

4. Searching

Linear Search, Binary Search, (Interpolation Search,) Lower Bounds
[Ottman/Widmayer, Kap. 3.2, Cormen et al, Kap. 2: Problems
2.1-3,2.2-3,2.3-5]

The Search Problem

Provided

- A set of data sets

examples

telephone book, dictionary, symbol table

- Each dataset has a key k .
- Keys are comparable: unique answer to the question $k_1 \leq k_2$ for keys k_1, k_2 .

Task: find data set by key k .

Search in Array

Provided

- Array A with n elements ($A[1], \dots, A[n]$).
- Key b

Wanted: index k , $1 \leq k \leq n$ with $A[k] = b$ or "not found".

22	20	32	10	35	24	42	38	28	41
1	2	3	4	5	6	7	8	9	10

Linear Search

Traverse the array from $A[1]$ to $A[n]$.

Linear Search

Traverse the array from $A[1]$ to $A[n]$.

- *Best case*: 1 comparison.

Linear Search

Traverse the array from $A[1]$ to $A[n]$.

- *Best case*: 1 comparison.
- *Worst case*: n comparisons.

Linear Search

Traverse the array from $A[1]$ to $A[n]$.

- *Best case*: 1 comparison.
- *Worst case*: n comparisons.
- Assumption: each permutation of the n keys with same probability.
Expected number of comparisons for the successful search:

Linear Search

Traverse the array from $A[1]$ to $A[n]$.

- *Best case*: 1 comparison.
- *Worst case*: n comparisons.
- Assumption: each permutation of the n keys with same probability.
Expected number of comparisons for the successful search:

Linear Search

Traverse the array from $A[1]$ to $A[n]$.

- *Best case*: 1 comparison.
- *Worst case*: n comparisons.
- Assumption: each permutation of the n keys with same probability.
Expected number of comparisons for the successful search:

$$\frac{1}{n} \sum_{i=1}^n i = \frac{n+1}{2}.$$

Search in a Sorted Array

Provided

- Sorted array A with n elements ($A[1], \dots, A[n]$) with $A[1] \leq A[2] \leq \dots \leq A[n]$.
- Key b

Wanted: index k , $1 \leq k \leq n$ with $A[k] = b$ or "not found".

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

Divide and Conquer!

Search $b = 23$.

Divide and Conquer!

Search $b = 23$.

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

Divide and Conquer!

Search $b = 23$.

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b < 28$

Divide and Conquer!

Search $b = 23$.

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b < 28$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b > 20$

Divide and Conquer!

Search $b = 23$.

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b < 28$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

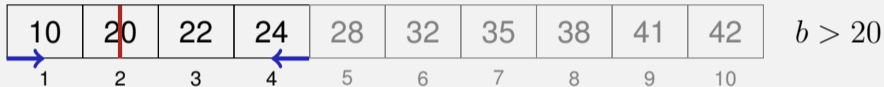
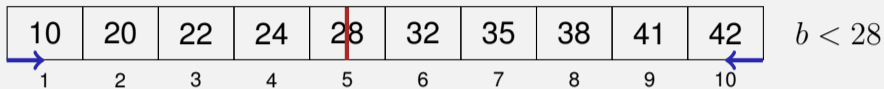
$b > 20$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b > 22$

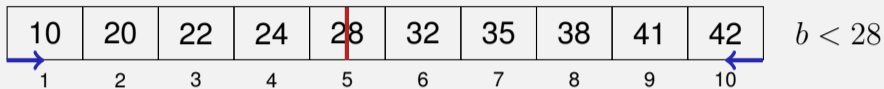
Divide and Conquer!

Search $b = 23$.



Divide and Conquer!

Search $b = 23$.



Binary Search Algorithm

BSearch($A[l..r], b$)

Input: Sorted array A of n keys. Key b . Bounds $1 \leq l \leq r \leq n$ or $l > r$ beliebig.

Output: Index of the found element. 0, if not found.

$m \leftarrow \lfloor (l + r) / 2 \rfloor$

if $l > r$ **then** // Unsuccessful search

return *NotFound*

else if $b = A[m]$ **then** // found

return m

else if $b < A[m]$ **then** // element to the left

return BSearch($A[l..m - 1], b$)

else // $b > A[m]$: element to the right

return BSearch($A[m + 1..r], b$)

Analysis (worst case)

Recurrence ($n = 2^k$)

$$T(n) = \begin{cases} d & \text{falls } n = 1, \\ T(n/2) + c & \text{falls } n > 1. \end{cases}$$

Compute:

$$T(n) = T\left(\frac{n}{2}\right) + c$$

Analysis (worst case)

Recurrence ($n = 2^k$)

$$T(n) = \begin{cases} d & \text{falls } n = 1, \\ T(n/2) + c & \text{falls } n > 1. \end{cases}$$

