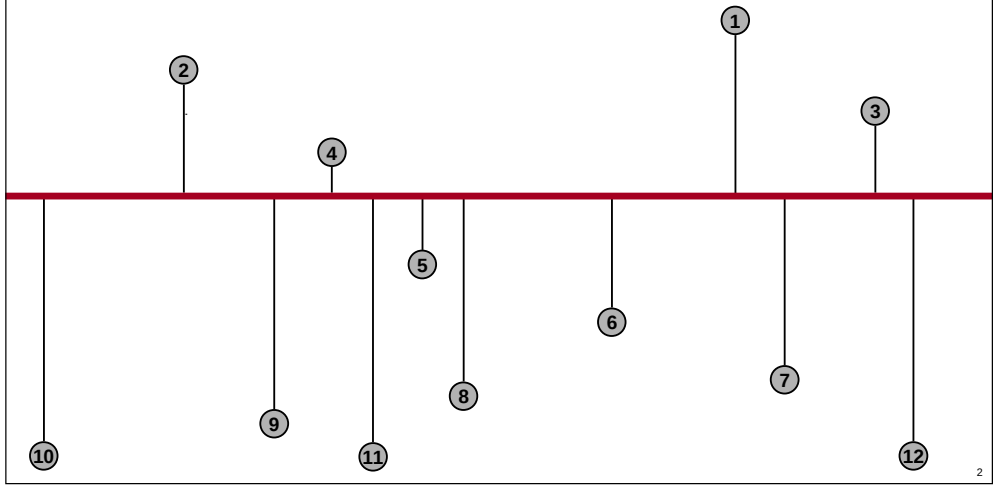
Linear Time Selection

These lecture slides are adapted from CLRS 10.3.

Where to Build a Road?

Given (x,y) coordinates of N houses, where should you build road parallel to x-axis to minimize construction cost of building driveways?

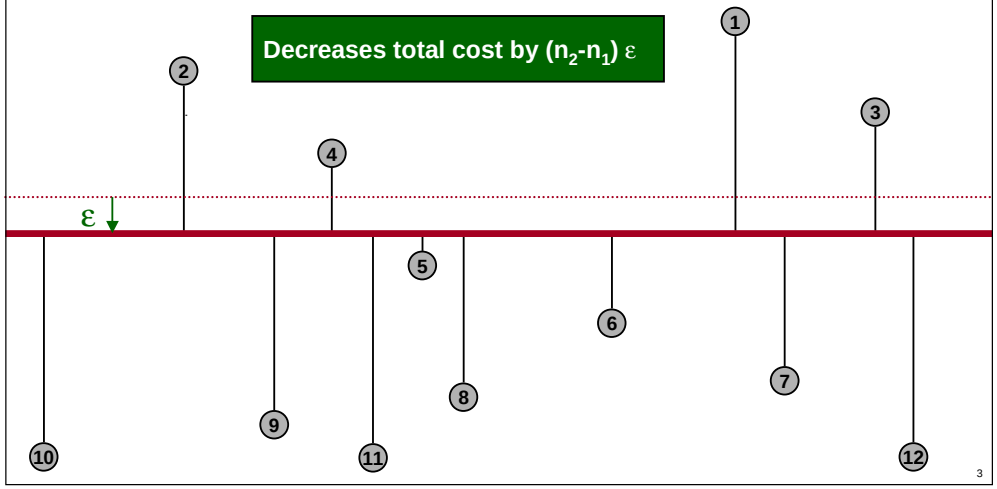


Where to Build a Road?

Given (x,y) coordinates of N houses, where should you build road parallel to x-axis to minimize construction cost of building driveways?

- n_1 = nodes on top.
- n_2 = nodes on bottom.

