Finding the Median

Prof. Michel Goemans

---

1 Median Finding

Suppose we have a list of n keys that are completely unsorted. If we want to find the largest or the smallest key, it is very easy to do so with $n-1$ comparisons. If we want to find the mth largest key then you can use heapsort to do it in $cn + cm\log(n)$ comparisons for some c. Indeed, it takes linear time to build the heap and then one can extract the maximum key m times in order to get the desired key (at a cost of $\log(n)$ per extraction). If m is larger than $n/2$ we can do slightly better by reversing the ordering (i.e. building the heap based on minus the keys). In any case, if we are interested in the mth smallest key (the key of *rank* m) and m is close to $n/2$ (or linear in n), we are not saving much time comparing to sorting all keys. In these notes, we show how to find the median (key of rank $(n+1)/2$ if n is odd, or key of rank $n/2$ or $n/2+1$ if n is even) with only a linear number (cn) comparisons in the worst-case.

The first approach that comes to mind when trying to find the rank m key is to take an arbitrary key, say p, and partition the n keys into two piles, one containing the keys smaller than p and the other containing the keys greater than p. By comparing m to the size of the piles, we can either find out that we got extremely lucky and the mth key is p or we can discard one of the piles and recursively search the appropriate key in the other pile. More precisely, if the first pile has at least m elements, we can recursively find the rank m key in the first pile; if not, we can replace m appropriately and search in the second pile. The problem of this approach is that, in the worst-case, one of the piles will (repeatedly) be empty, which would result in a number of comparisons equal to $(n-1)+(n-2)+\cdots$, which grows quadratically with n.

We will now describe a technique for finding a key p for which the two piles will be reasonably balanced, and this will result in an algorithm for finding the median or any rank m key that takes a linear number of comparisons. This will not be a practical algorithm, as there are algorithms that run much faster most of the time (although in the worst-case will take more than a linear number of comparisons). We show you this because it is a really neat algorithm, and because you will learn some useful techniques and facts about recursion from it.

The big question is how to find a "good" candidate for the key p. Ideally, we would like p to be as close as possible to the median. The following is a simple plan for doing just that.

Step 1. We arbitrarily split the keys into groups of size 5, assuming n is divisible by 5; we ignore small differences otherwise. (Why did we choose 5? Actually, any small odd number larger than 3 works, but it turns out that 5 gives the fewest number of comparisons.)

Here is a an example of a set of 35 keys; the seven groups of 5 correspond to the columns.

23	32	6	22	76	15	40
91	28	39	12	97	29	33
75	23	53	71	80	55	68
68	38	64	77	24	7	47
82	10	1	43	37	25	16

We then sort each group of 5 keys, noting that each one can be sorted using a maximum of 7 comparisons (this is a slightly tricky exercise). After performing $7n/5$ comparisons, this leaves us with $n/5$ groups of 5 sorted keys. Here is the example in which the columns (groups of 5) are sorted from top to bottom).

23	10	1	12	24	7	16
68	23	6	22	37	15	33
75	28	39	43	76	25	40
82	32	53	71	80	29	47
91	38	64	77	97	55	68

Step 2. Now, we take each of the keys of rank 3 (one per group), the median rank in its group of 5, and find the median of this group of $n/5$ median keys. This "median of medians" is our good candidate p. If $f(m)$ is the number of comparisons it takes to find the median of m keys, this step takes $f(m/5)$ comparisons.

In our example, we find the median of:

75	28	39	43	76	25	40

which happens to be $p = 40$.

Step 3. Now we partition the keys in two piles, those smaller than p are placed in $L_<$ and those larger than p are placed in $L_>$. If we find other keys equal to p, we can place them either in $L_<$ or in $L_>$.

In order to make this partition, we first compare the rank 3 key of each group of five to p. This takes $n/5$ comparisons. If such a rank 3 key is larger than p then not only do we know that it can be placed in $L_>$ but we also know that the rank 4 and rank 5 keys of that group can also be placed in $L_>$. Similarly, if the rank 3 key happens to be smaller then we can automatically place the rank 1 and 2 keys of that group in $L_<$. For example, we first compare 75 (the rank 3 key of the first group) to 40 and can immediately tell that the keys 82 and 91 are greater than 40 without doing additional comparisons. By comparing the rank 3 elements in the example to p, we can immediately decide in which pile to place the following keys:

	10	1			7	16
	23	6			15	33
75	28	39	43	76	25	40
82			71	80		47
91			77	97		68

This has two implications. First, to construct both piles, we only need to do (at most[1]) 3 comparisons for each group of 5, for a total of $3n/5$ comparisons.

