A Search Problem

**Input:**
- A key $K$.

**Output:**
- Does $K$ appear in $A$? **YES** or **NO**.
- If **YES**: an index $i$ such that $A[i] = K$.

**Method:**
- **Comparisons** between $K$ and the keys in the array.
A Search Game

- **Player 1:** Selects a number \( x \) in the range \([1..n]\).

- **Player 2:** Searches for \( x \) with comparisons \( x \leq i \) for some \( 1 \leq i \leq n \).

- **Player 2 Goal:** Minimize number of comparisons until finding \( x \).
  - In the worst case or in the average case.
  - As a function of \( n \).
Equivalency

- $x \leq i$ is "equivalent" to $K \leq A[i]$.

- Algorithms can be converted from one model to another while preserving the complexity.

- It is easier to design algorithms in the search game model.

- It is easier to prove lower bounds in the search game model.
Sequential Search

Sequential-Search \((n, x)\)

\[
i = 0
\]

repeat

\[
i = i + 1
\]

until \(x \leq i\) (*) comparison (*)

return \(i\)
Sequential Search – Correctness

- **Induction hypothesis:**
  - $i \leq x \leq n$ after $i - 1$ comparisons with a NO answer.

- **Termination:**
  - If $x \leq i$ then necessarily $x = i$.
  - Eventually $x \leq n$. 
Sequential Search – Complexity

- **n comparisons** in the worst case when \( x = n \).
  - Possible \( n - 1 \) comparisons since there is no need for the last question when \( x = n \).

- Could be only 1 comparison when \( x = 1 \).

- \( (n + 1)/2 \) comparisons on average for a random \( x \) selected with a uniform distribution from the range \([1..n]\):

\[
\frac{1}{n} (1 + 2 + \cdots + n) = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2}.
\]
Binary Search

**Binary-Search** \((n,x)\)

\[
\ell = 1 \\
u = n \\
\text{while } \ell < u \\
m = \lfloor (u + \ell)/2 \rfloor \\
\text{if } x \leq m \quad (\text{* comparison *}) \\
\quad \text{then } u = m \\
\quad \text{else } \ell = m + 1 \\
\text{return } \ell
\]
Let $u_i$ and $\ell_i$ be the values of $u$ and $\ell$ after iteration $i$ of the algorithm and let $\Delta_i = u_i - \ell_i + 1$.

Initially $u_0 = n$, $\ell_0 = 1$, and $\Delta_0 = n$.

**Claim:** $\Delta_{i+1} \leq \left\lceil \frac{\Delta_i}{2} \right\rceil$ for $i \geq 0$.

**Corollary:** $\Delta_k = 1$ for $k = \lceil \log n \rceil$.  

Correctness: By induction, always $\ell_i \leq x \leq u_i$. At the end, $\ell_i = u_i$ and therefore $x = \ell_i = u_i$.

Complexity: $\lceil \log n \rceil$: the number of iteration.
Let $u_i$ and $\ell_i$ be the values of $u$ and $\ell$ after iteration $i$ of the algorithm and let $\Delta_i = u_i - \ell_i + 1$.

Initially $u_0 = n$, $\ell_0 = 1$, and $\Delta_0 = n$.

Claim: $\Delta_{i+1} \leq \left\lceil \frac{\Delta_i}{2} \right\rceil$ for $i \geq 0$.

Corollary: $\Delta_k = 1$ for $k = \lceil \log n \rceil$.

Correctness:
- By induction, always $\ell \leq x \leq u$.
- At the end, $\ell = u$ and therefore $x = \ell = u$.

Complexity:
- $\lceil \log n \rceil$: the number of iteration.
Adversary Player I

- **Does not** select $x$ at the beginning of the game. Instead, maintains a set of candidates $S$ for $x$.

- Given a search question:
  - $S(Y)$ – the set of candidates if the answer is **YES**.
  - $S(N)$ – the set of candidates if the answer is **NO**.

