

Analysis of Algorithms

Order Statistics

The General Search Problem

Input:

- ★ An **ordered** array A of n **keys**: $A[1] \leq A[2] \leq \dots \leq A[n]$.
- ★ 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.
- ★ $K \leq A[i]$? $K < A[i]$? $K = A[i]$?

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 $1 \leq i \leq n$.

Goal: Minimize number of comparisons until finding x .

- ★ In the worst 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 $[1..n]$:

$$\frac{1}{n} (1 + 2 + \dots + n) = \frac{1}{n} \cdot \frac{n(n + 1)}{2} = \frac{n + 1}{2} .$$

Binary Search

Binary-Search (n, x)

$l = 1$

$u = n$

while $l < u$

$m = \lfloor (u + l) / 2 \rfloor$

if $x \leq m$ (* comparison *)

then $u = m$

else $l = m + 1$

return l

Binary Search – Correctness

- ★ The search **terminates** since in each iteration the size of the search range, $(u - \ell)$, is **decreased**.
 - $u > \ell \Rightarrow \ell \leq m < m + 1 \leq u$.
 - Therefore, $m - \ell < u - \ell$ and $u - (m + 1) < u - \ell$.
- ★ By **induction**, always $\ell \leq x \leq u$.
- ★ At the end, $\ell = u$ and therefore $x = \ell$.

Binary Search – Complexity – $n = 2^k$

- ★ Induction claim: $u - \ell = 2^{k-h} - 1$ after h comparisons.
- ★ $h = 0 \Rightarrow u_h - \ell_h = n - 1 = 2^k - 1 = 2^{k-h} - 1$.
- ★ $h = k \Rightarrow u_k - \ell_k = 2^{k-k} - 1 = 0$.
- ★ Termination when $u = \ell \Rightarrow h = k = \log_2 n$ comparisons.

Binary Search – Complexity – $n = 2^k$

★ Induction step: $m_h = \lfloor (u_h + \ell_h)/2 \rfloor = \ell_h + \lfloor (u_h - \ell_h)/2 \rfloor$.

1. $u_{h+1} = m_h$ and $\ell_{h+1} = \ell_h$ imply:

$$\begin{aligned}u_{h+1} - \ell_{h+1} &= m_h - \ell_h \\ &= \lfloor (u_h - \ell_h)/2 \rfloor \\ &= \lfloor (2^{k-h} - 1)/2 \rfloor = 2^{k-(h+1)} - 1 .\end{aligned}$$

2. $\ell_{h+1} = m_h + 1$ and $u_{h+1} = u_h$ imply:

$$\begin{aligned}u_{h+1} - \ell_{h+1} &= u_h - m_h - 1 \\ &= (u_h - \ell_h) - \lfloor (u_h - \ell_h)/2 \rfloor - 1 \\ &= (2^{k-h} - 2) - (2^{k-(h+1)} - 1) = 2^{k-(h+1)} - 1 .\end{aligned}$$

Binary Search – Complexity – $2^{k-1} < n \leq 2^k$

Monotonicity lemma: For $n_1 < n_2$ and $1 \leq x \leq n_1$, the number of comparisons performed by **Binary-Search**(n_1, x) is less or equal to the number of comparisons performed by **Binary-Search**(n_2, x).

Assumption: $2^{k-1} < n \leq 2^k \Rightarrow k = \lceil \log_2 n \rceil$.

Corollary $\lceil \log_2 n \rceil$ is an upper bound on the number of **comparisons** in the worst case.

Remark: Could be $\lfloor \log_2 n \rfloor$ **comparisons** in some cases but **never** less than $\lfloor \log_2 n \rfloor$ **comparisons**.

Adversary Player I

- ★ **Does not** select x at the beginning of the game. Instead, maintains a set of **candidates** \mathcal{S} for x .
- ★ Given a search question:
 - $\mathcal{S}(Y)$ – the set of candidates if the answer is **YES**.
 - $\mathcal{S}(N)$ – the set of candidates if the answer is **NO**.
- ★ **Observation:** $\mathcal{S} = \mathcal{S}(Y) \cup \mathcal{S}(N)$.
- ★ The adversary answer rule:
 - **YES** if $|\mathcal{S}(Y)| \geq |\mathcal{S}(N)|$.
 - **NO** if $|\mathcal{S}(Y)| < |\mathcal{S}(N)|$.

