CS2040 2023/2024 Sem 2 Midterm Questions

MCQ: 10 Questions, 30 Marks (3 Marks each)

1. What is the runtime complexity of classic quicksort (as taught in lecture) when all elements in the array have the same value?

a. O(log N)
b. O(N)
c. O(N log N)
d. $O(N^2)$

2. An array A is being sorted using the quicksort algorithm discussed in the lecture. After running for some time, the program was terminated and A becomes [2, 1, 3, 5, 4, 6, 7]. Which of the following array is NOT a valid initial ordering of A?

a. [2, 4, 3, 5, 1, 6, 7]
b. [7, 4, 3, 5, 1, 6, 2]
c. [6, 5, 1, 2, 7, 3, 4]
d. [6, 3, 1, 7, 5, 4, 2]
e. None of the above

3. What is the time complexity of f(N) in terms of N?

```
public static void f(int N) { // pre-cond : N is positive
  int i = 0, jump = N/2;
  while (i < N) {
    i += jump;
    for (int j = i; j < i + jump; j++) {
      System.out.print("*");
    }
  }
}
```

a. O(log N)
b. O(N)
c. O(N log N)
d. $O(N^2)$
e. O(N!)

4. What is the time complexity of g(N) in terms of N?

```
public static int g(int N) {
  int x = 0, y = 0;
  for (int i = 1; i < N; i *= 2) {
    for (int j = 1; j < N; j *= 2) {
      for (int k = 0; k < i; k++)
        x++;
      for (int m = 1; m < i; m *= 2)
        y += x;
    }
  }
  return x-y;
}
```

a. O(N)
b. O(N log N)
c. $O(N (\log N)^2)$
d. $O(N (\log N)^3)$
e. $O(N^2)$
f. $O(N^2 \log N)$
g. $O(N^3)$
h. $O(N^4)$

5. The following 2D array is used to simulate circular linked lists *cll1*, *cll2*, *cll3*.

   Its first row stores character data and second row stores the index of the next reference in the array (consider this to be 0-based indexing).

Data	A	B	C	D	E	F	G	H	I	J	K	L
Next	8	0	3	6	11	10	9	4	5	2	1	7
Index	0	1	2	3	4	5	6	7	8	9	10	11

   The tail references of *cll1*, *cll2* and *cll3* are in index 7, 8 and 9 respectively.

   With a function **getDataAtM(int tailIndex)** that takes in an integer **tailIndex** representing the tail and returns the data at the **M**-th position (where **M** = floor(size/2)), where position is 0-indexed (i.e. tail is at (size-1)-th position)

   What is the output of **getDataAtM(7)**, **getDataAtM(8)**, **getDataAtM(9)**?

a. E, K , C
b. E, K, D
c. E, K, G
d. L, B, C
e. L, B, D
f. L, B, G

6. In lecture 5 of the module, we introduced the DoublyLinkedList where each DListNode contains two references that points to the previous and next node, and contains an integer element.
   Below is an augmented implementation of binary search on a DoublyLinkedList of integers.

```
public static int binarySearch(DoublyLinkedList list, int len,
int x) {
  int mid, low = 0;
  int high = len - 1;

  while (low <= high) {
    mid = (low + high) / 2; // integer division

    int integerAtMidIndex = list.getItemAtIndex(mid);

    if (x == integerAtMidIndex) {
      return mid; // Element found
    } else if (x > integerAtMidIndex) {
      low = mid + 1;
    } else {
      high = mid - 1;
    }
  }
  return -1; // Element not found
}
```

   The method getItemAtIndex(int index) returns the integer at the specified position in the DoublyLinkedList.
   You can assume that the getItemAtIndex method traverse from the front if index < size(list) / 2, from the back otherwise.

