with [3,8] into [3, 8]

A is rearranged into [**3, 1, 7, 2, 6, 9, 8, 10**, 13, 5]

Every practice session, Oizne Mak will cast the REVERSE[*a, b*] spell a total of *k* times for different values of *a* and *b*. However, he is not sure that he has casted the spells correctly. He has enlisted your help to help him verify what the correct final sequence of cards should be.

Q19 [marks 6]: (a) Assume the intervals for the REVERSE[*a, b*] spells do not overlap and that the cards and intervals are each stored in an array. Design the most efficient algorithm to print the final sequence of cards. You are not allowed to use a hash table. Violation of the restrictions will result in no marks being awarded.

Ans: For every interval [a, b], we will swap the elements (a, b), (a+1, b-1), ... (i, j) and stop when i >= j. This will reverse the elements in the interval [a, b].

```
1.  Let A[1...n] denote the cards on the table.
2.  Let S[1...k] denote the intervals for each spell.
3.  for each [a, b] in S
4.     Let i = a, b = j.
5.        while i < j
6.           swap A[i] with A[j]
7.           i = i + 1
8.           j = j – 1
9.        end
10.    end
11. end
```

The algorithm above has time complexity $O(n + k)$.

Q20 [marks 11]: (b) The intervals for the REVERSE[*a, b*] spells can now overlap. Assume that the cards are stored as integers in nodes of a singly-linked list and the intervals are stored in an array. Design the most efficient algorithm to print the final sequence of cards. You are not allowed to modify the integer values of the nodes of the linked list, and you are not allowed to use a hash table or any additional array. Violation of the restrictions will result in marks being deducted.

Ans: First, sort the intervals by increasing order of the first interval value with any sorting algorithm covered in class other than merge or radix sort (both require additional array). Next, we merge the intervals by going through the sorted order.

```
1.  Let S[1...k] denote the intervals for each spell sorted by first value.
2.  Let j = 2.
3.  Let curr = S[1].
4.  while j <= k
5.     Let next = S[j].
6.     if next.b <= curr.b    # d <= b
7.        do nothing
8.     else if next.a <= curr.b    # c <= b
```

```
9.        Set curr =  [curr.a, next.d]
10.   else    # no overlap
11.       Add curr to output
12.       Set curr = next.
13.   end
14.   j = j + 1
15. end
```

The output can be stored in a Linked List or just back into the original array of intervals (with an additional variable to keep track of “last inserted index”). Now we have a list of “merged” intervals.

For the reversal, we use a counter variable to keep track of the current position of the node. For the first “merged” interval [a, b], traverse the linked list until we reach the (a - 1)th position node in the linked list. We can now push the nodes from position a to b into a Stack in that order, and keep track of the (b + 1)th position node. We then pop the elements one by one from the Stack and add it to the back of the “front” part of the linked list which will give the reversed order and connect it to the “back” part of the linked list. We can now continue from position (b + 1) and repeat for the subsequent intervals.

The algorithm above has time complexity *O(n + k^2)* and space complexity *O(n + k)*, or space complexity *O(n)* if not using an additional linked list for the “merged” intervals.