Example

Input $n = 34$ ($* x \in [1..34] *$).

Search:

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

Output: $x = 21$.

Impossible to Search Faster than Binary Search

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 \mathcal{S}_i be the set of candidates after i comparisons.
- ★ In particular, $|\mathcal{S}_0| = n$ and $|\mathcal{S}_k| = 1$.
- ★ By the observation, $|\mathcal{S}_{i+1}|/|\mathcal{S}_i| \geq (1/2)$ 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?
 - $x \in \{1, 2, 3, 5, 8, 13, 21, 34, 55\}$?

Find the Minimum Or the Maximum

Input: An array A of n keys: $A[1], A[2], \dots, A[n]$.

Output:

Minimum: Key $A[1]$ such that $A[1] \leq A[i]$ for $1 \leq i \leq n$.

Maximum: Key $A[n]$ such that $A[n] \geq A[i]$ for $1 \leq i \leq n$.

Method: Ask comparisons between any 2 keys: $A[i] > A[j]$?

Goal: Minimize number of key comparisons.

Find the Minimum

```
Trivial-Find-Min( $A[1], \dots, A[n]$ )  
  for  $i = 2$  to  $n$   
    if  $A[1] > A[i]$  (* comparison *)  
      then  $A[1] \leftrightarrow A[i]$   
  return  $A[1]$ 
```

Correctness:

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

Complexity: Exactly $n - 1$ comparisons.

Adversary Strategy

Key idea: Any entry could be the minimum.

Data structure:

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

Answer rule:

- ★ $(R_1 : R_2) \Rightarrow$ A consistent answer.
- ★ $(B : R) \Rightarrow B < R$.
- ★ $(B_1 : B_2) \Rightarrow B_1 < B_2$; transfer B_2 from \mathcal{B} to \mathcal{R} .