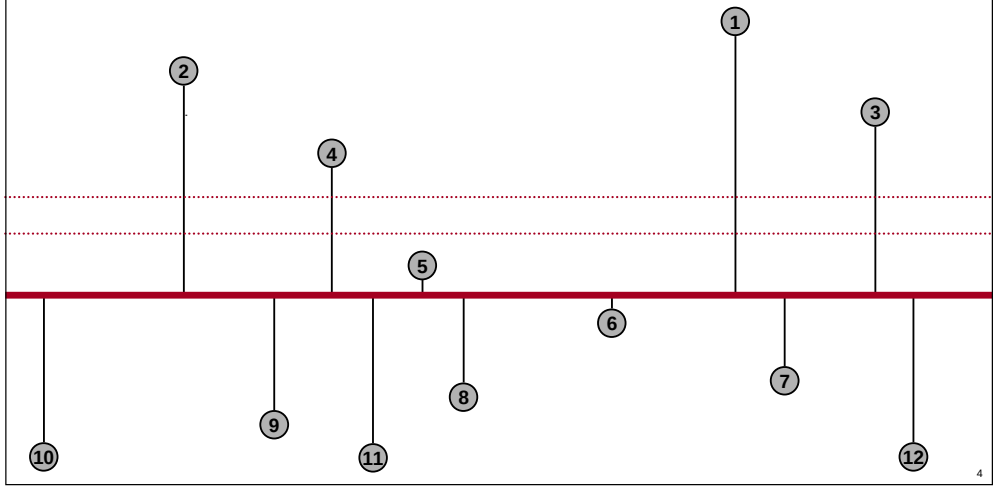
Decreases total cost by $(n_2-n_1)\ \varepsilon$



Where to Build a Road?

Given (x,y) coordinates of N houses, where should you build road parallel to x-axis to minimize construction cost of building driveways?

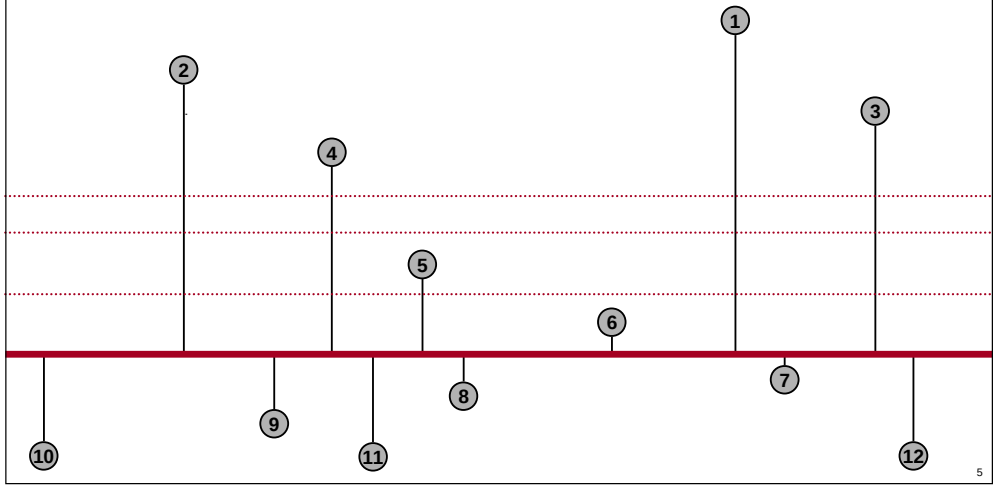
- n_1 = nodes on top.
- n_2 = nodes on bottom.



Where to Build a Road?

Given (x,y) coordinates of N houses, where should you build road parallel to x-axis to minimize construction cost of building driveways?

