1. **8 sorting algorithms.**

   4 6 3 2 1 7 5 9

2. **Algorithm Properties.**

   - $\log N$ Binary heaps are perfectly balanced by definition.

   - $N$ An unbalanced BST can have height proportional to $N$.

   - $N \log N$ The Sedgewick partitioning algorithm stops on equal keys. As a result, each partitioning step will create two subproblems of equal size, just like mergesort.

   - $N$ If all the keys are equal, 3-way quicksort will terminate after a single partitioning step.

   - $N$ Traversing a tree using \{ inorder, preorder, postorder, level-order \} takes linear time.

   - $N^2$ If all keys hash to the same bin, the $i$th insertion will take time proportional to $i$.

3. **Sorting a linked list.**

   Mergesort is the algorithm of choice for linked lists (Sedgewick 8.7) since the merging can be done in-place. Quicksort is also a good choice since it’s now easy to achieve stability.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Extra memory</th>
<th>Running time</th>
<th>Stability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mergesort</td>
<td>$O(\log N)$</td>
<td>$O(N \log N)$</td>
<td>Y</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$O(\log N)$</td>
<td>$O(N \log N)$</td>
<td>Y</td>
</tr>
</tbody>
</table>

   Some variants of mergesort (bottom-up mergesort and natural mergesort) avoid recursion and only require $O(1)$ extra space.
4. **Comparable interface.**

Because of the epsilon fudge-factor, it’s possible to have both \( a \text{.compareTo}(b) == 0 \) and \( b \text{.compareTo}(c) == 0 \), but not \( a \text{.compareTo}(c) == 0 \). This breaks the contract.

5. **Java API.**

It’s impossible because it would violate the \( \Omega(N \log N) \) lower bound for sorting. The argument is almost identical to the one presented in class for why not all priority queue operations can take \( O(1) \) time.

Here’s a sorting algorithm that uses the `OrderStatistic` API.

- Add the \( N \) elements to the data structure.
- For each \( k \) from 1 to \( N \), print the \( k \)th largest.

If all operations take \( O(1) \) time, this is an \( O(N) \) sorting algorithm. Since `OrderStatistic` only accesses the `Comparable` items through the `compareTo()` method, this contradicts the sorting lower bound.

6. **Binary heaps.**

(a) ![Binary heap diagram](image)

(b) Inserting \( M \) causes array entries 13, 6, and 3 to change.

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>X</td>
<td>W</td>
<td>M</td>
<td>V</td>
<td>U</td>
<td>J</td>
<td>H</td>
<td>S</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>
```

(c) Inserting \( M \) causes array entries 12, 1, 2, 4, and 8 to change.

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>W</td>
<td>V</td>
<td>J</td>
<td>S</td>
<td>U</td>
<td>D</td>
<td>H</td>
<td>C</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
```
7. Red-black trees.

Unfortunately, the figure shown is not a red-black tree! The 3-node containing I and C has only two children (instead of 3). As a result, we accepted either of the following two solutions (and awarded extra credit for correctly observing that the figure is not a red-black tree).

8. Two-sum.

There are two main approaches. (Note that we excluded 0 and \(-2^{63}\) since these are the only two \texttt{long} integers \(x\) such that \(x + -x = 0\).)

- \textit{Hashing}. Insert each integer \(x\) into a hash table (linear probing or separate chaining). When inserting \(x\), check if \(-x\) is already in the hash table. If so, you’ve found two integers that sum to 0.

  The running time is \(O(N)\) on average, under the assumption that the hash function maps the keys uniformly. The running time is quadratic in the worst case, if all the keys hash to the same bin.

- \textit{Sorting}. Sort the integers in ascending order into an array \(a[]\). Maintain a pointer \(i = 0\) to the most negative integer and a pointer \(j = N - 1\) to the most positive integer. If \((a[i] + a[j] == 0)\), you have two integers that sum to 0. Otherwise, if the sum is negative, increment \(i\); if the sum is positive, decrement \(j\). Stop when \(i = j\).

  The bottleneck operation is sorting. This takes \(O(N)\) time in the average-case and worst-case using a radix sorting algorithm.