   What is the worst-case time complexity of the function when it runs on a sorted DoublyLinkedList of size N? (choose the best option).

a. O(log N)
b. O(N)
c. O(N log N)
d. $O(N^2)$

7. A double hashing scheme is used to create a hash table of size 13. The primary and secondary hash function are h1(k) = k mod 13 and h2(k) = (k mod 19) + (k mod 3), respectively. Assuming that the index returned by the first two probes are already filled, what would be the index of the third probe for k = 30.

a. 4
b. 3
c. 2
d. 0
e. 7
f. None of the above

8. You are given the following three hash functions to implement a hash table of size 10:

   Given that the input to the hash table contains integers from 0 to 100, which of the hash functions below distribute the keys uniformly over buckets 0 to 9?

   i. h1(k) = (13*k) mod 10
   ii. h2(k) = $(7*k^2)$ mod 10
   iii. h3(k) = k^3 mod 10

a. i only
b. ii only
c. iii only
d. i and ii
e. ii and iii
f. i and iii
g. None of the above

9. Given a Stack S which contains the following unique integers from top (leftmost) to bottom (rightmost)

   1,2,3,4,5

   If S can only re-order its content (without using recursion) by making use of another initially empty queue Q in the following way:

   1. Can only pop an integer from S and enqueue it into Q
   2. Can only dequeue an integer from Q and then push it into S
   3. 1 and 2 may perform repeatedly to transfer items to and fro between S and Q.

   Which of the following options is not a valid re-ordering of S (Choose all options that apply). If all are valid, choose option e.

a. 5, 4, 3, 2, 1
b. 4, 5, 3, 2, 1
c. 5, 3, 2, 4, 1
d. 1, 2, 5, 4, 3
e. all the above are valid

10. A doubled ended stack is a stack where you can both push and pop from the top or the bottom of the stack.
    Let's call these four operations push_top(), pop_top(), push_bottom() and pop_bottom() respectively.
    Which of the following data structures can be used to implement a doubled ended stack such that all the above 4 operations runs in worst or amortized case O(1) time.

    i. 1 or more array(s)
    ii. 1 or more singly linked list(s)
    iii. 1 or more doubly linked list(s)
    iv. 1 or more hash table(s)

    Note: if the option has 2 or more of the above data structures, they are to be used independently and not in combination to implement the double ended stack.

a. i only
b. iii only
c. i and iii
d. iii and iv
e. i, ii and iii
f. all of i, ii, iii and iv

Analysis: 3 problems, 15 marks (5 marks each)

Question 11 and 12 refers to the problem given below:

11. **[5 marks]** In the array-based implementation of a list/stack/queue, whenever the array is full, the strategy is to create a new array that is double the size of the current array. This results in an amortized cost of O(1) time for addBack()/push()/offer() of list/stack/queue respectively.

    Claim: If we change the array enlarging strategy as follows, the amortized cost of addBack()/push()/offer() is still O(1) time.

    New enlarging strategy:

    Start with an array of size $1^2 = 1$, when it is full create an array of size $2^2 = 4$, once it is full create an array of size $3^2 = 9$ etc.

    That is, at the x^{th} time you need to create an array, create an array of size x^2 (x starts from 1).

    Select the option that is most correct about the claim and gives the most correct explanation.

a. True because this is simply the doubling strategy itself
b. True because if you make n addBack()/push()/offer() calls you will need to create O(logn) arrays of size 1,4,9... etc and fill each of them up which result in the same O(1) time amortized cost.
c. True because if you make n addBack()/push()/offer() calls you will need to create O(sqrt(n)) arrays of size 1,4,9... etc and fill each of them up, which result in the same O(1) time amortized cost.
d. False because if you make n addBack()/push()/offer() calls you will need to create O(sqrt(n)) arrays of size 1,4,9... etc and fill each of them up, which result in O(sqrt(n)) time amortized cost
e. False because if you make n addBack()/push()/offer() calls you will need to create O(n) arrays of size 1,4,9... etc and fill each of them up, which result in O(n) time amortized cost.
f. False because if you make n addBack()/push()/offer() calls you will need to create O(logn) arrays of size 1,4,9... etc and fill each of them up, which result in O(logn) time amortized cost.

Question 12 to 13 refers to the following problem

12. **[2 marks]** Consider the following alternative implementation of insertion sort:

```
public static int binarySearch(int[] arr, int start, int end,
int target) {
  int low = start;
  int high = end;
  while (low < high) {
    int mid = (low + high) / 2; // integer division
    if (arr[mid] < target) {
      low = mid + 1;
    } else {
      high = mid;
    }
  }
  return low;
}

public static void insertionSort(int[] arr) {
  for (int i = 1; i < arr.length; ++i) {
     int correctIdx = binarySearch(arr, 0, i, arr[i]);
     int toInsert = arr[i];
     for (int j = i; j > correctIdx; --j)
       arr[j] = arr[j-1];
     arr[correctIdx] = toInsert;
   }
 }
```

Instead of checking the elements from the back, we exploit the fact that the array from 0 .. i-1 is already sorted. We can use binary search to find the correct index of the element that we want to insert in O(log N) time.

