Question 1

Overall students performed well on this question.

1.1

- This subquestion was marked strictly. Students had to mention that the missing bit was the 5th index. If they did not use 1-based indexing, or did not mention the index of the missing bit and instead said that the correct bit was 0, it was marked incorrect.
- No partial marks were given for this subquestion.
- Some students gave basic algorithms to find the missing bit. These were marked incorrectly if the actual missing bit's index was not stated.
- Keep in mind that the subquestion was only worth 2 marks. This basically means that a sentence or two at most is sufficient.

1.2

- Some students gave the range 1-9 which we gave leniency to.
- If students used 0-based indexing in part 1, and hence also got this part wrong (i.e. said the error was in the range 11-20), carry forward was applied.

1.3 Most students had the correct idea for this part, and built off their solutions to 1.1 and 1.2. Some common errors:

- Algorithms that could not handle the first or last index being the missing bit.
- Algorithms that did not have a clear halting condition on the recursion that showed how the missing bit would be found.
- Not explicitly justifying the time complexity. Students who simply stated a time complexity without any justification were marked down.
- Not justifying the correctness of key parts of the algorithm. For example, if you defined a function to determine whether a bit was correct, you had to explain why that function was appropriate.

Question 2

This question was an excellent way of testing the fundamentals associated with data structures and sorting algorithms. However, overall there were some items that appeared across the cohort:

- Many students did not name their sorting algorithms nor understand the worst-case performance of their chosen sort e.g. bubblesort worst case is **not** $O(n \log n)$
- As this question was focusing on time complexities, simply stating that your algorithm was *"$O(n^2)$ as required"* is not sufficient justification. Each step of the algorithm has a cost associated with it and this should be summarised as the end of the algorithm.
- Many students did not justify why their algorithm was correct e.g. how all possible outcomes were assessed and that the output was valid.

2.1

- Many students attempted to perform a binary search on unsorted data. Binary search works only in $O(\log n)$ if the data is sorted.

- Almost all students forgot to keep track of the original indices in the input array A and returned the indices from a sorted array. When reading the questions always keep inputs and outputs in mind!

2.2

- There were many instances where students would state *"use a hash map"* or, *"put all values into a hash map"*. Hash maps by design are designed for key/value pairing and thus when used, it must be explicitly stated how the hash map is being set up.
- Most students assumed that hash map search/access would always occur in constant $O(1)$ time. This is not the case if collisions aren't handled appropriately. It is for this reason that almost all students that used their solution in 2.2 for 2.3 did not get any marks because the worst case of accessing a hash map in $O(n)$ within a nested loop results in a $O(n^3)$ time complexity.

2.3

- Where students fell short on this part of the question was being able to justify why in the **worst case** that the algorithm satisfied the time complexity requirement.

Question 3

Part 3 and part 4 of this question were slightly tricky and most students could not claim many marks for them. On average students got 8/20 - 9/20 marks for this question.

3.1

- No partial marks were given for this subquestion. If the student has provided the correct answer, 14, 2 marks were awarded.

3.2

- The set of conditions stated by some students were incomplete, not necessarily true or were unnecessarily complex.
- The problem statement assumes that the student is given the pair of indices (i, j). Hence, students are expected to specify the conditions applicable to i and j. There is no requirement in looking for other conditions that involve n, ℓ or r.

3.3

- This subquestion expects students to find the values of $g(i, j)$ for all values of $i < j$ (not just a single value of i).
- Some students used hash tables to solve this problem. It seemed like students assume hash tables to be the universal solution for $O(1)$ lookups. In this question, it makes no difference in having a simple array as opposed to having a hash table. Hence, we expected the students to specify the key set or the hashing function that they use to obtain the expected timing bounds as a measure of preventing blind use of hash tables.
- Some students seemed to be unaware of the level of detail expected in algorithm design subquestions. Students cannot vaguely suggest a scheme and expect full marks to be awarded. Also, these subquestions expect to have a justification of correctness and time complexity (not necessarily as separate sections).

