

Medians and Order Statistics

Let's say we have a set of n elements, for example: $\{4, 6, 5, 8, 7, 2, 9, 3\}$
We define the i th **order statistic** of a set of n elements to be the i th smallest element. For example:

- The **minimum** of a set of elements is the first order statistic, that is, $i = 1$. In the above example, this would be 2.
- The **maximum** of a set of elements is the n th order statistic, that is, $i = n$. In the above example, this would be 9.
- The **median**, or less formally, the “halfway point”, has two possibilities. In the case that n is odd, there is one unique element, that being the $\frac{n+1}{2}$ th order statistic. When n is even, there are two medians, the i th and $(i + 1)$ th order statistics where $i = n/2$. Regardless of the parity however, the median is both $\lfloor \frac{n+1}{2} \rfloor$ and $\lceil \frac{n+1}{2} \rceil$. In the above example, the median is 5 and 7.

For simplicity, when we say median we will be referring to the lower of the two.

1 Finding the Minimum and Maximum Elements Simultaneously

It is quite simple to find the minimum and maximum elements in $\Theta(n)$ time. At worst, we can find them independently, using $2n - 2$ comparisons. But we can do much better than that:

```
MIN-AND-MAX( $A, n$ )
1  if  $n = 0$ 
2    then return NIL, NIL
3  if  $n = 1$ 
4    then return  $A[1], A[1]$ 
5   $N_{min} \leftarrow$  smaller of  $\{A[1], A[2]\}$ 
6   $N_{max} \leftarrow$  larger of  $\{A[1], A[2]\}$ 
7   $i \leftarrow 3$ 
8  while  $i < n$ 
9    do Compare the  $i$ th and  $(i + 1)$ th elements, then
```

```

.           make a comparison between the larger and  $N_{max}$ ,
.           and the smaller one and  $N_{min}$ ,
.           assigning them to  $N_{max}$  and  $N_{min}$  if they
.           are larger or smaller respectively
10           $i \leftarrow i + 2$ 
11  if  $i = n$ 
12      then Compare the  $n$ th element with  $N_{min}$  and
.            $N_{max}$ , updating which ever is necessary.
13  return  $N_{min}, N_{max}$ 

```

This method uses $3\lfloor n/2 \rfloor$ comparisons.

2 The Selection Problem

The selection problem is defined as follows:

Given a set A of n (distinct) elements and a number i , with $1 \leq i \leq |A|$, the problem is to find an element x in A that is larger than exactly the other $i - 1$ elements in the set.

2.1 Variants of the Selection Problem

Some variants of the selection problem are listed as follows:

1. Find the k th largest element from A .
2. Find the first r smallest elements from A ($r \leq |A|$).
3. Find the first r largest elements from A ($r \leq |A|$).

2.2 The Linear-Time Selection Algorithm

The SELECTION algorithm finds the k th smallest element x from A , $1 \leq k \leq n$.

1. Divide the n elements of the input sequence into $\lceil n/5 \rceil$ groups of 5 elements each and the last group consists of the remaining $n \bmod 5$ elements.
2. Find the median of each of the $\lceil n/5 \rceil$ groups by any sorting algorithm. (If the number of elements in a group is even, take the larger one of the two medians).
3. Use SELECTION to find the median x of the the median sequence of $\lceil n/5 \rceil$ elements in Step 2 recursively.
4. Partition the set A into three disjoint sets using the median x of the median sequence, then

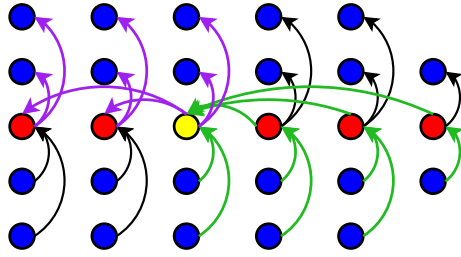


Figure 1: Bounds on the elements greater and smaller than x . The yellow dot is x . The green arrows are relationships that we know are greater than x , the purple arrows are relationships that we know are smaller than x , the black arrows we can not make any assumptions about in relationship to x .

$$\begin{aligned} R_1 &= \{a_i | a_i < x\}, \\ R_2 &= \{a_i | a_i = x\}, \text{ and} \\ R_3 &= \{a_i | a_i > x\}. \end{aligned}$$

5. If $|R_1| \geq k$ then use SELECTION to find the k th smallest element in R_1 recursively; if $|R_1| + |R_2| \geq k$ then return x ; otherwise, use SELECTION to find the $(k - |R_1| - |R_2|)$ th smallest element in R_3 recursively.

2.3 Analysis of Time Complexity of Selection

To analyse the running time of algorithm SELECTION, it is required to determine a lower bound on the number of elements that are greater than the partitioning element x .

Using Figure 1 as a visualisation, we can see that there is a minimum bound on the elements greater than x , that being that at least half of the $\lceil n/5 \rceil$ groups contribute 3 elements each, except for the group containing x , and the last group if n is not divided evenly by 5. Hence, the minimum bound of elements greater than x is:

$$3 \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$$

Similarly, the lower bound of the elements less than x is also $3n/10 - 6$. Thus, at worst case, Step 5 will call SELECTION recursively $7n/10 + 6$ times.

Steps 1, 2 and 4 all run in $O(n)$ time, noting that Step 2 consists of $O(n)$ calls to insertion sort on sets of size $O(1)$. Step 3 takes $T(\lceil n/5 \rceil)$ and Step 5 takes $T(7n/10 + 6)$. We can therefore formulate the following recursion:

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 80 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n > 80 \end{cases}$$

Using substitution, we can prove that this does indeed run in $\Theta(n)$ time.

3 Quick-sort - An Application of the Selection Algorithm

Let a_1, a_2, \dots, a_n be a real sequence. The basic steps of quick-sort are as follows:

1. Pick an element $x = a_i$ randomly (usually the first element a_1) as the **pivot element**
2. Partition the sequence into two disjoint subsequences, $S_1 = \{a_i : a_i \leq x\}$ and $S_2 = \{a_j : a_j > x\}$
3. Sort S_1 and S_2 recursively, using quick-sort.
4. Merge the sorted S_1 , $\{x\}$, and S_2 . The original sequence is sorted.

The running time of the quick-sort algorithm is heavily affected by the choice of the pivot element. For example, consider a case where the sequence is sorted initially, if we always pick the first element as the pivot element, what happens to the running time? If the median of the sequence is chosen as the pivot element, is the time complexity improved?