COMP2119
Assignment 2 Feedback

Assignment 2 Q1

Part (a)

1. Most students could explain the details and time complexity of the algorithm correctly.
2. Quite a number of students did not explain how they came up with the time complexity of the algorithm.

Part (b)

1. Most students were able to solve the problem by using binary search.
2. A lot of students forgot to consider the boundary case k = 0.
3. Some students mistakenly treated k as the list index where A[k] > A[k+1]. In fact, k represents the number of scripts read by Ben.
4. Some students wrote down the wrong index for the upper / lower bound of the search space.
5. Some students misunderstood the meaning of "asymptotically faster". An asymptotically faster algorithm means it has a lower growth rate (in terms of running time) as the input size increases. Therefore, an O(n/2) algorithm is not asymptotically faster than an O(n) algorithm, as they have the same growth rate.
6. Quite a number of students did not explain how they came up with the time complexity of the algorithm.

Assignment 2 Q2

Part (a)

1. Most students successfully completed the merging sorted linked lists, by comparing their head elements and adding them to the new linked list.
2. Some students tried to extract elements from all the linked lists, sort them and add them to a new linked list. This approach is correct, but not desirable, as it does not demonstrate the student's understanding of a sorted linked list.

Part (b)

1. Most students successfully gave a correct tortoise and hare algorithm.
2. Some students solved the problem by labelling whether a node is visited in the linked list, which is also a correct and good approach.
3. Some students tried to record all visited nodes, and check out whether a node is in the set of visited nodes when visiting it. This approach is correct, but not desirable. Checking out whether a node is in the visited nodes generally increases the time and memory complexity of the algorithm, and may rely on containers (e.g., set and hash table) and algorithms (e.g., hash function) that the student is not expected to be able to provide at this time.

Assignment 2 Q3

Part (a)

1. Did not take into consideration that there could be different integers having the same highest frequency in the pop function. In such cases, the one among these integers closest to the top of the stack should be popped first.
2. Some students described their algorithm in natural language instead of providing an implementation. Pseudocode or a progamming language should be used instead.

Part (b)

1. The common usage of high level data structures such as std::unordered_map complicated the worst-case time complexity dicussion. std::unordered_map is actually a hash table that can be dynamically resized. Depending on the implementation of std::unordered_map, the worst-case time complexity of inserting a unique integer into std::unordered_map can be O(1) if the size of the hash table used is large enough (e.g., with size N) or O(N) if the hash table size is very small (e.g., many collisions or extending the hash table internally). Such complexity is possibly beyond the scope of this class (actually students can easily avoid such complexity by using pseudocode style HashMap(size=N)).

   Thus, I have loosened the requirement of worst-case time complexity discussion: students only needed to explain why they picked O(1) or O(N) as the worst-case time complexity in their implementation regarding such data structures.

Assignment 2 Q3

Part (b)

2. Mistakenly considered the time complexity of looping through the frequency record hash table (map / array / linked list) to be O(N). Such a table typically takes each unique integer as the key and looping through it actually has the time complexity as O(1) since we have a fixed number of unique integers.

Assignment 2 Q4

Common Issue:

1. When calculating the average number of slots inspected for an unsuccessful search with a given specific hash table, considering the expectation of the number of probe starting from each slot is more accurate. Calculation using load factor is a rough estimation that cannot take situations such as clustering into consideration.
2. Some students seemed to have misused GenAI in solving these questions, which led to answers that are basically repetitions of the questions.

For each sub-question:

(a) 1. Did not notice that we should insert a new element at the head of the linked list in chaining in hash table collision handling. This is to avoid the need to traverse to the end of the linked list, which may cause the time complexity of O(N).

2. Did not realize that a non-empty slot also contains an NIL at the end of the linked list, which leads to incorrect calculation of the average number of slots inspected for an unsuccessful search.

(b) 1. Some students didn't use the provided linear probing function f(i) = 3i but used the default f(i) = i.

(c) 1. Many students only considered one of the two possible cases where k could be even or odd, leading to incorrect calculation of the average number of slots inspected for an unsuccessful search.