Sorting revisited

How did we use a binary search tree to sort an array of elements?

Tree Sort Algorithm

Given: An array of elements to sort
1. Build a binary search tree out of the elements
2. Traverse the tree in-order and copy the elements back into the array.

Runtime:

Worst-case:
When does the worst-case runtime occur? It occurs when the binary search tree is unbalanced, for example when the data is already sorted.

- Build the binary search tree: $O(n^2)$
- Traverse the binary tree: $O(n)$
- Total: $O(n^2) + O(n) = O(n^2)$

Best-case:
- Build the binary search tree: $O(n \log n)$
- Traverse the binary tree: $O(n)$
- Total: $O(n \log n) + O(n) = O(n \log n)$

Of what sort does tree sort remind you?

<table>
<thead>
<tr>
<th>Quicksort</th>
<th>Tree sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first element is the pivot, which is used to partition the elements into those less than the pivot and those greater than the pivot.</td>
<td>The first element becomes the root of the tree. All elements are compared with the root and are separated into those less than the root and those greater than the root.</td>
</tr>
<tr>
<td>Recursively apply quicksort to the two partitions:</td>
<td>Building the left and right subtree corresponds to the recursive calls in quicksort. The elements and size of the left and right subtrees are the same as the two partitions:</td>
</tr>
<tr>
<td>The first element less than the pivot becomes the pivot for partitioning in the recursive call. Similarly for the elements greater than the pivot.</td>
<td>The first element less than the root becomes the root of the left subtree and all elements in that subtree are separated by the left subtree root. Similarly with the right subtree.</td>
</tr>
</tbody>
</table>
What distinguishes Quicksort from Tree sort?

<table>
<thead>
<tr>
<th>Quicksort</th>
<th>Tree sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-place (uses only the original array)</td>
<td>Extra memory space for the tree</td>
</tr>
<tr>
<td>It can attempt to balance by picking a</td>
<td>Can maintain balance to avoid worst-</td>
</tr>
<tr>
<td>random pivot, in which case $O(n \log n)$</td>
<td>case behavior, in which case $O(n \log n)$,</td>
</tr>
<tr>
<td><strong>expected</strong> worst-case runtime</td>
<td><strong>guaranteed</strong> worst-case runtime</td>
</tr>
</tbody>
</table>

Can you use TreeSet to implement Tree Sort?
Only if you have don't have duplicates.

How can you use TreeMap to implement Tree Sort.
Use map to a count of the duplicated elements.
Then duplicate the element that number of times when you copy back to the array.

Recall:

Which sorting algorithm is always slow $O(n^2)$? Selection sort

Which sorting algorithm is sometimes $O(n)$? Insertion sort

What sort runs in $O(\log n)$? None! Need to look at each element at a minimum!

What other sorts are $O(n \log n)$ expected time? Quicksort, Merge sort.

Which sorting algorithm needs extra memory space? Merge sort, Tree sort

Can we do better than doing $O(n \log n)$ comparisons to sort?

No!

**Theorem:** There is no sorting algorithm that uses less than $\Omega(n \log n)$ comparisons (lower bound) on some input. That is, for any sorting algorithm, there is always some permutation of the input that forces the sorting algorithm to make at least $n \log n$ comparisons.
How did the student sort the deck of n playing cards by color, so that all the red cards appear before the black cards?

1. Go through the deck and put each card into one of two piles, red and black.
2. Move the red cards and then the black cards into a single pile.

*What is the run time?* Part 1: $O(n)$ Part 2: $O(n)$ Overall: $O(n)$

*Can we sort a deck of n cards by suit the same way?* Make 4 piles, one for each suit. Still $O(n)$ runtime.

*Can we sort a deck of n cards by rank in the same way?* Make 13 piles. What is the runtime? $O(n)$ Could there be an ordering of the cards that would take longer? No, it's always $O(n)$.

Can we sort n integers all in the range 1 to 100 in $O(n)$ time? Yes. Make 100 piles.

*How can that be?* Isn't sorting supposed to be at least $\Omega(n \log n)$? It is, but only if you sort by comparing one value with another value. All the other sorts we seen so far are comparison-based sorts. When sorting the playing cards we just put the values into the correct pile (buckets) without comparing one value against another.

---

**Bucket Sort (aka Pigeonhole sort)**

Let $k$ be the range of values of the elements being sorted, where each value is discrete.

**Algorithm**

1. Create an array of $k$ buckets, one for each possible discrete value.
2. Copy each element to its unique bucket by computing the bucket's index. Keep all the values within a bucket in a linked list.
3. For each bucket in order from smallest to largest value, copy each element within the bucket back to the input array.

Bucket sort is similar to hashing where hash function is, $h(X_i) = X_i$. That is, insert into bucket equal to the element value.
What is the runtime to sort n elements using Bucket Sort?

1. $O(k)$
2. $O(n)$
3. $O(k)$ to iterate over the k buckets + $O(n)$ to copy each card from the linked lists to the array

Overall: $O(k + n)$.
If $k = O(n)$, Bucket sort is linear!

But...

**Restriction**: Can only be use when the values are discrete and you know the range of values **before** sorting.

There is no such restriction for comparison-based sorts.
Priority Queues

What data structure has the following operations?

- isEmpty
- add - add at end
- remove - remove at front
- peek - at front

Answer: Queue

What happens when you go to the emergency room? Do they use a queue? No. If you have a serious life-threatening condition, you immediately get in the front of the queue.

The queuing paradigm for the emergency room is each person is given a priority and the person with the highest priority goes first.

Examples of a priority queue:
- Print queue for a printer: Smaller printouts have higher priority than larger printouts.
- Process scheduling on a computer: Process that tracks the mouse and displays it on the computer screen has higher priority than a process that is getting email from an Internet server.

Priority Queue has the following operations:

- isEmpty()
- add (with priority)
- remove (highest priority)
- peek (at highest priority)

What is the worst-case runtime for the following implementations of a priority queue?

<table>
<thead>
<tr>
<th>Implementation</th>
<th>add</th>
<th>peek</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>linked list</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>sorted array</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>array of linked lists</td>
<td>O(1)</td>
<td>O(k)*</td>
</tr>
<tr>
<td>binary search tree</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>balance binary search tree</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>heap</td>
<td>O(log n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

* k is the number of distinct priorities.