## 3. Finding the Median

We now address the following question: *We have $N$ keys, that are completely unsorted, and we wish to find the key at rank $m$ so that there are $m-1$ keys bigger than it, and $N-m$ keys smaller than it.*

When $m$ is $N/2$, this key is called the **median key**, and this is the hardest one to find.

One approach to this problem is to sort the entire set and then pick out the median key and we are done. This would require about $N \log_2 N$ comparisons as we saw in the previous notes.

However, it is a waste of time and computing power to sort the entire list when all we really want is to identify the median key, without regard to the position of the other keys. This leads to the question of whether there is a better way to find the median. Can we, for example, find the median in a linear number of comparisons, that is $cN$ comparisons, where $c$ is a constant independent of $N$ ?

If we want to do this for a small or limited sized list of keys, the answer is of course yes because $\log_2 N$ will be bound by a constant. Thus if we had a thousand keys, the fact that $\log_2 1024$ is about 10 shows us that we can certainly find the median with under $10N$ comparisons. If we went up to a billion keys, then maximum $30N$ comparisons would tell us which key we are looking for. But what about for very large N, can we still find a linear upper bound?

### 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).

We can pick a random sample of something like $N^{\frac{1}{2}}$ keys and find the median of this sample by sorting. This will only require $N^{\frac{1}{2}} \log_2 N^{\frac{1}{2}}$ comparisons, and since $\log_2 N^{\frac{1}{2}} << N^{\frac{1}{2}}$ for large N, we see that this will take much less than N comparisons. We then compare the median of the sample with all the keys not in the sample (which requires about N comparisons), and with any luck, it will not be very far from the true median rank.

Assume that the sample median turns out to have rank $\frac{N}{2}+d$, where $d$ is small compared to $N$. We can then immediately eliminate all the keys in our sample that are larger than our sample median (or smaller than our sample median if d is negative) and then use a tournament sort on the remaining keys to find the true median key. Note that we do not actually have to complete the tournament sort, but rather only find the $d$ highest ranked keys in order to be able to determine which key is the true median. Looking at the previous notes, we see that this tournament sort will take us a maximum of $\left(\frac{N}{2}+d\right)+d\,\log_2\left(\frac{N}{2}+d\right)$ comparisons or so. Since $d$ is small, and thus $d\,\log_2\left(\frac{N}{2}+d\right)$ is small as well, we see that this entire term is really on the order of $N/2$.

The big terms here will be N from comparing the median of the sample to the rest of the keys in order to find its rank, and $N/2$ from running the tournament, for a total of $3N/2$.

### 3.2 Worst-Case Method

Now we come to a harder question: Suppose we want to find the median ranked key "in 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 equally 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?[1]

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.

First notice that if we want to find the median, or any other rank, in at most 10N steps we can do that easily if N is less than half a million by using a tournament sort. We can use the fact that the second biggest loses only to the biggest to find the second biggest, then repeat this step $\frac{N}{2}-1$ times to get the median. If we wanted a rank lower than the median we could do the same thing, just looking at the losers instead of the winners.

---

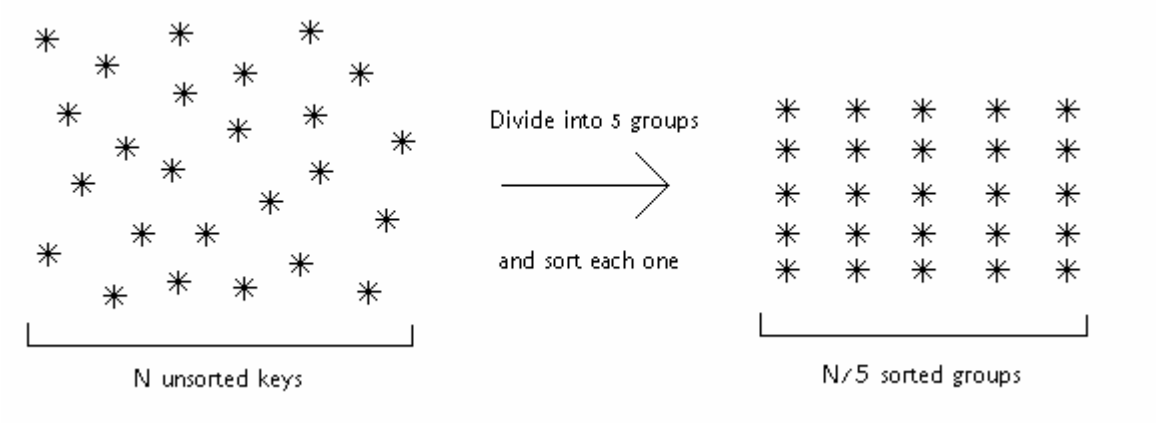
[1] Notice that in such a case, quicksort is quadratic, since if we pick our keys to insert at random, they will always turn out to place only themselves after comparing them to all other keys. Quicksort is a commonly used sorting algorithm, so obviously the above situation is not really a practical question. We choose to address it because it is interesting and instructive in how to find an upper bound on the number of steps required to perform an algorithm

So suppose that N is much greater than half a million. 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 higher than the candidate, and if the candidate is lower than the rank we seek we can eliminate all the keys of even lower rank than it, since they cannot possibly be the ones we want. Thus, we can reduce our problem of finding the median of N keys to a smaller problem of finding an off-median key.