There Is No Better Algorithm

Theorem: 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 \mathcal{B} .
- ★ Only $(B_1 : B_2)$ is a **useful** comparison.
- ★ Each **useful** comparison **decreases** the size of \mathcal{B} by 1.
- ★ $n - 1$ **useful** comparisons are required to **decrease** the size of \mathcal{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( $A[1], \dots, A[n]$ )  
  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( $A[1], \dots, A[n/2]$ )
```

Complexity

Number of comparisons:

- ★ $\frac{n}{2} + \frac{n}{4} + \dots + 1 = n - 1$.
- ★ 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**.
- ★ In **Trivial-Find-Min** there are $n - 1$ rounds.

Lower Bound on Number of 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 $\lfloor |\mathcal{B}|/2 \rfloor$ **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 \mathcal{B} from n to 1 by **halving**.

Find the Minimum & the Maximum for $n = 2^k$ – Solution I

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

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

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

Complexity:

★ $(n - 1) + (n - 2) = 2n - 3$ comparisons.

★ $2 \log_2 n$ rounds.

Find the Minimum & the Maximum for $n = 2^k$ – Solution II

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

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($A[1], \dots, A[n/2]$)

Parallel-Find-Max($A[n/2 + 1], \dots, A[n]$)

Complexity:

★ $\frac{n}{2} + 2 \left(\frac{n}{2} - 1 \right) = \frac{3n}{2} - 2$ 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:** $\mathcal{N} = \{A[1], \dots, A[n]\}$ 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 $N \rightarrow \mathcal{B}.$
- ★ $(B : N) \Rightarrow B < N$ and $N \rightarrow \mathcal{H}.$
- ★ $(N : H) \Rightarrow N < H$ and $N \rightarrow \mathcal{B}.$
- ★ $(N_1 : N_2) \Rightarrow N_1 < N_2$ and $N_1 \rightarrow \mathcal{B}$ and $N_2 \rightarrow \mathcal{H}.$
- ★ $(B : H) \Rightarrow B < H.$
- ★ $(B_1 : B_2) \Rightarrow B_1 < B_2$ and $B_2 \rightarrow \mathcal{R}.$
- ★ $(H_1 : H_2) \Rightarrow H_1 < H_2$ and $H_1 \rightarrow \mathcal{R}.$

There Is No Better Algorithm

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

Proof:

- ★ Non-max and non-min keys: $\mathcal{N} \rightarrow \{\mathcal{B}, \mathcal{H}\} \rightarrow \mathcal{R}$.
- ★ $(N_1 : N_2)$, $(B_1 : B_2)$, 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 $\lceil \frac{n}{2} \rceil$ **comparisons**.
- ★ The **fastest** way to leave one key in both \mathcal{B} and \mathcal{H} requires at least $n - 2$ **comparisons**.

Parallel Find Minimum Or Maximum for Any n – Solution I

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

- ★ Compare only $\lfloor n/2 \rfloor$ pairs.
- ★ Continue recursively with $\lceil n/2 \rceil$ candidates:
 - Even n : the $n/2$ smaller keys.
 - Odd n : the $\lfloor n/2 \rfloor$ smaller keys and the un-compared key.

Number of rounds: $R(n) = \lceil \log_2 n \rceil$.

- ★ $\lceil \log_2 n \rceil$ rounds are required to decrease n to 1 by halving.

Number of comparisons: $C(n) = n - 1$.

- ★ There are only useful comparisons.

Parallel Find Minimum Or Maximum for Any n – Solution II

Assumption: $2^{k-1} < n < 2^k$.

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

- ★ **Compare** only $n - 2^{k-1}$ pairs.
- ★ **Continue recursively** with the 2^{k-1} candidates:
 - Smaller keys: $n - 2^{k-1}$.
 - Un-compared keys: $n - 2(n - 2^{k-1}) = 2^k - n$.

Number of rounds: $R(n) = 1 + \log_2(2^{k-1}) = k = \lceil \log_2 n \rceil$.

Number of comparisons: $C(n) = n - 1$.

- ★ There are only **useful** comparisons.

Parallel Find the Minimum & the Maximum for Any n

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

- ★ **Compare** only $\lfloor n/2 \rfloor$ pairs.
- ★ **Find** the maximum among the $\lceil n/2 \rceil$ candidates:
 - Even n : the $n/2$ larger keys.
 - Odd n : the $\lfloor n/2 \rfloor$ larger keys and the un-compared key.
- ★ **Find** the minimum among the $\lceil n/2 \rceil$ candidates:
 - Even n : the $n/2$ smaller keys.
 - Odd n : the $\lfloor n/2 \rfloor$ smaller keys and the un-compared key.

Remark: For odd n , after the first round, one key is a candidate to be the maximum **and** the minimum.

Implementation

- ★ The **comparisons**:
 - $A[1]$ with $A[n]$
 - $A[2]$ with $A[n - 1]$
 - \vdots
 - $A[i]$ with $A[n + 1 - i]$
 - \vdots
 - $A[\lfloor n/2 \rfloor]$ with $A[n + 1 - \lfloor n/2 \rfloor]$.
- ★ **Exchange** the keys **if** $A[i] > A[n + 1 - i]$.
- ★ Even n : All the keys are **compared**.
- ★ Odd n : $A[\lceil n/2 \rceil]$ is **un-compared** since $\lfloor n/2 \rfloor < \lceil n/2 \rceil < n + 1 - \lfloor n/2 \rfloor$.

Implementation

- ★ The candidates for **minimum** are:
 - Even n : $A[1], \dots, A[n/2]$.
 - Odd n : $A[1], \dots, A[\lceil n/2 \rceil]$.
- ★ The candidates for **maximum** are:
 - Even n : $A[n/2 + 1], \dots, A[n]$.
 - Odd n : $A[\lceil n/2 \rceil], \dots, A[n]$.

Complexity

Number of rounds: Even n :

- ★ $RR(n) = \lceil \log_2 n \rceil$ – optimal.
- ★ The same as **Parallel-Find-Min**.

Number of rounds: Odd n :

- ★ $RR(n) = 1 + \lceil \log_2 n \rceil$ – almost optimal.
- ★ An **additional** round for the first round un-compared key.

Number of comparisons:

- ★
$$\begin{aligned} CC(n) &= 2C(\lceil n/2 \rceil) + \lfloor n/2 \rfloor = 2 \cdot \lceil n/2 \rceil - 2 + \lfloor n/2 \rfloor \\ &= \lceil 3n/2 \rceil - 2 - \text{Optimal} \end{aligned}$$

Find the First and the Second

Input: An array A of n keys: $A[1], A[2], \dots, A[n]$.

Output: Keys $A[n]$ and $A[n - 1]$:

★ $A[n] \geq A[n - 1]$.

★ $A[n - 1] \geq A[i]$ 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( $A[1], \dots, A[n]$ )  
  Trivial-Find-Max( $A[1], \dots, A[n]$ )  
  Trivial-Find-Max( $A[1], \dots, A[n - 1]$ )  
  return ( $A[n] \geq A[n - 1]$ )
```