Compute:

$$T(n) = T\left(\frac{n}{2}\right) + c = T\left(\frac{n}{4}\right) + 2c$$

Analysis (worst case)

Recurrence ($n = 2^k$)

$$T(n) = \begin{cases} d & \text{falls } n = 1, \\ T(n/2) + c & \text{falls } n > 1. \end{cases}$$

Compute:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c = T\left(\frac{n}{4}\right) + 2c = \dots \\ &= T\left(\frac{n}{2^i}\right) + i \cdot c \end{aligned}$$

Analysis (worst case)

Recurrence ($n = 2^k$)

$$T(n) = \begin{cases} d & \text{falls } n = 1, \\ T(n/2) + c & \text{falls } n > 1. \end{cases}$$

Compute:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c = T\left(\frac{n}{4}\right) + 2c = \dots \\ &= T\left(\frac{n}{2^i}\right) + i \cdot c \\ &= T\left(\frac{n}{n}\right) + \log_2 n \cdot c = d + c \cdot \log_2 n \in \Theta(\log n) \end{aligned}$$

Analysis (worst case)

$$T(n) = \begin{cases} d & \text{if } n = 1, \\ T(n/2) + c & \text{if } n > 1. \end{cases}$$

Guess : $T(n) = d + c \cdot \log_2 n$

Proof by induction:

Analysis (worst case)

$$T(n) = \begin{cases} d & \text{if } n = 1, \\ T(n/2) + c & \text{if } n > 1. \end{cases}$$

Guess : $T(n) = d + c \cdot \log_2 n$

Proof by induction:

- Base clause: $T(1) = d$.

Analysis (worst case)

$$T(n) = \begin{cases} d & \text{if } n = 1, \\ T(n/2) + c & \text{if } n > 1. \end{cases}$$

Guess : $T(n) = d + c \cdot \log_2 n$

Proof by induction:

- Base clause: $T(1) = d$.
- Hypothesis: $T(n/2) = d + c \cdot \log_2 n/2$

Analysis (worst case)

$$T(n) = \begin{cases} d & \text{if } n = 1, \\ T(n/2) + c & \text{if } n > 1. \end{cases}$$

Guess : $T(n) = d + c \cdot \log_2 n$

Proof by induction:

- Base clause: $T(1) = d$.
- Hypothesis: $T(n/2) = d + c \cdot \log_2 n/2$
- Step: $(n/2 \rightarrow n)$

$$T(n) = T(n/2) + c = d + c \cdot (\log_2 n - 1) + c = d + c \log_2 n.$$

Result

Theorem

The binary sorted search algorithm requires $\Theta(\log n)$ fundamental operations.

Iterative Binary Search Algorithm

Input: Sorted array A of n keys. Key b .

Output: Index of the found element. 0, if unsuccessful.

$l \leftarrow 1; r \leftarrow n$

while $l \leq r$ **do**

$m \leftarrow \lfloor (l + r)/2 \rfloor$

if $A[m] = b$ **then**

return m

else if $A[m] < b$ **then**

$l \leftarrow m + 1$

else

$r \leftarrow m - 1$

return *NotFound*;

Correctness

Algorithm terminates only if A is empty or b is found.

Invariant: If b is in A then b is in domain $A[l..r]$

Proof by induction

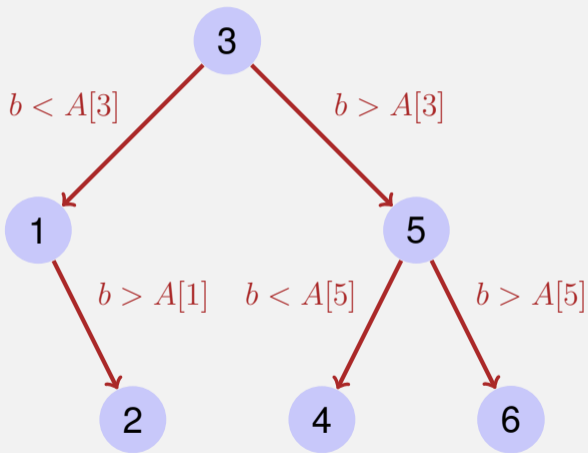
- Base clause $b \in A[1..n]$ (oder nicht)
- Hypothesis: invariant holds after i steps.
- Step:
 - $b < A[m] \Rightarrow b \in A[l..m - 1]$
 - $b > A[m] \Rightarrow b \in A[m + 1..r]$

Lower Bounds

Binary Search (worst case): $\Theta(\log n)$ comparisons.

Does for *any* search algorithm in a sorted array (worst case) hold that number comparisons = $\Omega(\log n)$?