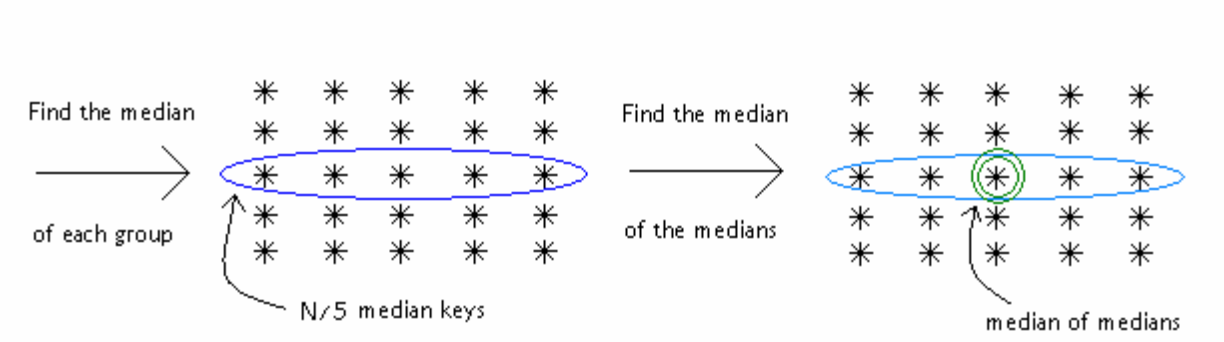
### 3.3 Finding a Good Candidate

So we have our plan, but 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.

First, we pick a nice odd number above 3, say 5. Having picked 5, we arbitrarily split the keys into sets of size 5 (assuming N is divisible by 5; we ignore small differences otherwise.) We then sort each set of 5 keys, noting that each one can be sorted using a maximum of 7 comparisons (see the exercises at the end for more detail). This leaves us with $N/5$ sorted sets of keys.

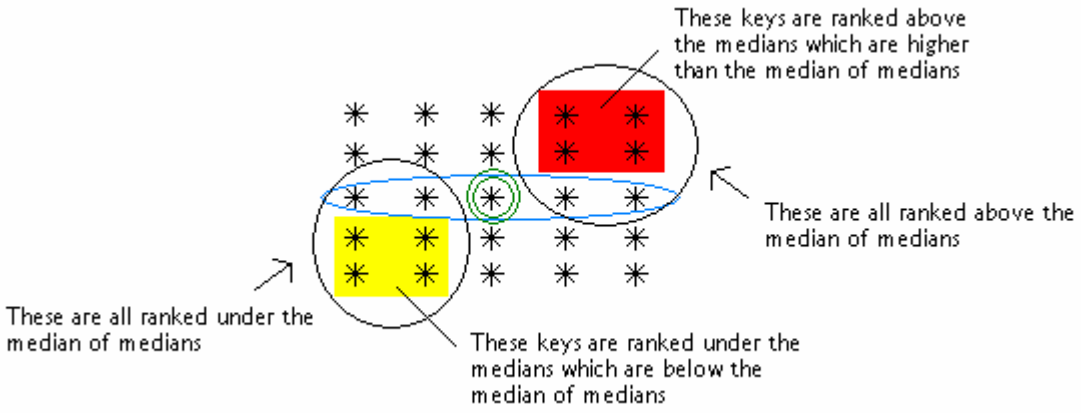



Now we take each of the keys of rank 3, the median rank in its set, and find the median of this group of $N/5$ median keys. This "median of medians" is our good candidate (although in the diagram below we have placed the median of medians in the center group of keys as a simplification, note that it could in fact be in any of the groups as they are not sorted left to right, only top-down).




It is important to recognize the apparent circular reasoning in the above algorithm. In the previous paragraph, we said that we needed to find the median of the medians. However, this entire procedure is designed to help us find the median of a set. So it is quite odd that we have to be able to find the median of a set as part of the procedure for finding the median of a set. This is an example of recursion. The reason that this works is because we are finding the median of N/5 keys, while our algorithm is designed to find the median of N keys. Recall in our description of the procedure that we run it several times, each time eliminating some of the N keys that are above or below our guess. Thus, eventually, N/5 will be so small that it will be trivial to find the median, and our problem will be resolved.