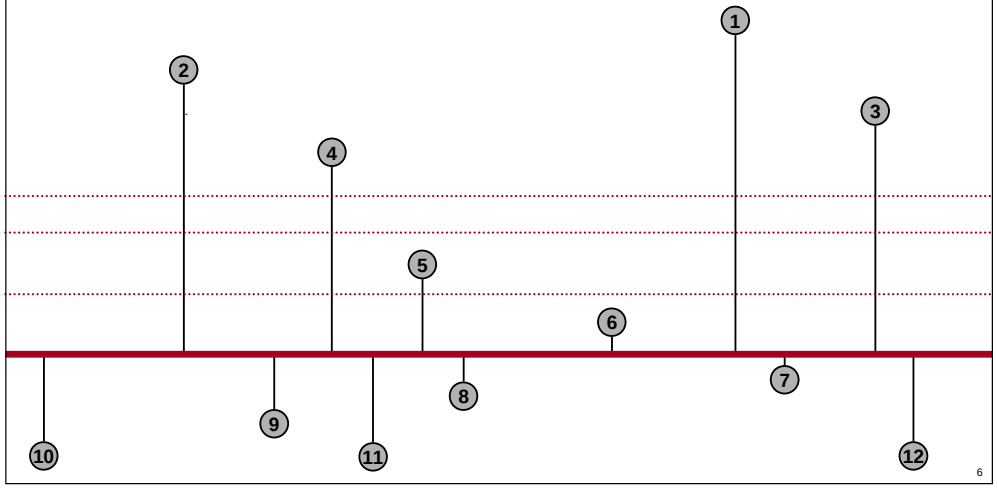
- n_1 = nodes on top.
- n_2 = nodes on bottom.



Where to Build a Road?

Given (x,y) coordinates of N houses, where should you build road parallel to x-axis to minimize construction cost of building driveways?

Solution: put street at median of y coordinates.



Order Statistics

Given N linearly ordered elements, find i^{th} smallest element.

- Min: i = 1
- Max: i = N
- Median: $i = \lfloor (N+1) / 2 \rfloor$ and $\lceil (N+1) / 2 \rceil$
- O(N) for min or max.
- O(N log N) comparisons by sorting.
- O(N log i) with heaps.

Can we do in worst-case O(N) comparisons?

- Surprisingly, yes. (Blum, Floyd, Pratt, Rivest, Tarjan, 1973)

Assumption to make presentation cleaner.

- All items have distinct values.

Select

Similar to quicksort, but throw away useless "half" at each iteration.

- Select i^{th} smallest element from $a_1, a_2, \ldots, a_N$.

Select (i^{th}, N, $a_1, a_2, \ldots, a_N$)

```
x ← Partition(N, a_1, a_2, ..., a_N)
k ← rank(x)

if (i == k)
   return x

else if (i < k)
   b[] ← all items of a[] less than x
   return Select(i^th, k-1, b_1, b_2, ..., b_{k-1})

else if (i > k)
   c[] ← all items of a[] greater than x
   return Select((i-k)^th, N-k, c_1, c_2, ..., c_{N-k})
```

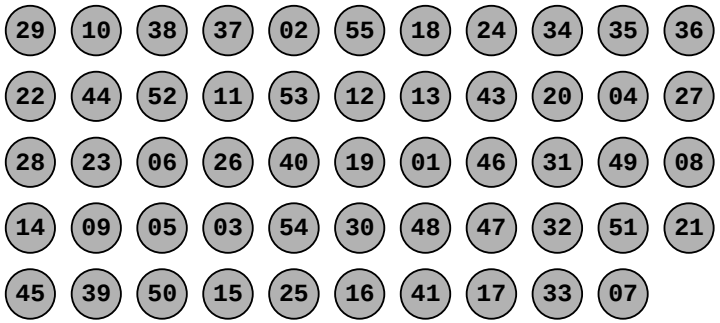
x = partition element

Want to choose x so that x is (roughly) the ith largest.

Partition

Partition().

- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.

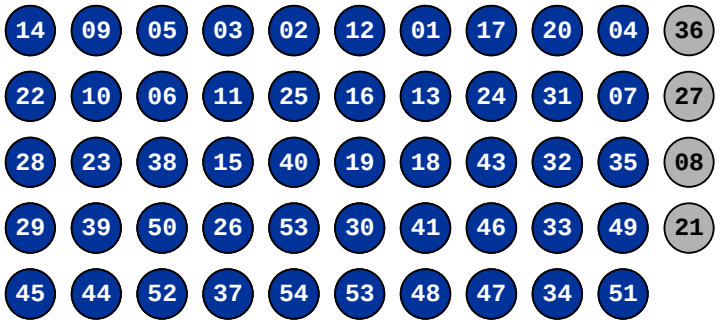


N = 54

Partition

Partition().

