CMPT 225: Data Structures & Programming – Unit 30 – Selection

Dr. Jack Thomas
Simon Fraser University
Spring 2021

Today’s Topics

- Order Statistics
- The Selection Problem
- Prune-and-Search
- Randomized Quick-Select

Finding Particular Elements in a Set

- A common thing we want from **Sets** is to **find a certain element** – the smallest element, perhaps, or the largest, or the one perfectly in the middle.
- In general, the process of finding an element in a Set by its rank is called **Order Statistics**.
- Because Sets are such a high-level and general data structure, **we don't have access to some of the tools or assumptions** (indexes, heap-ordering, tree searches) that other structures do.

The Selection Problem

- The **Selection Problem** is the general form for finding any one element in a Set by its rank (first, last, middle), assuming some total order of all elements.
- The **overkill solution would be to sort the entire Set** – a process we know can take O(n log n) from our work with sorting methods. If we only want one element, ordering every element is wasted effort.

Prune-And-Search

- Also called **Decrease-and-Conquer**, being related to Divide-and-Conquer, this is another general design pattern we can apply to create an **O(n) algorithm** for the selection problem.
- The idea is to **prune away** a fraction of the elements we know aren't the ones we want and then **recurse** on the smaller problem until it gets down to some fixed size we can solve outright.
- Applied to the selection problem, this will lead us to an algorithm for **pruning away everything that isn't the rank of the element we want**, until we're left with the one element that is.

Randomized Quick-Select

- An algorithm for **finding the kth-smallest element in an unordered sequence of elements** where a total order is possible.
- It has an **expected O(n)**, but technically a **worst-case of O(n^2)**.
- This might remind you of **Quick-Sort**, which makes sense, because the algorithm is very similar to Quick-Sort.

```
Algorithm quickSelect(S, k)):
Input: Sequence S of n comparable elements, and an integer k of [1, n]
Output: The kth smallest element of S
If n == 1
        return the (first) element of S
Pick a random (pivot) element x of S and divide S into three sequences:
        L, storing the elements in S less than x
        E, storing the elements in S equal to x
        G, storing the elements in S greater than x
If k <= |L|
        quickSelect(L,K)
Else if k <= |L| + |E| then
        return x
Else
        quickSelect(G, k - |L| - |E|)
```

Java Support for Quick-Select

- There is **no built-in support for Quick-Select in Java**, nor is there a particular function that easily recreates it.
- You could at least find the kth element, albeit inefficiently, by **getting the iterator** (array version) of the Set, **sorting** it, and then using the **index**, but that won't help if you actually need Quick-Select's speed.
- It's such a simple algorithm to code though that you **could easily add it to a class** that extends an existing Set or inherits the Set interface if your particular application has need of it.

Recap – A Selection of Topics

- **Order Statistics** are queries for Sets to retrieve an element of a given rank.
- The general form of these queries is called the **Selection Problem**, which can be solved through sorting in O(n log n), but which we want a more efficient solution to.
- We can apply the **Prune-and-Search** design pattern to produce **Randomized Quick-Select**, an algorithm that finds the kth-ranked element in a sequence Set in an expected O(n).