Now, how do we know that the above algorithm actually gives us the "good candidate" that we were seeking? We know that the median median key ranks higher than half of the other medians, which amount to $N/10$ of the keys. Since each of these medians is ranked above two other keys in its set (seen as yellow in the diagram below), we see that the median of medians is guaranteed to be ranked above $3N/10$ keys. We see that the median of medians also ranks lower than another $3N/10$ keys by a similar argument.




Since we have identified 3N/5 keys that we know are above or below our candidate, we can find the exact rank of our candidate by comparing it with the 2N/5 other keys whose position relative to the median of medians is as yet undetermined. When we have found the exact rank, we can find out whether this median median key is

above or below the true median. If our candidate is above the true median, then all keys above it are also above the true median and thus are eliminated from consideration. The same logic applies if our candidate is below the true median.

Thus, once we know the relation between the true median and our median median, we can eliminate either those keys we know to rank higher than the median median, or those we know to rank lower than it, and we have reduced our problem to a smaller one. In particular, here we will eliminate at least 3N/10 of the keys, so our problem will be reduced to one with at most 7N/10 keys.

**3.4 Showing that this procedure is linear in the number of comparisons**

Now that we have our procedure, we want to show that it can be done in at most cN comparisons, for some c independent of N. As is often the case in mathematics, we will introduce a bit of notation to simplify our equations.

Let $f(N)$ be the number of comparisons we have to make in order to find the median given N unsorted keys to start with. We seek an upper bound on $f(N)$ in the form cN.

We will use induction to find c. Recall that if N is small enough, then we can find the median in a linear number of comparisons. This is our base case. Now, let us assume by induction that for all groups of keys of size M, that is smaller than N, the median can be found in at most cM comparisons.

The first step of our procedure involves dividing N into groups of 5 and sorting them. As stated before, it takes 7 comparisons to sort 5 keys, and there are N/5 groups of 5, so it takes 7N/5 comparisons to complete this step. Let us define the number of comparisons that have to be done after this step to be $g(N)$.

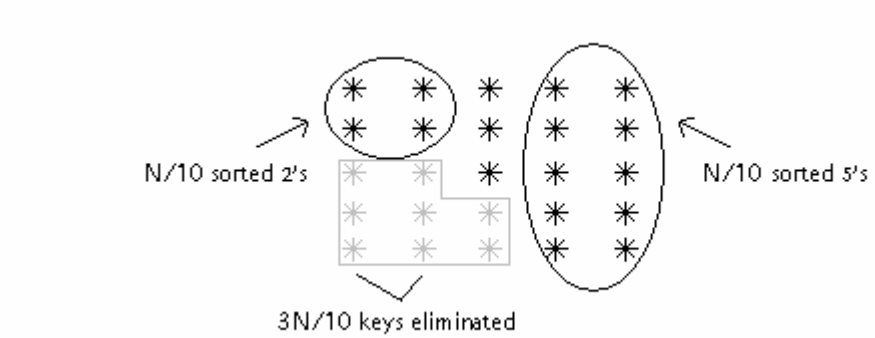
We have then that $f(N) = g(N) + 7N/5$

Now we have to find $g(N)$. Looking at the rest of our procedure, we see we now have to find the median of the medians of the groups of 5. We will call this the **median median key**. This step amounts to finding the median of N/5 keys, or in our new terminology, $f(N/5)$. We then have to compare this with the 2N/5 keys left after the first step to find its exact position, eliminate the proper group of 3N/10 keys, put the remaining 7N/10 keys into sorted blocks of 5 keys, and then find $g(7N/10)$. In this notation, the recursive nature of the procedure is quite evident.

So $g(N)$ is the sum of the number of comparisons required in all the steps in the previous paragraph. However, in the step where we find the position of the median median relative to the 2N/5 other keys, it turns out that we do not need to perform all 2N/5 comparisons. This is because we do not actually need to find its exact position, only on which side of the true median it lies. Furthermore, since the keys are in ordered

groups of 5, we already have some information. It turns out that only N/10 steps are required to figure out which side of the true median our candidate is on (see the exercises for more information).

We also need to figure out how many comparisons are required to put the remaining 7N/10 keys into sorted groups of 5 keys. Once again, we have some information from our previous sorting which will make this easier. Once we know where the median median goes, we will have N/10 sets of sorted 5's and N/10 sets of sorted 2's. These are the remnants of the sorted 5's after we have eliminated 3 of their members.