- As stated in the Assignment 1 worksheet, pseudo-code can only be included as a supplement to an otherwise complete explanation written in plain English (including mathematical notation).

3.4

- Some students managed to find an $O(n)$ solution for this subquestion using monotonic stacks. Congratulation!
- Some students seemed to not differentiate merge sort algorithm and divide and conquer approach. Students should note that merge sort is a sorting algorithm that uses divide and conquer approach and unless it is about sorting, it is unlikely that merge sort should be mentioned. For instance, if the student mentions that "For this problem, we use merge sort algorithm ...", it is implied that the student is sorting the given array.
- One common mistake found in answers is in the step where the student splits the array to generate the subproblems. Some students used the index of the maximum value to split the array rather than the mid point. Using the maximum value does not guarantee that the recurrence will have $\log n$ levels and therefore the time-bound cannot be guaranteed to be $O(n \log n)$. For example, assume an array A where the elements are sorted in decreasing order. Initially, the array would get split around $A[1]$ and then it will sequentially get split through $A[2], A[3], A[4], ..., A[n]$ making $O(n)$ many recurrence levels.

Question 4

Generally students did well on this question.

4.1

- Generally well done.

4.2

- Generally well done.

4.3

- Most students who did not achieve full marks had a vaguely correct idea. Most lost marks due to insufficient/incorrect justifications for correctness. Some lost marks for insufficient/incorrect justifications of their stated time complexity.
- Time complexity justifications were not expected to be perfect, but did expect the reasoning for the provided time complexity be made apparent. Full marks were generally achieved one of 2 ways.
  Either the master theorem was applied, with some brief justification of the overhead/remaining search space. E.g. each recursion asks 3 questions to the friend, done in constant time. Every recursion also halves the search space. Therefore applying master theorem ...
  Or the reasoning was explained. E.g. at each level of the recursion, we ask 2 questions, possible in constant time. We can only recurse $\log n$ times as the search space is halved each time, which can only occur $\log n$ times until we reach a singleton array.
- Correctness justifications were not expected to be perfect, but did expect the reasoning for the correctness of the algorithm be made apparent. Generally, those who attempted 4.3 separately from 4.4 attained all the marks for correctness justifications. Most marks lost due to insufficient justifications of correctness in 4.3 were in submissions which also attempt to address 4.4 with the same response. This is explained further below.

- Algorithms must be explained to a degree such that we can at least understand what your algorithm intends to do, and how it intends to do it. Some submissions are very vague in their explanation. E.g. not specifying that we only recurse on half the array/not specifying which half.
- As stated in the Assignment 1 worksheet, pseudo-code can only be included as a supplement to an otherwise complete explanation written in plain English (including mathematical notation). Translating pseudo-code to steps in prose, without further elaboration or explanation is also insufficient for the same reasons pseudo-code is insufficient.

4.4

- Most students who attempted and did not achieve all the marks for 4.4 either misunderstood/ignored the added condition or did not sufficiently justify the correctness of their algorithm.
- Many students simply searched for the largest element and employed the questions from 4.2. This clearly did not satisfy the added condition, the question should be read more carefully.
- Many students did not attempt to justify the correctness of their algorithm. Those who achieved full marks for this section generally did so either by explicitly justifying why their algorithm was correct, e.g.

  > Our array p (potential second largest elements) now contains the answers to all of our queries which did not yield the maximum element of the array. As the union of the intervals of these queries covers the entire array excluding the index of the maximal element, the maximum of p must be the maximum of the array, not including the largest element.

  or implicitly justifying why their algorithm was correct, e.g.

  > We now save the maximum over the interval of the array which does not contain the maximum of the entire array into a separate array p. Now either p contains the second largest element already, or the second largest element is in the interval which contains the largest element. Therefore, we recursively seach that half . . .

  Note how both of these contain logic equivalent to "the output is the maximal of a set which must contain the second largest element but not the largest".