Correctness: By definition.

Complexity: Exactly $(n - 1) + (n - 2) = 2n - 3$ comparisons.

Find the First and the Second – A Better Solution

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 array A of n keys, $A[1], A[2], \dots, A[n]$.
- ★ 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 .

Example: Distinct Keys

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

- ★ 5 is the 4th smallest and the 8th largest.
- ★ 13 is the **median**: the 6th smallest and the 6th largest.
- ★ 34 is the 8th smallest and the 4th largest.

Example: Arbitrary Keys

[5, 3, 5, 5, 2, 5, 1, 3, 5, 2, 3]

- ★ 3 is the 4th smallest and the 8th largest.
- ★ 3 is the **median**: the 6th smallest and the 6th largest.
- ★ 5 is the 8th smallest and the 4th largest.

Notations and Observations

- ★ S_i the set of all the keys that are **smaller** than $A[i]$:
 - $S_i = \{A[j] \mid A[j] < A[i]\}$.
- ★ G_i the set of all the keys that are **greater** than $A[i]$:
 - $G_i = \{A[j] \mid A[j] > A[i]\}$.
- ★ **Distinct keys:** $A[i]$ is the k smallest key if
 - $(|S_i| = k - 1)$ **AND** $(|G_i| = n - k)$.
- ★ **Arbitrary keys:** $A[i]$ is the k smallest key if
 - $(|S_i| \leq k - 1)$ **AND** $(|G_i| \leq n - k)$.

Examples

Distinct keys: $n = 11$, $k = 4$, $[21, 34, 8, 5, 55, 13, 1, 3, 89, 2, 123]$.

- ★ The k **smallest** key is 5.
- ★ $S_i = \{1, 3, 2\} \Rightarrow |S_i| = k - 1 = 3$.
- ★ $G_i = \{21, 34, 8, 55, 13, 89, 123\} \Rightarrow |G_i| = n - k = 7$.

Arbitrary keys: $n = 11$, $k = 5$, $A = [5, 3, 5, 5, 2, 5, 1, 3, 5, 2, 3]$.

- ★ The k **smallest** key is 3.
- ★ $S_i = \{2, 1, 2\} \Rightarrow |S_i| = 3 \leq k - 1$.
- ★ $G_i = \{5, 5, 5, 5, 5\} \Rightarrow |G_i| = 5 \leq n - k$.

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:** Find the minimum key k times.
- ★ **Complexity:** $\Theta(kn)$ comparisons:
 - $(n - 1) + (n - 2) + \dots + (n - k) = kn - \frac{k(k+1)}{2}$.

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 for Distinct Numbers

- ★ **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 .
- (1)** $|S| \geq k$: Recursively **select** the k smallest in S .
- (2)** $(|S| = k - 1)$ **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.

