Problem: Given an array E containing n elements with keys from some linearly ordered set, find an element with the k-th smallest key.

Why this is interesting?
- max, min, median, mean, ...

If the array is ordered (with $O(n \log n)$ time), then $E[k]$ is the answer.

Can we do better?
findMax(E, n)
1. max = E[0];
2. for (i = 1; i < n; i++)
3.    if(max < E[i])
4.       max = E[i];
5. return max;

It takes n-1 comparisons to find the largest key, that is better than O(n log n).

Is this the best we can do for finding the largest key by comparisons?
Yes.
- For n distinct keys, only one is the largest => n-1 losers.
- Each comparison generate only one loser => n-1 comparisons needed.
- If there are two or more nonlosers left when the algorithm terminates, it
can not be sure it has identified the max.
- **2ndLargest**
  apply findMax once and remove the max, then apply findMax again.

\[(n-1) + (n-2) = 2n - 3.\]

- **i-th key**

\[(n -1) + (n-2) + \ldots + (n-i)\]

- **Median** \(i=\frac{n}{2}\)

\[\sum_{i=1}^{\frac{n}{2}} (n-i) = (3/8) n^2 - n/4 \in O(n^2)\]

**Note:**
- This is even worse than sorting the array first.
- Finding the median seems to be the hardest selection problem.
Divide and conquer?

- Simple minded one: partitioning \( S \rightarrow S_1 \) and \( S_2 \), then the median is in the larger set, say \( S_2 \), and we gain by ignoring the smaller set.
  Then we do this recursively on \( S_2 \)!
  Wait, the median of \( S_2 \) is not the median of \( S \).

Algorithm select(\( S, k \))
1. Divide the \( n \) elements into \( n/5 \) groups of 5 elements each and at most one group made up of the remaining \( n \mod 5 \) elements.
2. Find the median of each \( n/5 \) groups
3. Use select recursively to find the median \( m^* \) of \( n/5 \) medians found in step 2
4. Partition using \( m^* \) as pivot:
   Compare each key in the sections \( A \) and \( D \) to \( m^* \).
   Let \( S_1 = C \cup \{ \text{keys from} \ A \cup D \text{that are smaller than} \ m^* \} \)
   Let \( S_2 = B \cup \{ \text{keys from} \ A \cup D \text{that are larger than} \ m^* \} \)
5. Divide and conquer:
   if \( k = \lceil |S_1| + 1 \rceil \)
      return \( m^* \); // because \( m^* \) is the \( k \)-th smallest key
   else if \( k \leq |S_1| \)
      return select(\( S_1, k \)); // the \( k \)-th smallest key of \( S \) is in \( S_1 \), and is the \( k \)-th smallest key in \( S_1 \).
   else
      return select(\( S_2, k - |S_1| - 1 \)); // the \( k \)-th smallest key of \( S \) is in \( S_2 \), and
                       // it is the \( k-|S_1|-1 \)-th smallest key in \( S_2 \).
For simplicity, let $n = 5(2r+1)$ and integer $r > 0$.

1. Find medians of 5 keys: 6 comparisons
2. There are $n/5$ sets: $6(n/5)$
3. Recursively find the median $m^*$ of the medians: $T(n/5)$
4. Compare all keys in section A and D to $m^*$: $4r$ comparisons.
5. Recursively subset S1 or S2: $T(7r + 2)$

B and C section each has $3r+2$ elements, plus
4$r$ elements from A and D. $r = n/10$.

$$T(n) = T(7n/10) + T(n/5) + (6/5)n + (4/10)n$$
$$= T(7n/10) + T(n/5) + (7/5)n$$
Important observation:
\[(n/5) + (7n/10) = (9/10)n < n\]
Parts add up to less than the whole.
This implies a decreasing geometric series
when untangle the recurrence equation.

\[T(n) = T(7n/10) + T(n/5) + 1.6n\]

\[T(n) = 1.6n (1 + 9/10 + (9/10)^2 + \ldots) \leq 1.6n \sum_{i=0}^{\infty}(9/10)^i = 1.6n (1/(1-9/10)) = 16n\]
More generally, for recurrence equation

\[ T(n) = cn + T(a \cdot n) + T(b \cdot n), \]

if \( a+b < 1 \), then

\[ T(n) \leq c \left[ \frac{1}{1-a-b} \right]^n \]

Question: will algorithm \textit{select} still be linear if we divide the keys into sets of 3, or 7?

---

Selection algorithms:

For median selection,

- Blum, Floyd, Pratt, Rivest & Tarjan
  
  5.34n
- Dor and Zwick (1995) 2.95n

Dor and Zwick (1996) (2+\( \epsilon \))n

\( \epsilon \approx 2^{-80} \) used in proof of lower bound.
Lower bound for median selection

- Decision tree approach
  - nodes ↔ comparisons of keys
  - leaves ↔ output. Since any key can happen to be the median, there are n leaves.

  Height ↔ max # of comparisons
  \[ \geq \lg (\text{# of leaves}) = \lg(n) \]

  This lower bound is not tight, because we know the lower bound of findMax is already n-1.

Lower bounds: worst-case

- Adversary argument approach
  adversary will enforce worst-case scenario.

  Algorithm must know the relation of every other key to the median, either directly or indirectly. Every key has to be compared at least once (otherwise no way to know its relation to others), this takes n-1 comparisons.

  Crucial comparison: a comparison of key x is crucial if it is the first comparison where x > y, for some y >= median, or x < y, for some y <= median. Namely, this comparison is enough to infer either x > median or the opposite.
Lower bound: best case

![Diagram of median comparison]

Lower bound:

Uncrucial comparisons: $x > y$ but $y < \text{median}$ or $x < y$ but $y > \text{median}$.

Status about a key: $L$, $S$, or $N$.

Worst cases:

These are unlucky (i.e., uncrucial) comparisons.

- $\text{compare}(N:N) \rightarrow L:S$ or $S:L$
- $\text{compare}(L:N) \rightarrow L:S$
- $\text{compare}(N:L) \rightarrow S:L$
- $\text{compare}(S:N) \rightarrow S:L$
- $\text{compare}(N:S) \rightarrow L:S$

Each compare will consume one or two $N$s, and generate one $L$ (or $S$). There can be $(n-1)/2$ keys with status $L$ (or $S$). Therefore, adversary can at least enforce $(n-1)/2$ uncrucial comparisons. Total number of comparisons $\geq (n-1) + (n-1)/2$

**Theorem 5.3** Any algorithm to find the median of $n$ keys (for odd $n$) by comparison of keys must do at least $3n/2 - 3/2$ comparisons in the worst case.