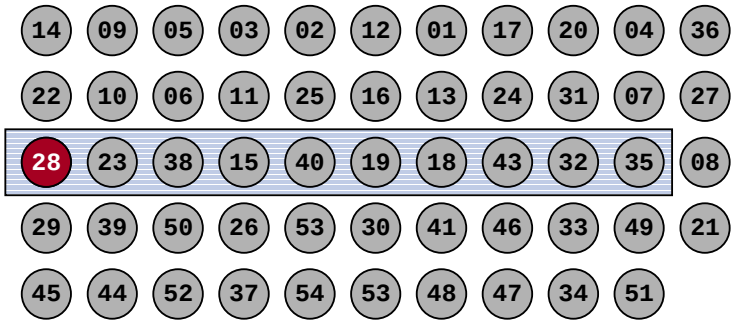
- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.
- Brute force sort each of the 5-element groups.



Partition

Partition().

- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.
- Brute force sort each of the 5-element groups.
- Find x = "median of medians" by Select() on ⌊N/5⌋ medians.



Select

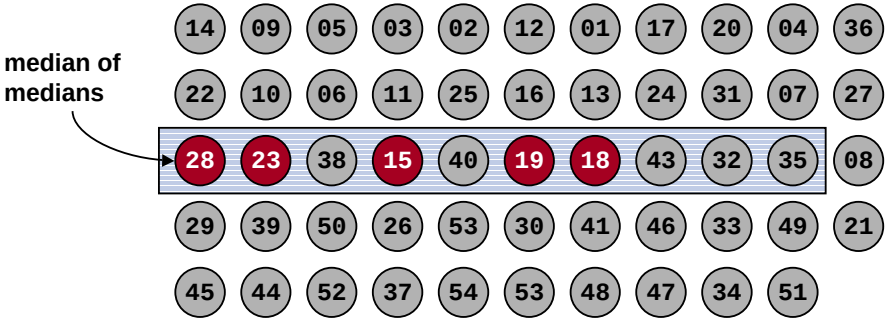
Select (i^{th}, N, a_1, a_2, . . . , a_N)

```
if (N is small) use mergesort

Divide a[] into groups of 5, and let
m_1, m_2, ..., m_{N/5} be list of medians.

x ← Select(N/10, m_1, m_2, ..., m_{N/5})   // median of medians

k ← rank(x)

if (i == k)       // Case 1
   return x

else if (i < k)  // Case 2
   b[] ← all items of a[] less than x
   return Select(i^th, k-1, b_1, b_2, ..., b_{k-1})

else if (i > k)  // Case 3
   c[] ← all items of a[] greater than x
   return Select((i-k)^th, N-k, c_1, c_2, ..., c_{N-k})
```

Selection Analysis

Crux of proof: delete roughly 30% of elements by partitioning.

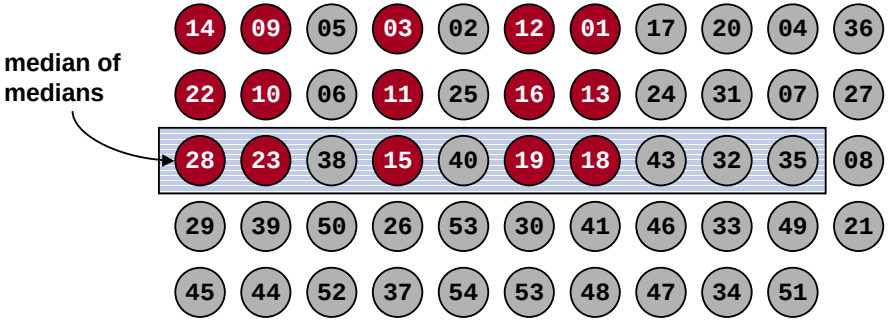
- At least 1/2 of 5 element medians ≤ x
  - at least $\lfloor \lfloor N / 5 \rfloor / 2 \rfloor = \lfloor N / 10 \rfloor$ medians ≤ x



Selection Analysis

Crux of proof: delete roughly 30% of elements by partitioning.