Randomized Solution for Arbitrary Keys

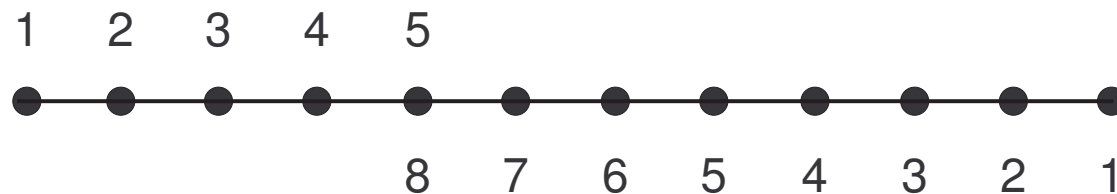
- ★ **Select** a **pivot** $p = A[i]$ for a **random** i from the range $[1..n]$.
- ★ **Partition** the array into 3 sets:
 - S the set of all keys that are **smaller** than p .
 - E the set of all keys that are **equal** to p .
 - G the set of all keys that are **greater** than p .
- (1)** $|S| \geq k$: Recursively **select** the k smallest in S .
- (2)** $(|S| < k)$ **AND** $(|G| < n + 1 - k)$: **Return** p .
- (3)** $|G| \geq n + 1 - k$: Recursively **select** the $(k - |S| - |E|)$ smallest in G .

Example

- ★ **Input:** $A = [5, 3, 5, 5, 2, 5, 1, 3, 5, 2, 3]$ and $k = 3$.
- ★ $p = 3$: $S = \{2, 1, 2\}$ and $G = \{5, 5, 5, 5, 5\}$.
- ★ **Second instance:** $A = [2, 1, 2]$ and $k = 3$.
- ★ $p = 2$: $S = \{1\}$ and $G = \emptyset$.
- ★ **Output:** The $k = 3$ smallest key is 2 .

Randomized Solution – Correctness

- ★ 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: $(|S| \leq \frac{3n}{4})$ **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.

Revised 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 .
 - (1) $|S| \geq k$: Recursively select the k smallest in S .
 - (2) $(|S| < k)$ 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**.

$T(n) = \Theta(n)$ – Direct Method

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

$$\begin{aligned} T(n) &\leq T(3n/4) + \alpha n \\ &\leq T(9n/16) + (3/4)\alpha n + \alpha n \\ &\leq T(27n/64) + (9/16)\alpha n + (3/4)\alpha n + \alpha n \\ &\vdots \\ &\leq \alpha n + (3/4)\alpha n + (9/16)\alpha n + \dots + (3/4)^i \alpha n + \dots \\ &< \alpha n \sum_{i=0}^{\infty} (3/4)^i < \alpha n \left(\frac{1}{1 - (3/4)} \right) = 4\alpha n . \end{aligned}$$

$T(n) = \Theta(n)$ – Master Theorem

$$T(n) = T(3n/4) + \Theta(n)$$

★ $a = 1$.

★ $b = 4/3$.

★ $\log_b(a) = 0$.

★ $d = 1$.