- **Observation:** $S = S(Y) \cup S(N)$.

- The adversary answer rule:
  - **YES** if $|S(Y)| \geq |S(N)|$.
  - **NO** if $|S(Y)| < |S(N)|$. 
Example

- **Input:** \( n = 34 \) \( \Rightarrow \) \( (* \ x \in [1..34] *) \).

- **Search:**
  - Q1: \( x \leq 13 \Rightarrow A1: \text{NO} \) \( (* \ x \in [14..34] *) \).
  - Q2: \( x \leq 26 \Rightarrow A2: \text{YES.} \) \( (* \ x \in [14..26] *) \).
  - Q3: \( x \leq 18 \Rightarrow A3: \text{NO.} \) \( (* \ x \in [19..26] *) \).
  - Q4: \( x \leq 23 \Rightarrow A4: \text{YES.} \) \( (* \ x \in [19..23] *) \).
  - Q5: \( x \leq 20 \Rightarrow A5: \text{NO.} \) \( (* \ x \in [21..23] *) \).
  - Q6: \( x \leq 22 \Rightarrow A6: \text{YES.} \) \( (* \ x \in [21..22] *) \).
  - Q7: \( x \leq 21 \Rightarrow A7: \text{YES.} \) \( (* \ x \in [21..21] *) \).

- **Output:** \( x = 21 \).
Theorem

There exists \(1 \leq x \leq n\) for which the adversary forces the second player to ask at least \(\lceil \log_2 n \rceil\) comparisons.

Proof:

- Assume the second player asks \(k\) comparisons.
- Let \(S_i\) be the set of candidates after \(i\) comparisons.
- In particular, \(|S_0| = n\) and \(|S_k| = 1\).
- By the observation, \(\frac{|S_{i+1}|}{|S_i|} \geq (1/2)\) for \(1 \leq i \leq k - 1\).
- \(\lceil \log_2 n \rceil\) rounds are required to decrease \(n\) to 1 by halving.
- Therefore, \(k \geq \lceil \log_2 n \rceil\).
Remarks

- This is a **worst case** bound implying that no algorithm can guarantee less comparisons for all values of $x$.

- The theorem holds for a **stronger** Player 2. One that can ask any **YES/NO** questions. For example,
  - Is $x$ even?
  - Is $x \in \{1, 2, 3, 5, 8, 13, 21, 34, 55\}$?
Find the Minimum Or the Maximum


**Output:**
- **Minimum:** A key $K$ from the array such that $K \leq A[i]$ for $1 \leq i \leq n$.
- **Maximum:** A key $K$ from the array such that $K \geq A[i]$ for $1 \leq i \leq n$.

**Method:** By comparisons between any two keys from the array.

**Goal:** Minimize number of key comparisons.
Find the Minimum

Trivial-Find-Min($A[1], \ldots, A[n]$)

$K := A[1]$

for $i = 2$ to $n$

if $K > A[i]$ (* comparison *)

then $K := A[i]$

return $K$

**Correctness:**

- $K = \min \{A[1], \ldots, A[i]\}$ after round number $i$.
- $K = \min \{A[1], \ldots, A[n]\}$ after $n - 1$ rounds.

**Complexity:** Exactly $n - 1$ comparisons.
Key idea: Any entry in the array could be the minimum.

Data structure:
- $B$ - Candidates for minimum.
- $R$ - Cannot be minimum.
Initially: $B = \{A[1], \ldots, A[n]\}$ and $R = \emptyset$.
At the end: $|B| = 1$ and $|R| = n - 1$.

Answer rule:
- $(R_1 : R_2) \Rightarrow$ Any consistent answer.
- $(B : R) \Rightarrow B < R$.
- $(B_1 : B_2) \Rightarrow B_1 < B_2$; transfer $B_2$ from $B$ to $R$. 
The adversary forces any algorithm that finds the minimum (or the maximum) to perform at least \( n - 1 \) comparisons.

Proof:
- A **useful** comparison decreases the size of \( B \).
- Only \((B_1 : B_2)\) is a useful comparison.
- Each useful comparison decreases the size of \( B \) by 1.
- \( n - 1 \) useful comparisons are required to decrease the size of \( B \) from \( n \) to 1.
Parallel Find the Minimum or the Maximum

Round:
- May contain several comparisons.
- Each key may participate in at most one comparison.

Goals:
- Minimize number of rounds.
- Minimize number of comparisons.
Parallel Find the Minimum for \( n = 2^k \)

Parallel-Find-Min\((A[1], \ldots, A[n])\)

if \( n = 1 \) then return \( A[1] \)

for \( i = 1 \) to \( n/2 \)

if \( A[i] > A[i + (n/2)] \) (* comparison *)

then \( A[i] \leftrightarrow A[i + (n/2)] \)

return Parallel-Find-Min\((A[1], \ldots, A[n/2])\)
Complexity

- **Number of comparisons:**
  - \( \frac{n}{2} + \frac{n}{4} + \cdots + 1 = n - 1 \).
  - The same as in Trivial-Find-Min
  - Optimal.

- **Number of rounds:**
  - \( \log_2 n \); number of recursive calls required to decrease the size of the array \( A \) from \( n \) to 1 by halving.
  - Optimal.
  - **Remark:** In Trivial-Find-Min there are \( n - 1 \) rounds.
Theorem

The adversary forces any algorithm that finds the minimum (or the maximum) to perform at least $\lceil \log_2 n \rceil$ rounds.
Theorem

The adversary forces any algorithm that finds the minimum (or the maximum) to perform at least $\lceil \log_2 n \rceil$ rounds.

Proof:

- At most $\lfloor |B/2| \rfloor$ useful comparisons per round since any key may participate in only one comparison.

- $\lceil \log_2 n \rceil$ rounds are required to decrease the size of $B$ from $n$ to 1 by halving.
Find-Min-and-Max\( (A[1], \ldots, A[n]) \)

Find-Min\( (A[1], \ldots, A[n]) \)

Find-Max\( (A[2], \ldots, A[n]) \)
Find the Minimum & the Maximum (Sol. I)

Find-Min-and-Max($A[1], \ldots, A[n]$)

Find-Min($A[1], \ldots, A[n]$)

Find-Max($A[2], \ldots, A[n]$)

**Complexity:**

- $(n - 1) + (n - 2) = 2n - 3$ comparisons.
- At most $2 \log_2 n$ rounds using Parallel-Find-Min and Parallel-Find-Max.
Find the Minimum & the Maximum for $n = 2^k$ (Sol. II)

Parallel-Find-Min-and-Max($A[1], \ldots, A[n]$)
for $i = 1$ to $n/2$
    if $A[i] > A[i + (n/2)]$ (* comparison *)
        then $A[i] \leftrightarrow A[i + (n/2)]$
Parallel-Find-Min($A[1], \ldots, A[n/2]$)
Parallel-Find-Max($A[n/2 + 1], \ldots, A[n]$)
Find the Minimum & the Maximum for $n = 2^k$ (Sol. II)

Parallel-Find-Min-and-Max($A[1], \ldots, A[n]$)
for $i = 1$ to $n/2$
  if $A[i] > A[i + (n/2)]$  (* comparison *)
    then $A[i] \leftrightarrow A[i + (n/2)]$
Parallel-Find-Min($A[1], \ldots, A[n/2]$)
Parallel-Find-Max($A[n/2 + 1], \ldots, A[n]$)

Complexity:

- $\frac{n}{2} + 2 (\frac{n}{2} - 1) = \frac{3n}{2} - 2$ comparisons.
- $1 + \log_2(n/2) = \log_2 n$ rounds.
Adversary Strategy – Data Structure

- $\mathcal{N}$ - Candidates for **either** maximum or minimum.
- $\mathcal{H}$ - Candidates **only** for maximum.
- $\mathcal{B}$ - Candidates **only** for minimum.
- $\mathcal{R}$ - Can be **neither** maximum nor minimum.

Initially: $\mathcal{N} = \{A[1], \ldots, A[n]\}$ and $\mathcal{H} = \mathcal{B} = \mathcal{R} = \emptyset$.

At the end: $|\mathcal{H}| = 1$, $|\mathcal{B}| = 1$, $|\mathcal{N}| = 0$, $|\mathcal{R}| = n - 2$. 
Adversary Strategy – Answer Rule

- $(R_1 : R_2) \Rightarrow A$ consistent answer.
- $(R : H) \Rightarrow R < H.$
- $(B : R) \Rightarrow B < R.$
- $(N : R) \Rightarrow N < R$ and $N \rightarrow B.$
- $(B : N) \Rightarrow B < N$ and $N \rightarrow H.$
- $(N : H) \Rightarrow N < H$ and $N \rightarrow B.$
- $(N_1 : N_2) \Rightarrow N_1 < N_2$ and $N_1 \rightarrow B$ and $N_2 \rightarrow H.$
- $(B : H) \Rightarrow B < H.$
- $(B_1 : B_2) \Rightarrow B_1 < B_2$ and $B_2 \rightarrow R.$
- $(H_1 : H_2) \Rightarrow H_1 < H_2$ and $H_1 \rightarrow R.$
There Is No Better Algorithm

Theorem

The adversary forces any algorithm that finds the minimum and the maximum to perform at least \( \left\lceil \frac{3n}{2} \right\rceil - 2 \) comparisons.
Theorem

The adversary forces any algorithm that finds the minimum and the maximum to perform at least $\lceil \frac{3n^2}{2} \rceil - 2$ comparisons.

Proof:

- Non-max and non-min keys: $\mathcal{N} \rightarrow \{B, H\} \rightarrow \mathcal{R}$.
- $(N_1 : N_2), (B_1 : B_2)$, and $(H_1 : H_2)$ are useful.
- $(N_1 : N_2)$ is better than $(N : R), (B : N), (N : H)$.
- The rest of the comparisons are not useful.
- Emptying $\mathcal{N}$ requires at least $\lceil \frac{n}{2} \rceil$ useful comparisons.
- The fastest way to leave one key in both $B$ and $H$ requires at least $n - 2$ useful comparisons.
Find the First and the Second


- **Output:** Keys $A[n]$ and $A[n - 1]$:
  - $A[n - 1] \geq A[i]$ for $1 \leq i \leq n - 2$.

- **Goal:** Minimize number of comparisons between keys.
Find the First and the Second – Trivial Solution

Trivial-Find-First-and-Second($A[1], \ldots, A[n]$)

Trivial-Find-Max($A[1], \ldots, A[n]$)

Trivial-Find-Max($A[1], \ldots, A[n-1]$)


Correctness:
By definition.

Complexity:
Exactly ($n - 1$) + ($n - 2$) = $2n - 3$ comparisons.
Find the First and the Second – Trivial Solution

Trivial-Find-First-and-Second\((A[1], \ldots, A[n])\)

\[
\text{Trivial-Find-Max}(A[1], \ldots, A[n])
\]

\[
\text{Trivial-Find-Max}(A[1], \ldots, A[n - 1])
\]

\[
\text{return } (A[n] \geq A[n - 1])
\]

- **Correctness**: By definition.
- **Complexity**: Exactly \((n - 1) + (n - 2) = 2n - 3\) comparisons.
**Observation:** Only “losers” to **First** can be **Second**.
- Trivial solution: \( n - 1 \) possible losers to **First**.
- Parallel solution: \( \lceil \log n \rceil \) possible losers to **First**.

**Parallel Algorithm:**
- **First**: Maximum of the original array.
- **Second**: Maximum of the \( \lceil \log_2 n \rceil \) losers to **First**.
Better Solution – Complexity and Optimality

- **Complexity:**
  - \((n - 1)\) comparisons to find First.
  - \((\lceil \log_2 n \rceil - 1)\) comparisons to find Second.
  - \(n + \lceil \log_2 n \rceil - 2\) comparisons to find First and Second.

- **Optimality:** There exists an adversary strategy that forces any algorithm that finds First and Second to perform at least \(n + \lceil \log_2 n \rceil - 2\) comparisons.
The $k$-Selection Problem

- **Input:**
  - An integer $k$, $1 \leq k \leq n$.

- **Output:** The key $A[i]$ that is the $k$ smallest key in $A$.

- **Goal:** Minimize number of **comparisons** between keys.

- **Median:** $k = \lceil n/2 \rceil$ for an odd $n$.

- **Assumption:** all the $n$ keys are distinct.
Example

[21, 34, 8, 5, 55, 13, 1, 3, 89, 2, 123]

- 5 is the 4\textsuperscript{th} smallest and the 8\textsuperscript{th} largest.
- 13 is the median: the 6\textsuperscript{th} smallest and the 6\textsuperscript{th} largest.
- 34 is the 8\textsuperscript{th} smallest and the 4\textsuperscript{th} largest.
**Notations**

- $S_i$ the set of all the keys that are **smaller** than $A[i]$: 

- $G_i$ the set of all the keys that are **greater** than $A[i]$: 

**Observation:** $A[i]$ is the $k$ smallest key iff 
  $$|S_i| = k - 1 \quad \text{AND} \quad |G_i| = n - k.$$

Example

\[ [21, 34, 8, 5, 55, 13, 1, 3, 89, 2, 123] \]

- \( n = 11, \ k = 4. \)
- The \( k \) smallest key is 5.
- \( S_i = \{1, 3, 2\} \Rightarrow |S_i| = k - 1 = 3. \)
- \( G_i = \{21, 34, 8, 55, 13, 89, 123\} \Rightarrow |G_i| = n - k = 7. \)
Possible Solutions to the $k$-Selection Problem

**Solution I:**
- **Algorithm:** Sort the array and find the $k$ smallest key.
- **Complexity:** $\Theta(n \log n)$ comparisons.

**Solution II:**
- **Algorithm:** Repeat finding the minimum key $k$ times.
- **Complexity:** $\Theta(kn)$ comparisons:
  $$(n - 1) + (n - 2) + \cdots + (n - k) = kn - \frac{k(k+1)}{2}.$$ 

**Which one is better?**
- **I** is better than **II** for $k = \omega(\log n)$.
- **II** is better than **I** for $k = o(\log n)$. 
Randomized Solution

- Select a **pivot** $p = A[i]$ for a random $i$ from the range $[1..n]$.

- **Partition** the array into 2 sets:
  - $S$ the set of all keys that are smaller than $p$.
  - $G$ the set of all keys that are greater than $p$.

- **Decision:**
  1. $|S| \geq k$: Recursively select the $k$ smallest in $S$.
  2. $(|S| = k - 1) \text{ AND } (|G| = n - k)$: Return $p$.
  3. $|G| \geq n + 1 - k$: Recursively select the $(k - |S| - 1)$ smallest in $G$. 
Example

- **Input:** $A = [21, 34, 8, 5, 55, 13, 1, 3, 89, 2, 123]$ and $k = 4$.
  - $p = 8$: $S = \{5, 1, 3, 2\}$ and $G = \{21, 34, 55, 13, 89, 123\}$.

- **Second instance:** $A = [5, 1, 3, 2]$ and $k = 4$.
  - $p = 2$: $S = \{1\}$ and $G = \{5, 3\}$.

- **Third instance:** $A = [5, 3]$ and $k = 2$.
  - $p = 5$: $S = \{3\}$ and $G = \emptyset$.

- **Output:** The $k = 4$ smallest key is 5.
**Observation:** The \( k \) smallest key is the \((n + 1 − k)\) largest key.

The sizes of \( S \) and \( G \) determine in which part of the array to look for the \( k \) smallest key.

1. The \( k \) smallest key is in \( S \).
2. The \( k \) smallest key is not in \( S \cup G \) \(\Rightarrow\) it is the pivot.
3. The \( k \) smallest key is in \( G \).
Randomized Solution – Expected Number of Comparisons

A good pivot: \((|S| \leq \frac{3n}{4}) \text{ AND } (|G| \leq \frac{3n}{4})\).

Probabilities facts:
- With probability 1/2 the random pivot is good.
- The expected number of random selections until a good pivot is found is 2.
Modified Randomized Solution

- Repeat selecting a pivot $p = A[i]$ for a random $i$ from the range $[1..n]$ until finding a good pivot.

- Partition the array into the 3 sets: $S$, $E$, $G$.

**Decision:**

1. $|S| \geq k$: Recursively select the $k$ smallest in $S$.
2. $(|S| < k) \text{ AND } (|G| < n + 1 - k)$: Return $p$.
3. $|G| \geq n + 1 - k$: Recursively select the $(k - |S| - |E|)$ smallest in $G$. 
Randomized Solution – Expected Number of Comparisons

- **Expected number of comparisons:** $T(n)$.
  - $\Theta(n)$ to perform one partition.
  - $\Theta(n)$ until a good partition is found.

- $T(n) \leq T(3n/4) + \Theta(n) = \Theta(n)$.
  - The expectation of a sum is the sum of expectations.
\[ T(n) = \Theta(n) \text{ – Direct Method} \]

- Assume a **nice** value for \( n \).
- Assume \( T(n) \leq T(3n/4) + \alpha n \) for constant \( \alpha > 0 \).

\[
egin{align*}
T(n) & \leq T(3n/4) + \alpha n \\
& \leq T(9n/16) + (3/4)\alpha n + \alpha n \\
& \leq T(27n/64) + (9/16)\alpha n + (3/4)\alpha n + \alpha n \\
& \vdots \\
& \leq \alpha n + (3/4)\alpha n + (9/16)\alpha n + \cdots + (3/4)^i \alpha n + \cdots \\
& < \alpha n \sum_{i=0}^{\infty} (3/4)^i < \alpha n \left( \frac{1}{1 - (3/4)} \right) = 4\alpha n .
\end{align*}
\]
$T(n) = \Theta(n)$ – Master Theorem

\[ T(n) = T\left(\frac{3n}{4}\right) + \Theta(n) \]

- $a = 1$.
- $b = \frac{4}{3}$.
- $\log_b(a) = 0$.
- $d = 1$.
- $d > \log_b(a)$.

\[ \Rightarrow \text{Case 3: } T(n) = \Theta(n^d) = \Theta(n). \]
Deterministic Solution to the $k$-Selection Problem

- Assume a **nice** value for $n$ and ignore ceilings and floors.

- Finding a pivot:
  - **Partition** the array into $n/5$ groups each with 5 keys.
  - **Find** the medians of each one of the $n/5$ groups.
  - **Find** the median of the $n/5$ medians recursively.
  - The **pivot** is the median of the medians.

- The rest of the **procedure** is as the randomized solution.
Deterministic Solution – Illustration

G={greater than pivot}

S={Smaller than pivot}

Pivot
Assume distinct keys and ignore floors and ceilings.

