Lecture 1 (08-29) - Quick Selection, Median of Medians

> **How to succeed in the class:**

1. Read the problem.
2. Think really hard.
3. Write down the solution.

Algorithms, at their core are a series of steps that solve some well-defined problem.

- We also want to have some guarantees, usually based on resources used (time and space).

Proving efficiency

One way to think about proving efficiency is that of the prover and the adversary:

The prover will attempt to assert that for all inputs, this algorithm runs in some $O(f(n))$ time. An adversary will attempt to find an input that disproves this.

Median Finding

Definition: The median of an array A is the $\frac{n}{2}$ smallest element. We want to return the index of it. This can, if fast enough, be a good way to find a good pivot for quicksort.

Let's begin with some naïve algorithms:

1. Guess a random element in A, and compare all other $n-1$ elements with it. If there are $\frac{n}{2}-1$ elements smaller than it, return it. Else, choose a different element and repeat. In the worst case, we will have to do this $(n-1)^2$ or $O(n^2)$ time.
2. Comparison sort our array A in $O(n \log n)$ time. Then, pick the $\frac{n}{2}$ element in $O(1)$ time. This provides an overall time of $O(n \log n)$ time.

Can we go faster, though? Say $O(n)$ time?

Perhaps it is better to treat the median case as a specific case of a more generalized problem: what is the k^{th} smallest element in the array, where $k = \frac{n}{2}$ in our case.

We can take some inspiration from Quicksort:

Quick Selection

1. Choose a random pivot p and make two lists of the rest of the elements: one where the elements are $\leq$ the pivot, `lt` and the other where the elements are $>$ the pivot, `gt`. This will take $n-1$ comparisons.

2. Let L be the length of `lt` the pivot:
    - If $L = k - 1$, return p (our pivot is the k^{th} element).
    - If $L > k - 1$, then our k^{th} element is in the `lt` array - return `QuickSelect(less, k)`.
    - Otherwise, our k^{th} element is in the `gt` array - return `QuickSelect(more, k - L - 1)`. Notice we subtract the size of L and p from k as we have ruled out the $L + 1$ smallest elements.

Showing Efficiency

Evidently, the worst-case performance of this algorithm would be pretty bad as we could be really unlucky with our pivot and choose the least/greatest element each time, necessitating $\frac{n}{2}(n-1) = O(n^2)$ comparisons.

However, what about the *expected* case?

To begin, we know that for each each call we will be comparing on the order of $(n-1)$ elements (this will actually get smaller with each iteration as we rule out more elements, but we will use it as an upper bound).

Next, how many calls will we be doing in total? First, it may be useful to find how much of the array we eliminate on average with each iteration. Consider the following:

- We will either find our pivot immediately **OR** discard the smaller of the two arrays (when L is greater, we discard `gt` for instance).
- In this manner, we reduce the problem to a size of $\frac{n}{2}$ at most and $n-1$ at least. The average case will be:

$$\frac{1}{\frac{n}{2}}\sum_{i=\frac{n}{2}}^{n-1} i \Rightarrow \frac{2}{n}\Big(\Big(\frac{n}{2}+n-1\Big)+\Big(\frac{n}{2}+1+n-2\Big)\dots\Big) \Rightarrow \frac{2}{n}(\frac{n}{4}\times\frac{3n}{2}) \Rightarrow \frac{3n}{4}$$

  Then, we can write the recurrence $T(n) = n - 1 + T\left(\frac{3n}{4}\right),\ T(1) = 1$.
  However, this isn't quite right. While we know the average size of the new problem, $E[T(i)]$ does not necessarily equal $T(E[i])$. Therefore, our true recurrence is:

$$T(n) = n - 1 + \frac{2}{n}\sum_{i=\frac{n}{2}}^{n-1} T(i),\ T(1) = 1$$

  We can now show that the expected runtime is linear via induction.
  Let us assume as our inductive hypothesis that $T(i-1) \leq 4(i-1)$ for all $i \leq$ some $n-1$ (the case of $i = 1$ is trivially true by our base case as our inductive base).