Claim: This version of insertion sort is stable, and its best case and worst case time complexity is O(N) and O(N log N), respectively.

a. True
b. False

13. **[3 marks]** Give your rationale for your answer to the previous question.

Question 14 to 15 refers to the following problem

14. **[2 marks]** Given the following function (which compiles, runs, terminates and does NOT crash):

```
public static int pairUp(int[] arr) { // arr has length N
  ArrayList<Integer> list = new ArrayList<Integer>();

  // first set of nested loop
  for (int i = 0; i < arr.length; i++)
    for (int j = 0; j < arr.length; j++)
      list.add(arr[Math.min(i, j)]);// adds to back of the list
                                    // assume add is worst case
                                    // O(1) time
  int total = 0;
  // second set of nested loop
  for (int i = 0; i < list.size(); i++)
    for (int j = 0; j < list.size(); j++)
      total += list.get(i) * list.get(j);

  return total;
}
```

Claim: The time complexity of pairUp is $O(N^2)$, since the second set of nested loops is done after the first set of nested loops complete

a. True
b. False

15. **[3 marks]** Give the rationale for your answer to the previous question.

Structured Questions: 4 questions, 30 marks

This section is worth 30 marks. Answer all questions.

Write in **pseudo-code**.

Any algorithm/data structure/data structure operation not taught in CS2040 must be described, there must be no black boxes. For the written questions partial marks will be awarded for correct answers not meeting the time complexity required.

16. **[5 marks]**

    A palindrome is a string of characters that reads the same forward and backward. E.g. "php" and "anna" are palindromes, while "pie" and "java" are not.

    There are multiple ways to store a string of N characters in a List ADT.
    For each of the N characters, we will attempt to insert it to the List in sequence from first to last and each one will be inserted to the end of the list.

    Which of the data structure(s) given in the options can be used to implement the List ADT (only 1 list) such that after inserting the string of characters, it also allows us to come up with a non-recursive algorithm that runs in O(N) worst case time complexity, and uses worst case O(1) extra space to check whether the string is a palindrome?

    Note: The algorithm to check if the string is a palindrome is **not allowed** to make modifications to the data structure.

    Choose all options that apply.

a. Array
b. Basic Linked List
c. Tailed Linked List
d. Circular Linked List
e. Doubly Linked List

17. **[5 marks]** Mr Zak wants to design a data structure called fastList to store the most recently used emojis in his chat app. The objective is to read from and write into the fastList in O(1) average or worst case. The user first searches for emojis in fastList. If the emoji is already in fastList, it would be accessed from the fastList and also placed to the front of fastList via a **get** operation. If an emoji is not in fastList, it needs to be accessed from slowList. Once an emoji is accessed from slowList, it is added to the front of fastList via a **put** operation. If the fastList does not have the space for this new emoji, the least recently used emoji would be evicted from the fastList so that the new emoji can be added (this is all part of the **put** operation). The size of the fastList is fixed.

The read and write mechanisms of fastList are illustrated with an example below.

Assume that the size of the fastList is 4 and each emoji is assigned a unique integer. User accesses the emojis in the following order: 1,2,3,1,5,2,6

The fastList interface has two methods as mentioned: get(int i) and put(int i), where i is the emoji number.

fastList: (initially, it is empty)

Access emoji-1 from fastList using get(1). emoji-1 is not present in the fastList. It is read from slowList and it will be added to the front of fastList via put(1).

1			

Access emoji-2 from fastList using get(2). emoji-2 is not present in the fastList. It is read from slowList and it will be added to the front of fastList via put(2).

2	1		

Access emoji-3 from fastList using get(3). emoji-3 is not present in the list. It is read from slowList and it will be added to the front of fastList via put(3)

3	2	1	

Access emoji-1 from fastList using get(1). It would be relocated to the front of fastList via put(1).