**Observations:**

- **S** contains the $n/10$ medians that are smaller than the pivot and the $2n/10$ keys that are smaller than these $n/10$ medians.
  \[ |S| \geq \frac{3n}{10} \Rightarrow |G| \leq \frac{7n}{10}. \]

- **G** contains the $n/10$ medians that are greater than the pivot and the $2n/10$ keys that are greater than these $n/10$ medians.
  \[ |G| \geq \frac{3n}{10} \Rightarrow |S| \leq \frac{7n}{10}. \]
\( \Theta(n) \) Worst Case Number of Comparisons

- **Worst case complexity:** \( T(n) \).
  - \( \Theta(n) \) to find the \( n/5 \) medians.
  - \( T(n/5) \) to find the median of the medians.
  - \( \Theta(n) \) to perform the partition.
  - At most \( T(7n/10) \) for the recursion.

- \( T(n) \leq T(7n/10) + T(n/5) + \Theta(n) = \Theta(n) \).
  - Because \( 7n/10 + n/5 = (1 - \varepsilon)n \) for a constant \( \varepsilon \).
Solving the Recursive Formula

**Formula:** \( T(n) \leq T(7n/10) + T(n/5) + \alpha n. \)
- For some constant \( \alpha \) that is independent of \( n \).

**Guess:** \( T(n) \leq \beta n. \)
- For some constant \( \beta \) that is independent of \( n \).

**Induction:** \( T(n) \leq \beta(7n/10) + \beta(n/5) + \alpha n \)
\[ = ((7\beta/10) + (\beta/5) + \alpha) n. \]

**Set:** \( \beta = 10\alpha \Rightarrow T(n) \leq ((7\beta/10) + (\beta/5) + (\beta/10)) n. \)

**Conclude:** \( T(n) \leq \beta n \leq 10\alpha n. \)
The Value of the Constants $\alpha$ and $\beta$

Finding all the $n/5$ medians:
- The median of 5 keys can be found with 6 comparisons.
- $6(n/5) = 1.2n$ comparisons to find all the medians.

$(2/5)n = 0.4n$ comparisons, only with the keys not in $S \cup G$, to perform the partition.

$\Rightarrow \alpha \leq 1.6$.

$\Rightarrow \beta \leq 10\alpha \leq 16$.

$\Rightarrow T(n) \leq \beta n \leq 16n$. 
Why Not Groups of 3?

- $S$ contains the $n/6$ medians that are smaller than the pivot and the $n/6$ keys that are smaller than these $n/6$ medians.
  \[ \Rightarrow |S| \geq n/3 \Rightarrow |G| \leq 2n/3. \]

- Similarly, $|S| \leq 2n/3$.

- At most $T(2n/3)$ for the recursion.

- $T(n/3)$ to find the median of the medians.

- Therefore, $T(n) \leq T(2n/3) + T(n/3) + \Theta(n)$.

- The solution to this recursive formula is $T(n) = \Theta(n \log n)$. 
Groups of $2k + 1$

- At most $T \left( \frac{(3k+1)n}{4k+2} \right)$ for the recursion.

- $T \left( \frac{n}{2k+1} \right)$ to find the median of the medians.

- $T(n) \leq T \left( \frac{(3k+1)n}{4k+2} \right) + T \left( \frac{n}{2k+1} \right) + \Theta(n) = \Theta(n)$.

Therefore, $T(n) \leq \beta_k n$ for a constant $\beta_k$ that depends on $k$ but independent on $n$.

The best $k$ is determined by the number of comparisons required to find all the $n/(2k + 1)$ medians and the number of comparisons needed to complete the partition.
The $k$-Selection Problem Complexity

- **Lower bound**: $\Omega(n)$ comparisons are required for selecting the minimum.

- **Randomized upper bound**: $\Theta(n)$.

- **Deterministic upper bound**: $\Theta(n)$.

- **Complexity**: $\Theta(n)$ average and worst case.
**The $k$-Selection Problem Known Bounds**

- **First linear upper bound:** $T(n) \leq 5.43n$.
- **Best upper bound:** $T(n) \leq 2.95n + o(n)$.
- **Simple lower bound:** $T(n) \geq 1.5n$.
- **Best lower bound:** $T(n) \geq 2n + o(n)$. 

Amotz Bar-Noy (CUNY)