Bounds on Selection

Eklavya Sharma

Abstract

This document analyzes lower and upper bounds on the worst-case number of comparisons required for selection, i.e. finding the k^{th}-smallest element in an array of n elements.

Contents

1 General lower bound

There must be at least $n - 1$ comparisons [proof needed].

2 Specific algorithms

2.1 Sort and select

First sort the array, then return the k^{th} element.

Sorting has a lower bound of $\lceil \lg(n!) \rceil$ comparisons. It can be upper bounded by considering specific sorting algorithms. All efficient algorithms use $\Theta(n \lg n)$ comparisons.

2.2 Build heap and pop

Build a min-heap of all elements and pop k times.

A binary min-heap of size n can be built using at most $2(n-1)$ comparisons and an item can be popped from it using at most $2 \lfloor \lg(n-1) \rfloor$ comparisons [1].

Therefore, number of comparisons to select the k^{th}-smallest element is upper bounded by

$$2 \left(n - 1 + \sum_{i=1}^{k} \lfloor \lg(n-i) \rfloor \right) \leq 2(n - 1 + k \lfloor \lg(n-1) \rfloor)$$

2.3 Maintain heap and pop

Build a max-heap of first k elements. For each of the rest of the elements, compare it with the heap top. Replace top by new element and repair heap. The heap top at the end of this process is the k^{th}-smallest element.

A binary max-heap of size k can be built using at most $2(k-1)$ comparisons and top can be replaced using at most $2 \lfloor \lg k \rfloor$ comparisons [1].

Therefore, number of comparisons to select the k^{th}-smallest element is upper bounded by

$$2((n-k) \lfloor \lg k \rfloor + (k-1))$$

Although this is not as efficient as the previous algorithm, it has the advantage of being an online streaming algorithm.

2.4 Randomized quick-select

This algorithms partitions the array using a random pivot and then recursively selects from one of the partitions.

Let $f(n)$ be the expected number of comparisons required to select from n elements. $f(0) = f(1) = 0$.

Partitioning the array can be done using $n-1$ comparisons.

$$\begin{aligned} f(n) &\leq (n-1) + \frac{1}{n}\sum_{i=1}^{n} \max(f(i-1), f(n-i)) \\ &\leq (n-1) + \frac{2}{n}\sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} T(i) \end{aligned}$$

We will prove using mathematical induction that $f(n) \leq 4n$. Since $f(0) = 0$, $d \geq 0$.

As the induction hypothesis, assume $f(i)$ for all $0 \leq i \leq n-1$.

$$\begin{aligned} f(n) &\leq (n-1) + \frac{2}{n}\sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} T(i) \\ &\leq (n-1) + \frac{2}{n}\sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} 4i \\ &= (n-1) + \frac{4}{n}\left(n(n-1) - \left\lfloor \frac{n}{2} \right\rfloor \left(\left\lfloor \frac{n}{2} \right\rfloor - 1 \right)\right) \\ &\leq (n-1) + \frac{4}{n}\left(n(n-1) - \frac{n}{2}\left(\frac{n}{2} - 1 \right)\right) \\ &= 4n - 3 \leq 4n \end{aligned}$$

Therefore, randomized quick-select performs at most $4n$ expected comparisons in the worst case.

2.5 Median-of-medians quick-select

Median-of-medians quick-select (MoMQS) is a deterministic variant of quick-select which uses quick-select recursively to decide a good pivot for partitioning. This guarantees worst-case linear time complexity.

Let $f(n)$ be the worst-case number of comparisons needed to run MoMQS on n elements. $f(0) = f(1) = 0$.

Steps in MoMQS:

1. Split array into sub-arrays of length 5 or less such that at most 1 sub-array has length less than 5.
2. Find median of each of those subarrays of length 5. It takes at most 6 comparisons [2] to find each median.
3. Find the median of these $\lceil \frac{n}{5} \rceil$ medians. This takes $f(\lceil \frac{n}{5} \rceil)$ comparisons.
4. Partition the array about the median of medians. This takes $n-1$ comparisons.

5. Recurse into the appropriate partition. It can be proven that the larger part of the array has size at most around $\frac{7n}{10}$. Therefore at most $\frac{7n}{10}$ comparisons are required.

Therefore, a rough recurrence relation for f is:

$$f(n) \approx \frac{6n}{5} + f\left(\frac{n}{5}\right) + (n-1) + f\left(\frac{7n}{10}\right)$$

It can be verified that $f(n) \leq 22n$.

References

[1] Eklavya Sharma. Notes: Heaps. URL: https://sharmaeklavya2.github.io/notes/algorithms/heaps.pdf.

[2] Kenneth Oksanen <cessu@iki.fi>. Selecting the i^{th} largest of n elements. URL: http://www.cs.hut.fi/~cessu/selection/.