1	3	2	

Access emoji-5 from fastList using get(5). emoji-5 is not in fastList. It is read from slowList and it will be added to the front of fastList via put(5).

5	1	3	2

Access emoji-2 from fastList using get(2). It will be relocated to the front of fastList via put(2).

2	5	1	3

Access emoji-6 from fastList using get(6). emoji-6 is not in the fastList. It is read from slowList and it will be added to the front of fastList. Item 3 is the least recently used and it would be evicted from the fastList.

6	2	5	1

As shown above, the most recently used emoji would be at the front of the list and least recently used emoji would be at the back of the list. Select the most appropriate method for the Zak to implement the fastList such that its **get** and **put** operations can each be done in average or worst case O(1) time.

a. Double linked list to store the emojis (their number) and a hash table containing <key,value> entries where the value is a reference to a node of the double linked list.
b. Singly linked list to store the emojis (their number) and a hash table containing <key,value> entries where the value is a reference to a node of the singly linked list.
c. Tailed linked list to store the emojis (their number) and a hash table containing <key,value> entries where the value is a reference to a node of the tailed linked list.
d. An array to store the emojis (their number) and hash table containing <key,value> entries where the value is an index of the array.
e. A circular linked list to store emojis (their number) and a hash table containing <key,value> entries where the value is a reference to a node of the circular linked list.
f. A circular array to store emojis (their number) and a hash table containing <key,value> entries where the value is an index of the circular array.

18. **[10 marks]** John's niece has finished playing a game involving **n** balls of different numbers which are unbounded floating point values.
Being mummy's good girl, she is placing the balls back into their container which is a long rectangular box with **n** (**n** >= 1000) holes placed along the entire length of the box from one end to the other end. Each hole is marked with the corresponding number of the ball that should go into it. The holes are arranged from smallest ball number to largest ball number from the leftmost hole (call this hole 1) to the rightmost hole (call this hole **n**), that is, the holes are arranged in ascending order of ball number. Eg if there are 3 balls are numbered 1.13, 3.112, 2.77, then the holes from left to right will be the holes for balls 1.13, 2.77, 3.112 respectively.

Just as she finishes placing all the balls correctly, her baby brother runs past and kicks the box so that the balls jump out of their hole and are now possibly shifted to another hole!

After inspecting the container, John discovers that the balls in every $\mathbf{m}^{th}$ hole (**m** = **n**/10) is still in the correct position (i.e hole 1***m**, hole 2***m**, ...).
Call these *unmoved holes*.

For block of holes fulfilling either one of the following conditions:
1. holes between 2 consecutive *unmoved holes*
2. holes between the 1st hole and the 1st *unmoved hole* (if they are not the same)
3. holes between the last *unmoved hole* and the last hole (if they are not the same)

There can possibly be at most 2 holes among holes in each such block with their balls swapped.

Help John to devise the most efficient algorithm to place all balls correctly back to their respective holes.
You can assume that number of the balls are unbounded floating point values in the range 1.0 to 100.0 respectively.
The container is represented by an array **A** of size **n**, with each slot representing a hole (slot 0 is hole 1, slot 1 is hole 2 etc) and the number of the ball in the hole is stored in that slot. The value of **m** is also given to you.

19. **[10 marks]** Ivan took **n** photographs on a trip. You are given **n**, as well as an array **r** of each photograph's rating in sequence (the ratings may be arbitrarily large integers).

    You are to write an algorithm, to most efficiently recommend, for each photograph **j** from **j** = 1, **j** = 2, **j** = 3, ... to **j** = (**n-1**) in that order, all the photographs **i** taken before photograph **j** that have strictly lower ratings than **j** , i.e. **i** < **j** and **r[i]** < **r[j]**, for which photograph **i has not already been recommended**.

    For each recommended pair of photographs, output (**j**, **i**) ordered by non-descending **j**, using ascending **i** to break ties.

    e.g. for input **r** = [8, 6, 3, 2, 1, 3, 1, 7, 4], the output should be:

    (5, 3)
    (5, 4)
    (7, 1)
    (7, 2)
    (7, 5)
    (7, 6)

    The photographs at indexes 3 and 4 are recommended in the photograph at index 5, but will not be recommended again with the photograph at index 7.