

3. Finding the Median

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 quite easy with $N-1$ comparisons. If we want to find the m^{th} largest key, then you can use tournament or heap sort to do it in $O(N) + O(m \log N)$ comparisons. But suppose you want to find the key at rank $N/2$. This is called the median key. The tournament sort will take $O(N \log N)$ comparisons, which is essentially as much work as sorting. Can we do better? Can we find the median in linear time?

3.1 Probabilistic Method

If we want to find the median in an efficient and practical way, we can use a probabilistic method to give us a very good chance of doing so without much more than $3N/2$ comparisons. (However, if luck is against us it could take a great deal more than $3N/2$ comparisons. See the next section for more about this "worst-case" scenario.)

What is the probabilistic method? We pick a random sample of something like N^α keys, and find the median of this sample. If we find the median by sorting the sample, this will require only $N^\alpha \log_2 N^\alpha$ comparisons, and if $\alpha < 1$, this step takes much less than N comparisons.

Now, a fairly easy probabilistic argument says that the median of the sample is going to be pretty close to the median of the original set. Our next step is to find the actual rank of the median of the sample by comparing this median with all the keys not in the sample, which requires about N comparisons. With any luck, the sample median will have rank only around N^β away from the true median at $N/2$. (Here, it turns out that using probability theory, we can show that β is probably around $1-\alpha/2$. However, the exact value of β is really unnecessary for the algorithm to work, as long as $\beta < 1$.)

Let's assume that we've found a key which is larger than the true median. (If the key we've found is smaller than the true median, just do everything we say from now while standing on your head.)

Suppose the key we've found has rank $N/2 + d$, where d is around N^β . How do we find the true median? Well, if our sample median is larger than the true median, we only need to look at keys smaller than the sample median. We've already figured out which ones these were, so we have a list of around $N/2 + d$ keys of which we want the one with rank $N/2$. We can find this by using tournament sort (or heapsort) to pull the largest key off this list d times. If we use tournament sort, this process takes around $N/2 + d + d \log_2$

$(N/2 + d)$ comparisons, and so the whole process has taken something like $N + N/2 + O(N^\alpha \log_2 N + d \log_2 N)$ comparisons, which is only a little more than $3N/2$.

3.2 Worst-Case Method

Now we come to a harder question: Suppose we want to find the median ranked key "in the worst case." This means we assume that we get no breaks at all. If we choose a key at random, it will turn out to have rank 1 or N and be almost useless to us. Any sample we choose will be atypical. We will no longer be able to assume, as in the previous section, that d is much smaller than N , and our nice error bound will disappear. Can we still find the median in a linear number of comparisons? This will not be a practical algorithm, as the probability that the probabilistic method fails to work in a

reasonable time is astronomically low. We show you this because it is a real neat algorithm, and because you will learn some useful techniques and facts about recursion from it.

The answer is yes, and in fact this can be done with at most cN comparisons, with c under 10. We will find a crude way to do this, which can be improved by various clever tricks.

Our plan is this: find a good candidate for the median and compare that candidate with every other key. This will establish the rank of the candidate; if it is higher than the rank we seek, then we can eliminate every key larger than the candidate, and if the candidate's rank is lower than the rank we seek, we can eliminate all the keys of lower rank than the candidate, since they cannot be the ones we want. Thus, we can reduce our problem of finding the median of N keys to the simpler problem of finding an off-median key of rank not too far from $N/2$.

3.3 Finding a Good Candidate

So we have our plan, and the big question is how to find a "good" candidate for the median. We are assuming that a randomly selected candidate will never be close to the median, so we have to come up with a method that guarantees that our candidate is somewhat close 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.)

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 (see exercises at the end for more detail). After performing $7N/5$ comparisons, this leaves us with $N/5$ sorted sets of keys.

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

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

Now, let's consider the situation. We know the median of the $N/5$ medians. Let's put all the medians of groups of five that are smaller than this median median to its left, and the medians of all groups of five that are larger than it to its right. We now get an array that looks like this.

38	64	55	68	91	77	97
32	53	29	47	82	71	80

28	39	25	40	75	43	76
23	6	15	33	68	22	37
10	1	7	16	23	12	24

All the keys above and to the right of the median median must be larger than it, because they are larger than the median of their group of five, and this median is in turn larger than the median median. Similarly, all the keys below and to the left of the median median are smaller than it.