Now, we can show that $T(n) \leq 4n$.

$$T(n) = n - 1 + \frac{2}{n}\sum_{i=\frac{n}{2}}^{n-1} T(i)$$

As $\left[\frac{n}{2}+1, n-1\right] \leq n-1$, we may substitute $T(i)$ in the summation with simply $4i$ by our hypothesis as follows:

$$T(n) = n - 1 + \frac{2}{n} \times 4 \sum_{i=\frac{n}{2}}^{n-1} i$$

From the previous result, we've computed the sum to be $(\frac{n-1}{4} \times \frac{3n}{2})$. Substituting this in, we get:

$$T(n) = n - 1 + 3n = 4n - 1 \leq 4n$$

Which is precisely what we wanted to show.

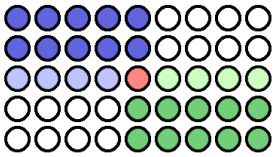
Median of Medians and Deterministic Selection

While quick selection is great, its worst case can be rather slow, and we aren't guaranteed a good outcome - just the average case will be linear. Can we do so in a guaranteed fashion?

The key is to select a pivot that is guaranteed to do well. To do so quickly, we can use a median of median algorithm:

1. Split your array into groups of 5 (doesn't have to be 5, any odd number ≥ 5 is generally fine).
2. For each of the $\frac{n}{5}$ groups of 5, find the median in constant time.
3. Then, recursively find the median of the medians until we get a single median to rule them all.

How good of a pivot this is depends on our partition size. For groups of 5, we can refer to the figure below:



Notice that we know the median of medians is guaranteed to be $\leq \frac{n}{5} \times \frac{1}{2}$ medians $+\frac{2n}{5} \times \frac{1}{2}$ elements smaller than those medians, or $\frac{3n}{10}$ total elements (blue). By symmetry, it is also $\geq \frac{3n}{10}$ total elements (green).

We can then use this pivot to perform a selection:

1. Split your array into groups of 5 (doesn't have to be 5, any odd number ≥ 5 is generally fine).
2. For each of the $\frac{n}{5}$ groups of 5, find the median in constant time.
3. Then, recursively find the median of the medians until we get a single median to rule them all.
4. Let the result of (3) be the pivot p and make two lists of the rest of the elements: one where the elements are $\leq$ the pivot, `lt` and the other where the elements are $>$ the pivot, `gt`. This will take $n - 1$ comparisons.
5. Let L be the length of `lt` the pivot:
   - If $L = k - 1$, return p (our pivot is the k^{th} element).
   - If $L > k - 1$, then our k^{th} element is in the `lt` array - return `QuickSelect(less, k)`.
   - Otherwise, our k^{th} element is in the `gt` array - return `QuickSelect(more, k - L - 1)`. Notice we subtract the size of L and p from k as we have ruled out the $L + 1$ smallest elements.

Showing Efficiency

Once again, we'd like to show that even with the additional step of finding a better pivot, we can still perform selection in $O(n)$ time.

Let's split the steps into how long they take:

1. Splitting an array is relatively free.
2. For each of the $\frac{n}{5}$ partitions, we find the median in constant time: $\frac{n}{5} \times O(1) = O(n)$.
3. If we call the entire process $T(n)$, recursively doing finding the median takes $T\left(\frac{n}{5}\right)$ time.
4. As discussed before, this takes $O(n)$ comparisons.
5. Here, by our median of medians pivot, we are guaranteed that no matter what, we will be throwing away a piece of $\frac{3n}{10}$ or more. Therefore, this would take at most $T\left(\frac{7n}{10}\right)$ time.

To solve the recurrence of: $T(n) = O(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right),\ T(1) = 1$. For a linear recurrence such as this, they oftentimes boil down to a geometric sum. For this particular problem, we get:

$$\sum_{i=0}^{n} \left(\frac{9}{10}\right)^{i} \times O(n)$$

As this is a geometric series, it evaluates to a constant in its closed form and therefore our runtime is still linear (even as $n \to \infty$ the sum approaches 10).

#algo #selection #recurrences #median #averagecase