

ORDER STATISTICS

Reading: CLR Chapter 10

Terminology

Let S be a set of n elements (necessarily distinct).

The i th **order statistic** of S is the i th smallest element (i.e., the element that is larger than exactly $i - 1$ other elements in the set). Such an element is said to have **rank** i .

The **minimum** of S is the first order statistic (element with rank 1).

The **maximum** of S is the n th order statistic (element with rank n).

The **median(s)** of S is (are) the element(s) with rank $(n+1)/2$ and $(n+1)/2$.

Example: $B = \{43 \ 5 \ 17 \ 91 \ 2 \ 42 \ 19 \ 72 \ 37 \ 3\}$

- Minimum of B :
- Maximum of B :
- Medians of B :
- Rank of 17:

The Selection Problem

Specification:

```
Select(A, i)
  Return the ith order statistic from an array of n (distinct) elements.
```

Trivial algorithm:

```
Select(A, i)
  Sort(A)
  return A[i]
```

Can obviously be done in $\Theta(n \lg(n))$ time.

The main question of today's lecture: *Can we do better?*

Important special cases:

- Can find minimum or maximum with $n - 1$ comparisons.
 - Can find minimum *and* maximum with $3 \lceil n/2 \rceil$ comparisons.
 - For any fixed $k \geq 1$, can select *k*th or $(n - k)$ th element in $\Theta(n)$ time by *k* applications of minimum/maximum + deletion.
-
-

Selection in Expected Linear Time

Key idea: Use randomized quicksort-like partitioning, but only explore *one* partition.

```
Randomized-Select(A, lo, hi, i)
  if lo = hi then
    return A[lo] {i guaranteed = lo = hi at this point}
  split <- Randomized-Partition(A, lo, hi)
  lower_length <- (split - lo) + 1
  if i < lower_length then
    return Randomized-Select(A, lo, split, i)
  return Randomized-Select(A, split + 1, hi, i - lower_length)
```

Can derive that the expected running time is $T(n) = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + \Theta(n)$.

(See CLR for details.)

Can use the substitution method to show that $T(n) = cn$ is a solution to the above recurrence.

Notes:

The substitution method uses a form of induction. If we assume that the smaller problems have a given running time and we can show that the whole problem also has that running time, then that running time is a valid solution of the recurrence.

While the substitution method shows that $T(n) = cn$ is a solution to $T(n) = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + \Theta(n)$,

it cannot show that this is a solution to the quicksort recurrence $T(n) = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + \Theta(n)$.

(Try it and see!)

Selection in Worst Case Linear Time

Presentation in CLR is confusing. I prefer the following explanation.

We will consider the **median-of-median-of-c** algorithm, where c is an integer constant ≥ 1 .

Median-of-Median-of- $c(c, A, i)$

1. View input array $A[1..n]$ as a two-dimensional array $B[1..c, 1..n/c]$. (If c does not evenly divide n , can always pad the array with extra large elements that won't affect results). Sort each column using any method (even quicksort). Since each column contains c elements, this step is linear, not quadratic. After this step, the row $B[(c+1)/2, 1..n/c]$ are medians of their respective columns.
2. Use Median-of-Median-of- $c(c, B[(c+1)/2, 1..n/c], (n/c+1)/2)$ to find median of medians mm . At least $1/4$ elements are $\geq mm$; at least $3/4$ elements are $< mm$. (Can see this by imagining that the columns are sorted from left to right by their medians. Important: we only imagine this sorting, we don't actually do it!)
3. Partition A around mm , guaranteeing that mm ends up in the high partition. Let k be the number of elements in the low partition and $(n - k)$ the number of elements in the high partition.
4. If $i \leq k$, use Median-of-Median-of- c with index i on the low partition;
If $i > k$, use Median-of-Median-of- c with index $i - k$ on the high partition;

Analysis:
