Question 1 Asymptotic Analysis

Many students made a mistake in 1(b) and 1(c); they chose only two options (missing Y mostly) instead of W, X and Y.

Question 2 True/False

(f) Many students thought the statement was true and tried to prove it by using the definition of Θ.

Question 3 Short Answer

(a) Many student wrote Θ\((n^2)\) or Θ\((n^2 \log n)\). Some students incorrectly thought that the run-time of the fourth line was Θ\((\log n)\).

Question 4 Sorting Algorithm

(a) Many students wrote \(O(n)\) instead of \(O(n^2)\). It seems they ignored the domain range of the array.

(c) • Many students knew to use Radix Sort, but most did not specify the radix or wrote \(radix = 10\), which does not give and \(O(n)\) run-time. Choosing \(radix = n\) will.
  • Some students wrote merge sort or interpolation sort, but merge sort has run-time \(O(n \log n)\) and interpolation sort does not exist.

(d) Many students assumed the auxiliary space of Count Sort was always \(O(n)\), but the auxiliary space space of Count Sort here is \(O(n^2)\) because of the domain range \([0, n^2 - 1]\).

Question 5 Heaps

Most students did parts (d) and (e) incorrectly.

Question 6 Lower Bound

• Majority of the students did not know what it meant to prove the lower bound complexity of a problem.
• Most students gave a single algorithm that solves the problems but this only gives us an upper bound on the complexity.

• To show a lower bound, students should show that any algorithm that solves this task must perform at least 4 weighings.

• Many students started their incorrect proofs by providing an algorithm that preformed 4 weighings and stating that this is the best possible algorithm for this task. However, to actually justify if it is the best possible algorithm, students still need to prove a lower bound.

• Many students tried to bound the number of leaves based on the number of 3-coin permutations out of 7 coins, which is 210, rather than \( \binom{7}{3} \). Note that the lighter coins do not need to be distinguished from each other.

Comments:
Students should spend time understanding that there is a difference between lower bounding/upper bounding the complexity of a problem and lower bounding/upper bounding the run-time of a specific algorithm that solves a problem. Providing an algorithm to solve a problem gives us an upper bound on the complexity of the problem but to get a lower bound we will need more specialized techniques such as decision trees.

Question 7  AVL Trees: Part1

Many students did not know what the in-order successor was so they deleted the wrong node.

Question 8  AVL Trees: Part2

Many students wrote incorrect balance factors for some nodes.

Question 9  Skip Lists: Part 1

(a)  • Some students only inserted two 8s and did not increase the height. Some students inserted three 8s but forgot to write the top level.

(b)  • Some students did not delete leaf 5.
    • Many students did not read the question carefully and performed deletion on the modified skip list.
    • Some students forgot to decrease the height and had an extra tower.
Question 10  Skip Lists: Part 2

- Some students did not provide a run-time analysis.
- Some students did not consider the case that $A[i].next \neq \text{min element}$ which requires to break the loop otherwise run-time would not be $O(1)$.
- Many students did not specify the unlink process and they just assumed there was a delete function to call.

Question 11  Randomized Algorithms: Part 1

Most students did part (b) incorrectly. It seems that they could derive $T(n) = T(n) + c$ but they could not obtain the final result that the algorithm might run forever.

Question 12  Randomized Algorithms: Part 2

- Many students wrote an algorithm that checks a constant number of random indices (usually just one) before running a deterministic $\Theta(n)$ algorithm. Checking a constant number of random indices means the probability of failure (and running the deterministic $\Theta(n)$ part) is also constant, resulting in an expected run-time of $\Theta(n)$.
- Several students wrote a deterministic algorithm, arguing that the "probability" of a given index containing the dominant element is at least $1/2$. This is incorrect, since the input instance is not randomly generated, so the probability of any given index containing the dominant element is either 0 or 1, depending on the input instance. The worst-case input instance can be arranged such that these algorithms always take $\Omega(n)$ time (so expected run-time is also in $\Omega(n)$).
- A few students claimed to use a modified random function that returns distinct values with each call, without providing any details whatsoever on how this function can be designed. Correctly constructing such a function would likely involve $\Theta(n)$ initialization time, which would already violate the $\Theta(1)$ expected run-time requirement.
- A few students attempted to delete elements from the array without apparently realizing that each delete takes $\Theta(n)$ time in the worst-case, resulting in the total expected run-time of the algorithm to be in $\Omega(n)$. 

3