Step 3: We now find the exact rank of the median median. We already know $3N/10$ keys that it is smaller than (those above and to its right), and $3N/10$ keys that it is larger than (those below and to its left). Thus, we need only compare it with the remaining $2N/5$ keys to find its exact rank.

Step 4: Now, here is the clever part. If the median median is larger than the true median, then we can throw away all keys larger than the median median, in the upper right of the array. If it is smaller than the true median, then we can throw away all keys smaller than it. Since we have eliminated $3N/10$ keys, we have only $7N/10$ keys left, out of which we need to find either the key at rank $5N/10$ or the one at rank $2N/10$. This takes $f(7N/10)$ more comparisons, where $f(N)$ is the number of comparisons to find the rank m key out of N keys (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 median. What we are doing is using the technique of **recursion**. To find the median of N keys, as intermediate steps we find the median of $N/5$ keys and the the rank m key (for some m) 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 we are finding the median of is small (which might when we reach fewer than five keys); in this case we must use a different procedure for finding the median (we could sort them, for example — this is more efficient for small N).

3.4 Showing the Procedure is Linear in the Number of Comparisons

So if $f(N)$ is the number of comparisons to find the rank m key out of N keys, we have the formula

$$f(N) \leq 7N/5 + f(N/5) + 2N/5 + f(7N/10),$$

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 median, the $2N/5$ is the cost of comparing the median median to elements to its upper left and lower right, and $f(7N/10)$ is the number of comparisons used to find the rank m element out of $7N/10$ elements. Note that in the recursion, we must make sure that our procedure works for a slightly more general problem: finding the rank m element rather than finding the median. Although we haven't analyzed this carefully, our procedure works for this slightly more general problem with no alterations.

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 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

$$f(N) \leq f(N/5) + f(7N/10) + 9N/5$$

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

$$f(N) \leq cN/5 + c7N/10 + 9N/5$$

Now, we'd like to choose c so that the right-hand side of this equation is equal to cN . (The proof would also work if the right-hand side were strictly less than cN , but then we wouldn't obtain the best value of c .) This is a linear equation

$$cN = cN/5 + c7N/10 + 9N/5$$

where the N 's cancel and the solution is $c=18$.

3.5 Improving the Procedure

There are lots of clever tricks you can use to improve the constant in this procedure. We'll only tell you about one in these notes (more are explained in the supplementary notes). The first thing we did in the algorithm was make groups of five keys and sort these groups. Then, when we applied recursion on $7N/10$ keys, we again divide these remaining keys into groups of five and sort the groups. But we're wasting effort here, **because $N/2$ of these keys are already in sorted groups of five**. We really don't need to sort these again, and we can save a lot of effort that way.

The rest of the keys are already in sorted groups of two. What is the most efficient way to put them into sorted groups of five? One way is to first put them into groups of five, with each group containing two sorted groups of two and one more key. It then takes five additional comparisons to sort groups of five (you can check this as an exercise). Thus, when we do the recursion on $7N/10$ keys, we only need to use $N/5$ additional comparisons to put the remaining keys into sorted groups of five.

How do we figure out how many steps this revised algorithm takes. It's a little more complicated, but we can use the same idea. Define $g(N)$ to be the (worst-case) number of comparisons to find the rank m key starting with N keys **which have already been placed into sorted groups of 5**. Now, we have the equations

$$f(N) \leq g(N) + 7N/5$$

and

$$g(N) \leq f(N/5) + g(7N/10) + N/5 + 2N/5$$

where the first equation comes from the fact that you can put N keys into sorted groups of five using $7N/5$ comparisons, and the second comes from analyzing the procedure starting with sorted groups of five, and using the fact you can put the $N/5$ keys that are already in sorted groups of two into sorted groups of five using $N/5$ comparisons.

How do we solve these equations? We can substitute the first equation into the second to get the equation

$$g(N) \leq g(N/5) + 7N/25 + g(7N/10) + N/5 + 2N/5$$

and then solve it the same way we used for the recursion equation in the previous section. This gives $g(N) \leq 44N/5$, so $f(N) \leq g(N) + 7N/5 = 51N/5 = 10.2 N$. Using various other clever tricks, you can

make the constant somewhat lower. Some of these tricks are described in the supplementary notes for this lecture.

Exercises:

- 1. Exhibit how to sort 5 keys using 7 comparisons (this is tricky), and how to start from two sorted pairs and one other key and completely sort them using 5 comparisons (this shouldn't be hard if you know how to do the first part).**
- 2. Derive a similar bound dividing into sets of 7 keys rather than the 5 used here. What bounds do you get?**