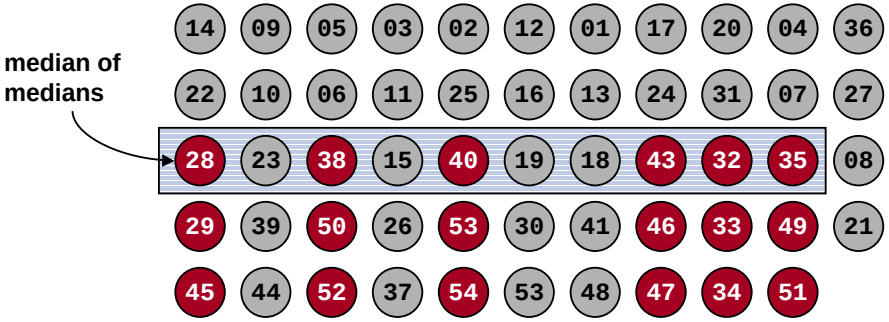
- At least 1/2 of 5 element medians ≤ x
  - at least $\lfloor \lfloor N / 5 \rfloor / 2 \rfloor = \lfloor N / 10 \rfloor$ medians ≤ x
- At least $3 \lfloor N / 10 \rfloor$ elements ≤ x.



Selection Analysis

Crux of proof: delete roughly 30% of elements by partitioning.

- At least 1/2 of 5 element medians ≤ x
  - at least $\lfloor \lfloor N / 5 \rfloor / 2 \rfloor = \lfloor N / 10 \rfloor$ medians ≤ x
- At least $3 \lfloor N / 10 \rfloor$ elements ≤ x.
- At least $3 \lfloor N / 10 \rfloor$ elements ≥ x.



Selection Analysis

Crux of proof: delete roughly 30% of elements by partitioning.

- At least 1/2 of 5 element medians ≤ x
  - at least $\lfloor \lfloor N / 5 \rfloor / 2 \rfloor = \lfloor N / 10 \rfloor$ medians ≤ x
- At least $3 \lfloor N / 10 \rfloor$ elements ≤ x.
- At least $3 \lfloor N / 10 \rfloor$ elements ≥ x.
  - ⇒ `Select()` called recursively (Case 2 or 3) with at most $N - 3 \lfloor N / 10 \rfloor$ elements.

C(N) = # comparisons on a file of size N.

$$C(N) \le \underbrace{C(\lfloor N/5 \rfloor)}_{\text{median of medians}} + \underbrace{C_{\le}(N - 3\lfloor N/10 \rfloor)}_{\text{recursive select}} + \underbrace{O(N)}_{\text{insertion sort}}$$

Now, solve recurrence.

- Apply master theorem?
- Assume N is a power of 2?
- Assume C(N) is monotone non-decreasing?

Selection Analysis

Analysis of selection recurrence.

- T(N) = # comparisons on a file of size ≤ N.
- T(N) is monotone, but C(N) is not!

$$T(N) \le \begin{cases} 20cN & \text{if } N < 50 \\ T(\lfloor N/5 \rfloor) + T(N - 3\lfloor N/10 \rfloor) + cN & \text{otherwise} \end{cases}$$

Claim: T(N) ≤ 20cN.

- Base case: N < 50.
- Inductive hypothesis: assume true for 1, 2, . . . , N-1.
- Induction step: for N ≥ 50, we have:

$$\begin{aligned} T(N) &\le T(\lfloor N/5 \rfloor) + T(N - 3\lfloor N/10 \rfloor) + cN \\ &\le 20c\lfloor N/5 \rfloor + 20c(N - 3\lfloor N/10 \rfloor) + cN \\ &\le 20c(N/5) + 20c(N) - 20c(N/4) + cN \\ &= 20cN \end{aligned}$$

For n ≥ 50,
$3\lfloor N / 10 \rfloor \ge N / 4$.

Linear Time Selection Postmortem

Practical considerations.

- Constant (currently) too large to be useful.
- Practical variant: choose random partition element.
  - O(N) expected running time ala quicksort.
- Open problem: guaranteed O(N) with better constant.

Quicksort.

- Worst case O(N log N) if always partition on median.
- Justifies practical variants: median-of-3, median-of-5.