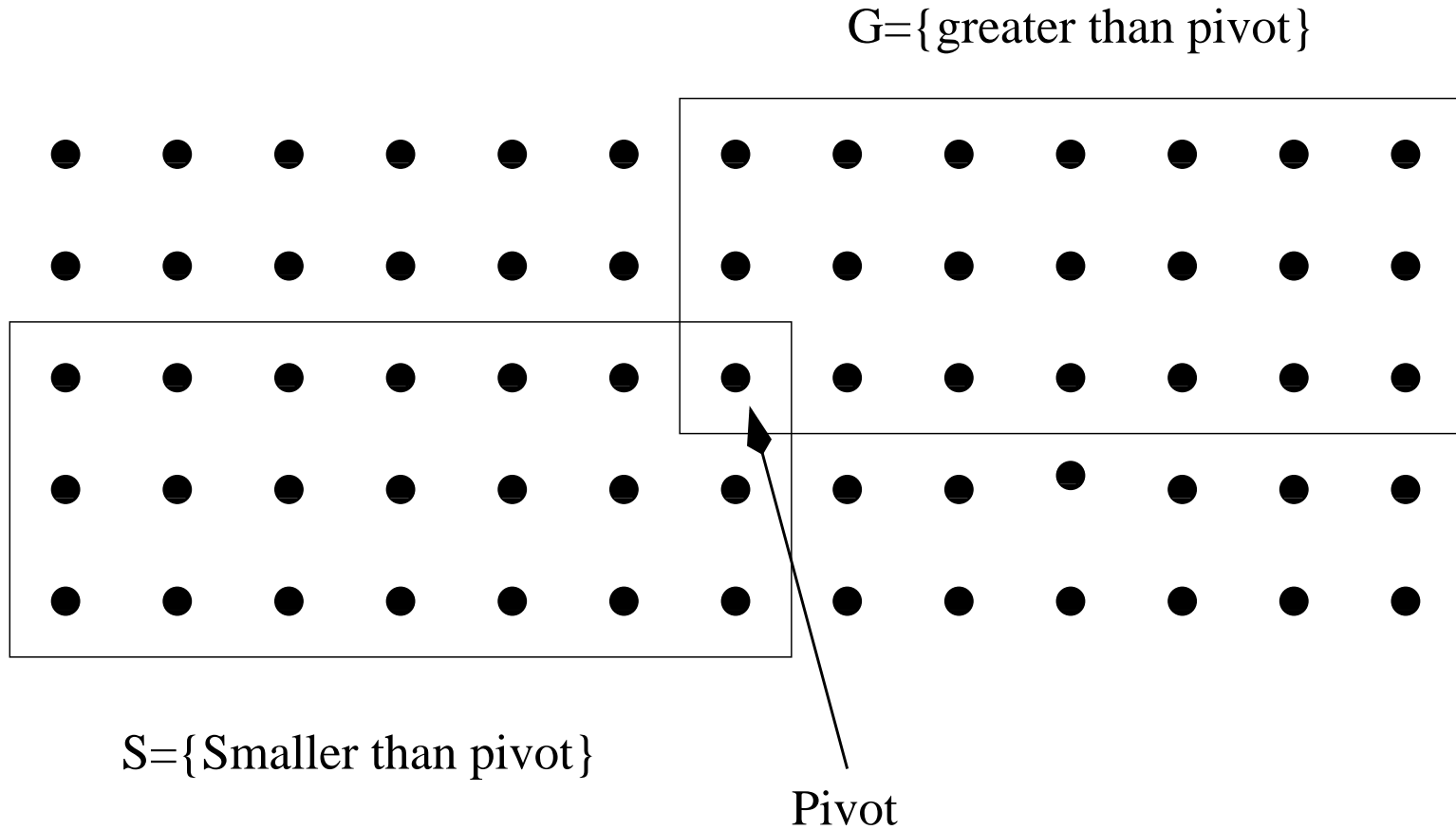
★ $d > \log_b(a)$.

⇒ 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



Deterministic Solution – Worst Case Number of Comparisons

- ★ 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.

$$\Rightarrow |S| \geq 3n/10 \Rightarrow |G| \leq 7n/10.$$

- ★ 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.

$$\Rightarrow |G| \geq 3n/10 \Rightarrow |S| \leq 7n/10.$$

$\Theta(n)$ 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** ε .

Solving the Recursive Formula

- ★ **Formula:** $T(n) \leq T(7n/10) + T(n/5) + \alpha n$.
 - For some constant α that is independent of n .
- ★ **Guess:** $T(n) \leq \beta n$.
 - For some constant β 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 α and β

- ★ 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 SUG , to perform the partition.
- ⇒ $\alpha \leq 1.6$.
- ⇒ $\beta \leq 10\alpha \leq 16$.
- ⇒ $T(n) \leq \beta n \leq 16n$.

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 β_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

Best upper bound: $T(n) \leq 2.95n$.

Simple lower bound: $T(n) \geq 3n/2 - 3/2$.

Best lower bound:

- $T(n) \geq 1.8n$ for any n .
- $T(n) > 2n$ for large enough n .