Secondly, since half of the rank 3 keys will be smaller or equal to p, at least $3\frac{1}{2}\frac{n}{5} = \frac{3n}{10}$ keys will be in $L_>$ and similarly for $L_>$. This means that both

$$\frac{3n}{10} \leq |L_<| \leq \frac{7n}{10},$$

and

$$\frac{3n}{10} \leq |L_>| \leq \frac{7n}{10}.$$

By simply counting the number of keys we place in $L_<$ and in $L_>$, we can find out the exact rank of our candidate p.

Step 4. Knowing the rank of p and comparing it to m, we can keep only one of the two piles thereby eliminating at least $3n/10$ keys. If m is smaller than the rank of p we can throw away all keys in $L_>$, while if m is larger than the rank of p, we can discard $L_<$ (and decrease our value of m by the number $|L_<|$ of keys we have just discarded).

In any case, we have at most $7n/10$ keys left, and need to find the rank m' key in them. This takes at most $f(7n/10)$ comparisons in the worst-case.

It is important to realize that our procedure does not involve circular reasoning, even though our procedure uses as a subroutine a procedure for finding the rank m' key. What we are doing is using the technique of recursion. To find the rank m key out of n keys, as intermediate steps we find the median of $n/5$ keys and the the rank m' key out of $7n/10$ keys. During each of these intermediate steps, we again run the procedure for a smaller number of keys. Our algorithm does not run forever because we will terminate this recursion whenever the number of keys is small (which might be when we reach fewer than five keys); in this case we must use a different procedure for finding the rank m key (we could sort them, for example this is more efficient for small n).

1.1 Showing the Procedure is Linear

If $f(n)$ is the number of comparisons to find the rank m key out of n keys (irrespective of what m is), we have the formula

$$f(n) \leq \frac{7n}{5} + f\left(\frac{n}{5}\right) + \frac{3n}{5} + f\left(\frac{7n}{10}\right),$$

where the $7n/5$ is the cost of sorting the groups of 5, the $f(n/5)$ is the number of comparisons required to find the median of medians, p, the $3n/5$ is the cost of comparing the median of medians to all the keys, and $f(7n/10)$ is the number of comparisons used to find the rank m element out of $7n/10$ elements.

How do we know that this formula is linear in the number of comparisons. That is, how do we know that $f(n) \leq cn$ for some constant c? We will show this by using induction. Suppose that we

[1] if the rank 3 key happens to be equal to p, we don't need any additional comparisons since we can place the rank 1 and 2 keys in $L_<$ and the rank 4 and 5 keys in $L_>$.

have shown that the equation $f(m) \leq cm$ holds for all $m < n$ (we'll figure out the value of c later). Then, we have

$$f(n) \leq f\left(\frac{n}{5}\right) + f\left(\frac{7n}{10}\right) + 2n.$$

But we can substitute $f(m) \leq cm$ for both of these values of f on the right hand side of the equation, because for both of these we have $m \leq n$. This gives:

$$f(n) \leq \frac{cn}{5} + \frac{c7n}{10} + 2n.$$

Now, we'd like to choose c so that this inequality is true for all n. To get the best value for c, we can impose that the right-hand side of this equation is equal to cn. This is a linear equation $cn = c\frac{n}{5} + c\frac{7n}{10} + 2n$ where the n's cancel and the solution is $c = 20$. We also need to verify that when n is small enough and we decide to simply sort the keys that the number of comparisons is indeed at most $20n$.

1.2 Improving the procedure

There are lots of clever tricks we can use to improve the constant in this procedure. All use the fact that we have been discarding lots of precious information that we have gathered in previous steps, and that we could instead recycle. We will describe one extremely simple improvement.

In step 2, when finding the median p of the $n/5$ rank 3 keys, we end up knowing for each of these $n/5$ keys whether they are greater or smaller than p. There is no need therefore to compare them again to p in step 3. By combining step 2 and part of step 3, we can therefore decrease the number of comparisons in step 3 from $3n/5$ to $2n/5$. The recurrence we thus get says that

$$f(n) \leq \frac{7n}{5} + f\left(\frac{n}{5}\right) + \frac{2n}{5} + f\left(\frac{7n}{10}\right),$$

and this implies that $f(n) \leq 18n$.

Many further improvements can be made, but we won't discuss them here.