Selection

Given an unordered tuple $\underline{a} = (a_1, \ldots, a_k)$ and an integer m in $[1, k]$, we wish to find the m-th element of $\underline{a}$ in order. In other words, if $(a_{\sigma(1)}, \ldots, a_{\sigma(k)})$ is the sorted tuple we want to find $a_{\sigma(m)}$. This is called the *selection problem.*

One way to solve this problem is to apply one of the standard sorting methods to find σ and then use that to find $a_{\sigma(m)}$. As seen earlier, the optimal sorting methods perform $O(k \log(k))$ comparisons, so we will need the same number of comparisons to solve the selection problem this way.

The question is whether we can do better. For example, is there a method that makes $O(k)$ comparisons?

When $m = 1$ (respectively $m = k$) we are looking for the minimum (respectively maximum) element of the tuple. The following algorithm clearly makes $k - 1$ comparisons to solve this particular case.

```
define minimum(a)
    Set ans to be a_1.
    for i in the range [2, k]
        if ans is not < a_i
            Set ans to be a_i.
    return ans
```

A similar algorithm works for `maximum`($\underline{a}$).

There does not seem to be anything similar when m is $k/2$, which is the case when we are looking for the *median* of the tuple. Thus, the selection problem has `median` as a special case.

Randomized algorithm

The following randomized algorithm could be used to solve the selection problem. Note that we *assume* that $m \leq k$ in this algorithm.

```
define ransel(a, m)
    Set k as the length of a.
    if k = 1
        return a_1
    Set p as rand(k).
    Set λ and ρ as empty tuples.
    for i in the range [1, k].
        if a_i ≤ a_p
            append a_i to λ
        else
            append a_i to ρ
    Set n to be the length of λ.
    if m ≤ n
        return ransel(λ, m)
```

else
  return `ransel`$(\rho, m-n)$

Basically, we are using a_p to partition the tuple into λ and ρ and then looking for the appropriate element in each of these sub-tuples.

We have *already* used k comparisons to do the partitioning, so how are we expecting this to be like $O(k)$?

The basic idea is that in each "round" we *expect* the tuples λ and ρ to roughly half the length of the original tuple. Thus, we expect the running time to be roughly

$$k + k/2 + k/4 + \cdots = 2k$$

which is $O(k)$.

We can prove this using the same techniques as were used to prove the expected running time of the randomized quick sort.

The question is whether we can do better? Can we remove the use of `rand` in selection and *still* achieve $O(k)$ time?

The key point to note is that it is enough if the *longest* among λ and ρ is of length at most ck for a constant $c < 1$ independent of k. In that case, we get the number of comparisons bounded above by

$$k + ck + c^2k + \cdots = k/(1-c)$$

which is $O(k)$. So the question is whether we can choose a_p which is "always good enough" in this sense.

Median of medians

We will assume that the elements of the tuple $\underline{a} = (a_1, \ldots, a_k)$ are all *distinct*. If not, we can make a new tuple consisting of pairs (a_i, i) and use dictionary order:

$$(a_i, i) < (a_j, j) \iff ((a_i < a_j) \text{ or } ((a_i = a_j) \text{ and } (i < j)))$$

This assumption ensures that there is a *unique* solution to the selection problem which is essential for the following algorithm to work.

The idea is as follows. Use an algorithm `median1` to find medians of small tuples of size at most 9 (let's say). This can be used to find the medians for the tuples:

$$(a_{5i+1}, a_{5i+2}, \ldots, a_{5i+5}) \text{ for } i \text{ in the range } [0, \lfloor k/5 \rfloor - 2]$$
$$(a_{5t+1}, a_{5i+2}, \ldots, a_k) \text{ for } t = \lfloor k/5 \rfloor - 1$$

This last tuple will have between 5 and 9 elements. Let μ be the tuple of these medians. Then μ has at most $\lfloor k/5 \rfloor$ elements. We choose a_p as the median of the tuple μ.

In this paragraph and later analysis, we will ignore the fact that $k/10$ is not always an integer. One can show that this does not affect things much. At least $k/10$ elements of μ are less than a_p and at least $k/10$ elements of μ are greater than a_p. For each element of μ, at least 2 elements are less than it and at least 2 elements are greater than it. Thus, there are at least $3k/10$ elements less than a_p and at least $3k/10$ elements greater than a_p. It follows that if we partition at this choice of a_p, then λ and ρ are of size at most $k - 3k/10 = 7k/10$. Finally note that $7k/10 + k/5 = 9k/10$ which is ck for $c < 1$!

We now consider the algorithm `mselect` as follows. As mentioned above, this algorithm *assumes* that the elements of $\underline{a}$ are distinct.

define `mselect`$(\underline{a}, m)$
  Set k as the length of $\underline{a}$.
  if $k \leq 9$
    return `mselect1`$(\underline{a}, m)$
  Set b as `medmed`$(\underline{a})$.
  Set λ and ρ as empty tuples.
  for *i in the range* $[1, k]$.
    if $a_i \leq b$
      append a_i to λ
    else
      append a_i to ρ
  Set n to be the length of λ.
  if $m \leq n$
    return `mselect`(λ, m)
  else
    return `mselect`$(\rho, m - n)$

This uses the algorithm `mselect1` to *directly* solve the problem when the tuple has length at most 9. It uses the following "median of medians" algorithm `medmed` when the length is at least 10.

define `medmed`$(\underline{a})$
  Set k as the length of $\underline{a}$.
  Set μ to be the empty tuple.
  for *i in the range* $[0, \lfloor k/5 \rfloor - 2]$
    Append `median1`$(a_{5i+1}, \ldots, a_{5i+5})$ *to μ.*
  Append `median1`$(a_{5\lfloor k/5 \rfloor - 4}, \ldots, a_k)$ *to μ.*
  Set m to be half the length of μ.
  return `mselect`(μ, m)

In turn, this uses the algorithm `median1` that returns the median of lists of length at most 9. This too can be written in terms of `mselect1`. Since `mselect1` can be written in terms of a program that sorts a list of length at most 9, we can assume that this requires at most 30 comparisons. (Note that $9 \log_2(9) \leq 30$.)

Suppose the number of comparisons made by `medmed` on a list of size k is at most $M(k)$ and the number of comparisons make by `mselect` on a list of size k

is at most $S(k)$. Since the length of μ is $\leq k/5$ we obtain the inequalities:

$$S(k) \leq k + M(k) + S(7k/10)$$
$$M(k) \leq 30(k/5) + S(k/5)$$

Again, we are ignoring the divisibility issues which can be seen not to have a major effect.

We claim, by induction on k, that for some large enough constant N, we have $S(k) \leq Nk$. If $N > 1$ this is clearly true for $k = 1$. Suppose we have proved this for all positive integers less than k where $k > 1$. Then, we have

$$S(k) \leq k + M(k) + N(7k/10)$$
$$M(k) \leq 30(k/5) + N(k/5)$$

which combine to give $S(k) \leq k(1 + 6 + 7N/10 + N/5) = k(7 + 9N/10)$. Now, we only need $N \geq 70$ to ensure $S(k) \leq Nk$. Thus, we have proved that `mselect` makes $O(k)$ comparisons.