We can put these remaining keys into sorted blocks of 5 by taking the blocks of 2 and combining them with solitary keys into blocks of 5, separating groups of 2 to create single keys as necessary. Now, if you have done exercise 1 you will see that given a group of 5 keys, where two comparisons are already known, only 5 more comparisons are needed to put them in order. Now, since there were N/10 sets of 2 keys, a little thinking tells us that we can form N/25 groups of 5 from them in the manner described above. Since each group of 5 required 5 comparisons to sort, this entire process requires 5N/25 comparisons total.

Of course, it is important to note that we are looking at the very worst case scenario in the above calculations, so our value for $g(N)$ will involve an inequality, bounded above by the sum of the terms we described above. If you look back over the work we have done in this section you will see that we arrive at the following:

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

Now, we invoke our inductive hypothesis that $f(M) \leq cM$ for all $M < N$. This tells us that $f(N/5) \leq c(N/5)$ for some c independent of N.

We need to find an expression for $g(7N/10)$. We know that:

$$f(7N/10) = g(7N/10) + [\#\ of\ \ comparisons\ to\ \ put\ \ 7N/10\ keys\ in\ sorted\ \ groups\ of\ \ 5]$$

We saw that 5N/25 comparisons were needed to put N/5 keys in sorted groups of 5, when we began with N/10 groups of 2 and N/10 groups of 5. Here we will not use as detailed a calculation for the sake of simplicity (but we would get a lower bound if we

did). We know from above that $f(N) = g(N) + 7N/5$. So we shall just substitute $N = 7N/10$ into this equation. This gives us

$$f(7N/10) = g(7N/10) + 49N/50.$$

Now we employ some algebra to get the bound we are looking for. We use the fact that $f(7N/10) \leq c(7N/10)$ by induction, to get that $g(7N/10) + 49N/50 < c(7N/10)$ and thus

$$g(7N/10) < c(7N/10) - 49N/50$$

We see from our expressions for $f(N/5)$, $g(7N/10)$, and $g(N)$ above that:

$$g(N) < c(N/5) + N/10 + 5N/25 + c(7N/10) - 49N/50$$

This simplifies to:

$$g(N) < c(9N/10) - 17N/25$$

And if we substitute this into our expression for $f(N)$, we get:

$$f(N) < c(9N/10) + 18N/25$$

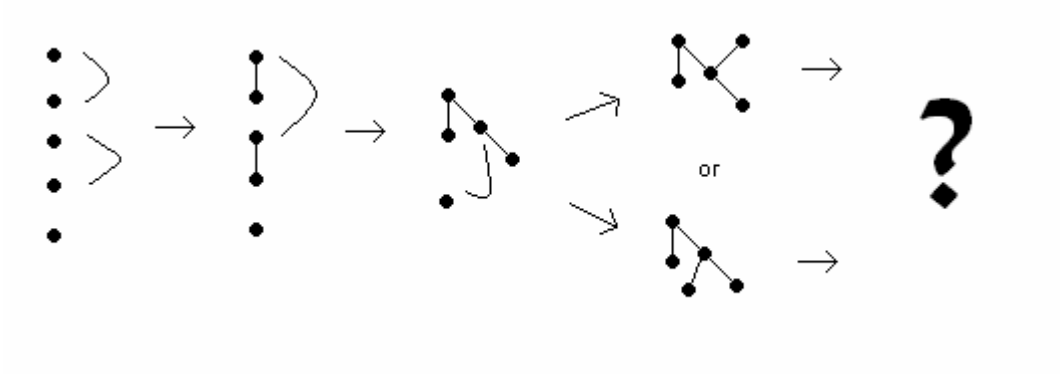
Now, we want to get that $f(N) < cN$. This would be true if $c(9N/10) + 18N/25 < cN$, which works if $c > 180/25$. So we see that $f(N)$ requires a linear number of comparisons provided that

$$\boxed{c > 7.2}$$

**Exercises**

*Exercise 1* We saw in section 2 many methods that can be used to sort a list of keys, and how to figure out about how many comparisons they require. However, for very small lists of keys, one can actually figure out the number of comparisons required by brute force. Show that you can sort 5 keys using only 7 comparisons. Use this to show how to start with two sorted 2's and one other and completely sort them using 5 comparisons.

*Hint: We solved this problem by making a drawing where the keys are points, and a relationship between two keys is designated by a line connecting them. Here is the first few steps:*




*Exercise 2* We saw how our median median key lies above 3N/10 keys and below another 3N/10 keys. In order to eliminate one of those groups we had to find which side of the true median our candidate lies on. Show how N/10 comparisons can be enough to locate the median median with respect to the true median in the worst case (to within 1 or 2).

*Exercise 3* Derive a similar bound dividing into sets of 7 keys rather than the 5 used here. What bounds do you get?

*Hint: As a first step, figure out how many comparisons are required to sort 7 keys, using 8.1 as a starting point.*

~Edited by Jacob Green