Data Structures
Fall 2019
First Midterm Exam — Solutions

1. Question: [10 points] What is the definition of big-oh? State the best big-oh bound that you can for this function:

$$f(n) = 3n^2 + n\ lg\ n - 6n$$

and argue for the correctness of your answer.

Answer: The (or, at least, a) formal definition of big-O is:

$$\exists c, \exists n_0 : \forall n \geq n_0 : f(n) = cg(n)$$

$f(n) \in O(n^2)$. If we define $g(n) = n^2$, then the above holds if there is some c where $f(n) = cg(n)$ for sufficiently large n. So:

$$\begin{aligned} 3n^2 + n\ lg\ n - 6n &\leq 3n^2 + n^2 - 6n \\ &\leq 3n^2 + n^2 + 6n \\ &\leq 3n^2 + n^2 + 6n^2 \\ &\leq 10n^2 \end{aligned}$$

So, if $c = 10$, then $f(n) \leq cg(n)$.

Discussion: The most typical mistake was to give an informal, non-mathematical definition of big-O; that is, providing just the intuition (an "upper bound"). Some got too loose a bound (e.g., $O(n^3)$), which is technically correct, but haphazardly so.

2. QUESTION: [10 points] What is the running time for binary search in a sorted array? Explain briefly.

   ANSWER: $O(lg\ n)$

   The problem starts with the full list of values as candidates. Each step eliminates half of the remaining list of candidates. Therefore, it is only possible to divide n in half $lg\ n$ times before getting down to a single element.

   DISCUSSION: This problem went well overall.

3. QUESTION: [10 points] What is a priority queue?

   ANSWER: A *prority queue* is an *abstract data type* that allows for two essential operations:

   (a) `insert(key, priorty)`: Add the given `key` to the queue, assigning it the given `priority` value.

   (b) `remove()`: Return the `key` with the *minimum priority value* (for a *min-queue*) or the *maximum priority value* (for a *max-queue*) among all of the keys in the queue.

   One can also include `is_empty()`, `size()`, `peek()` (for examining the min/max priority key) without removing it.

   DISCUSSION: The most common error was to give some sort of implementation-specific discussion (e.g., several students said that the highest priority element is at the root). A number failed to mention the operations that must be supported, mentioning that "the data is ordered by priority," but not that only the highest priority element can be removed.

4. QUESTION: [15 points] What are the properties obeyed by a min-heap?

   ANSWER: A *min-heap* is:

   (a) A binary tree of values, where
   (b) Each node's value is less than its children's values, and
   (c) All levels of the tree are full, expect perhaps the bottom-most such that
   (d) On that bottom level, leaves are flush to the left.

   DISCUSSION: The first two properties were often clearly expressed. The latter two properities were sometimes not fully stated. Attempts to explain how operations on the min-heap were performed (especially sifting) were not a substitute for stating the properties themselves.

5. QUESTION: [15 points] Create a grid with 4 rows and 4 columns. The rows should be labelled: sorted array, unsorted array, BST of height h; min-heap. The columns should be labelled lookup (of a particular value), insertion (of a new value), deletion of the min element, deletion of the max element. Give the worst-case running time for each operation in the given data structure.

   ANSWER:

	lookup	**insertion**	**delete min**	**delete max**
sorted array	$lg\ n$	n	n	1
unsorted array	n	1	n or 1	n or 1
binary search tree (h)	h	h	h or 1	h or 1
min-heap	n	$lg\ n$	$lg\ n$	n

   DISCUSSION: First, note that some times for **delete min** and **delete max** have two options. These depend on whether one includes the lookup time for finding the min/max, or whether it is solely the time for removal having already found it. Either was acceptable so long as the answers were consistent.

   A number of people expressed, for example, that an $O(n)$ operation followed by an $O(lg\ n)$ operation required $O(n\ lg\ n)$ time. It does not. If the times follow one another, they are additive, not multiplicative, and the dominant term is the only one that remains. So, in the above example, $O(n + lg\ n) = O(n)$.

6. QUESTION: [10 points] Explain how you might delete the minimum element from a BST.

   ANSWER: From the root, follow the path of left-children until a node with a *null* left-child is reached; this node is the minimum one. Then, set this node's parent to point to this node's right-child. If the right-child is *null*, then the minimum node was a leaf, and has simply been removed; if the right-child is non-*null*, the the parent now takes the subtree rooted as the right child as its left-node, replacing the minimum node itself.

   DISCUSSION: Nearly eveyone described correctely how to find the minimum node. Sometimes, then, no statement was made about how to remove it; some made vague statements about "setting the node to *null*," which is insufficient for describing how to remove the node from the structure. Some remembered to change the parent's left-child pointer, but asserted only that it had to be made *null*, ignoring that the minimum node might have a right child.

7. [30 points] Write code for the following three operations:

   (a) QUESTION: Deletion in a linked list. Suppose you have a pointer to a cell in a linked list. Show the code that you could use to remove that cell from the linked list.

   ANSWER: Assuming a doubly-linked list and use of sentinels for the head and tail:

```
E remove (Node n) {
  n.prev.next = n.next;
  n.next.prev = n.prev;
  return n.val;
}
```

   DISCUSSION: Some failed to absorb that the question assumed a direct pointer to the node to be removed, and then confused this question with the next one, doing a search for the node of rank r. Mostly, though, this question went well.

(b) QUESTION: Deletion of the rank `r` element from a list stored in position `0` through `count-1` of an array. You can assume that the element is in the second half of the array.

ANSWER: The question presupposes that the array's length is larger than `count`, and any wrap-around is irrelevant. Finally, assume that `count` itself, and even the array, are data members that need not be passed.

```
E remove (int r) {
  E val = array[r];
  for (int i = r; i < count - 1; i += 1) {
    array[i] = array[i + 1];
  }
  array[count - 1] = null;
  return val;
}
```

DISCUSSION: Although no points were removed from it, some ignored the simplifying assumptions provided in the question and wrote more complex code in which the queue's storage wrapped around. Most of the errors here were small, being off by one position in the shifting of elements. Many forgot to decrement `count` at the end.

(c) QUESTION: Lookup in a BST of ints. You should determine whether or not the BST contains a particular value.

ANSWER: Assume that `root` is a data member that provides the root of the BST.

```
boolean lookup (int val) {
  return do_lookup(root, val);
}
boolean do_lookup (Node n, int val) {
  if (n == null) return false;
  if (n.val == val) return true;
  if (val < n.val) return do_lookup(n.left, val);
  if (val > n.val) return do_lookup(n.right, val);
}
```

DISCUSSION: The most common outright error was not to test whether `n == null` *first*, which must occur before other tests (e.g., `n.val == val`) can be performed. Some convoluted the issue by testing initially that the root was non-null, and then determining the end of the search by separately testing each child before recurring. It was also valid to perform this search with a loop.