ALGORITHMS, FALL 2018, HOMEWORK 3

Due Thursday, September 27 at 11:59pm. Worth 2% of the final grade. Submit each problem on a separate page. Subproblems can be on the same page.

1. Suppose that you have n real numbers, placed in groups of size k. The groups are already in "sorted" order in the sense that every number from one group is smaller than every number from the next. However within each group nothing is sorted. You must return all numbers in sorted order. (a) How many possible outputs (permutations of the input) are there?

   $n(k-1)!$

   (b) Provide a lower bound on the worst case time complexity of fully sorting the input, using your answer from (a). Do this without Θ-notation.

   With each of the k groups, you only have to sort each individual group. That means you need to run a constant time operation on each group to make sure they're sorted, this operation being at least $k \log k$, but still in constant time. There are also $\frac{n}{k}$ groups, meaning that the runtime of this is $k \cdot \frac{n}{k} \cdot k \log k$.

   (c) Convert the answer that you get in (b) to an expression that is more "user-friendly", using Θ-notation.

   In the average case, the runtime of this would be $\Theta(n)$

   (d) Briefly confirm that the bound in (c) can be matched: what would you do to sort this input, and what would the cost be?

   I would use a form of merge sort to sort this input, as the groups are already decided and sorted for me. The cost would be sorting each individual group, a cost of $k \cdot \frac{n}{k} = n$, and joining the entire list together, which is a cost of n. Therefore the runtime is $\Theta(n)$.

2. You work at the emergency room at a hospital. Patients come in and you evaluate the severity of their injury by assigning a **real** number from 1 to 100. Whoever has the highest number (and in case of ties, whoever arrived first) sees the doctors first. When a doctor is available, no time should be wasted; you want to identify the person who goes next as quickly as possible (minimize the worst case). (a) How will you handle the data, to automate this process?

   The priority here is to minimize the time that a doctor will be kept waiting, so I searched for a data struct with $O(1)$ deletion time. Insertion here is secondary, as we mainly want to make sure that doctors will never be kept waiting if there are patients around. The way I choose to handle this process is with a sorted linked list. Insertion is $O(n)$ because you have to find the exact place to put the patients, but deletion is $O(1)$ (you just remove the tail – or head – of the list). This is especially effective in the case of multiple doctors being free all at once, in which case we can just pop off N elements from the end of the array and distribute them.

   (b) How much time will it take to send the next patient to the doctors, assuming you've had time to set up properly?

   The runtime for sending the next patient to the doctor is $O(1)$.

   (c) How long will it take to reorganize after a patient is sent through to the doctors?

   The runtime for reorganizing is $O(1)$ because the list stays sorted.

   (d) How long will it take to reorganize after a new patient arrives and you have evaluated their injury?

   The runtime for new patients is $O(n)$ to find their spot. I considered using a max-heap for this case, mainly so that the runtime here would be $O(\log n)$, but decided that it was more important in a parallelized situation (with multiple doctors) for each doctor to receive their patient as quickly as possible.

   (e) Someone in the wait room gets worse, and you assign a new number for them. How do you reorganize and how long does it take?

   This will also take $O(n)$, to search through the list and insert them again. This is another case where a max-heap would be more effective, again with a runtime of $O(\log n)$.

   (f) Someone miraculously gets all better and leaves (after letting you know). How do you reorganize and how long does it take?

   Deletion out of the linked list will be $O(1)$, no problems here!

   (g) Now suppose that all of the assessment numbers are values that get rounded to the first decimal (e.g., 23.6, 63.9, 44.1, etc). How do you handle the data and how does it affect the time complexity of the above situations?

   Limiting the data set to a specific range of numbers allows us to treat it as a counting sort problem. We can initially allocate a 1000 length array of linked lists, each one representing a single possible assesment number. This allows us to find the bucket that a patient belongs in $O(1)$, while also being able to rearrange patients in $O(1)$. There

is a constant cost associated with looking up the next highest patient, with worst-case 1000 lookups (for bucket 1.1), but it is still $O(n)$.