Decision tree



- For any input $b = A[i]$ the algorithm must succeed \Rightarrow decision tree comprises at least n nodes.
- Number comparisons in worst case = height of the tree = maximum number nodes from root to leaf.

Decision Tree

Binary tree with height h has at most
 $2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1 < 2^h$ nodes.

Decision Tree

Binary tree with height h has at most
 $2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1 < 2^h$ nodes.

$$2^h > n \Rightarrow h > \log_2 n$$

Decision tree with n node has at least height $\log_2 n$.

Decision Tree

Binary tree with height h has at most $2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1 < 2^h$ nodes.

$$2^h > n \Rightarrow h > \log_2 n$$

Decision tree with n nodes has at least height $\log_2 n$.

Number decisions = $\Omega(\log n)$.

Theorem

Any comparison-based search algorithm on sorted data with length n requires in the worst case $\Omega(\log n)$ comparisons.

Lower bound for Search in Unsorted Array

Theorem

*Any comparison-based search algorithm with **un**sorted data of length n requires in the worst case $\Omega(n)$ comparisons.*

Attempt

② Correct?

"Proof": to find b in A , b must be compared with each of the n elements $A[i]$ ($1 \leq i \leq n$).

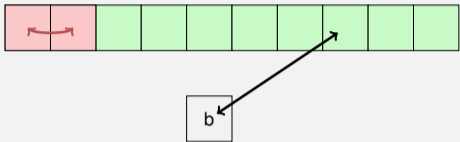
Attempt

❓ Correct?

"Proof": to find b in A , b must be compared with each of the n elements $A[i]$ ($1 \leq i \leq n$).

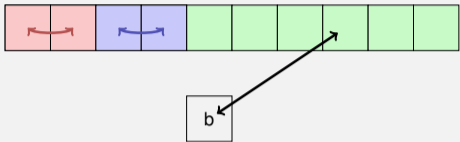
❗ Wrong argument! It is still possible to compare elements within A .

Better Argument



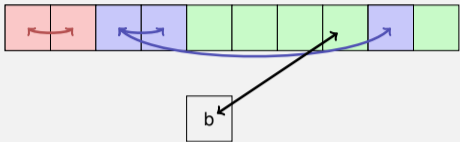
- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.

Better Argument



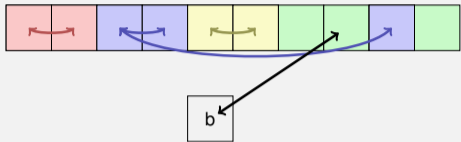
- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.
- To connect two groups at least one comparison is needed:
 $n - g \leq i$.

Better Argument



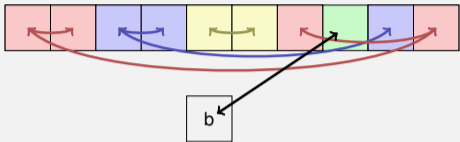
- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.
- To connect two groups at least one comparison is needed:
 $n - g \leq i$.

Better Argument



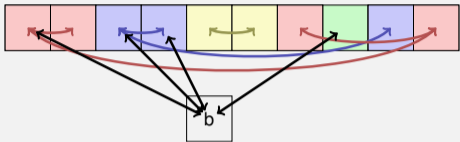
- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.
- To connect two groups at least one comparison is needed:
 $n - g \leq i$.

Better Argument



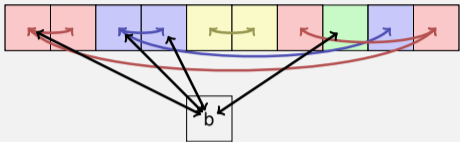
- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.
- To connect two groups at least one comparison is needed:
 $n - g \leq i$.

Better Argument



- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.
- To connect two groups at least one comparison is needed:
 $n - g \leq i$.
- At least one element per group must be compared with b .

Better Argument



- Different comparisons: Number comparisons with b : e Number comparisons without b : i
- Comparisons induce g groups. Initially $g = n$.
- To connect two groups at least one comparison is needed:
 $n - g \leq i$.
- At least one element per group must be compared with b .
- Number comparisons $i + e \geq n - g + g = n$. ■

5. Selection

The Selection Problem, Randomised Selection, Linear Worst-Case Selection [Ottman/Widmayer, Kap. 3.1, Cormen et al, Kap. 9]

The Problem of Selection

Input

- unsorted array $A = (A_1, \dots, A_n)$ with pairwise different values
- Number $1 \leq k \leq n$.

Output $A[i]$ with $|\{j : A[j] < A[i]\}| = k - 1$

Special cases

$k = 1$: Minimum: Algorithm with n comparison operations trivial.

$k = n$: Maximum: Algorithm with n comparison operations trivial.

$k = \lfloor n/2 \rfloor$: Median.

Naive Algorithm

Naive Algorithm

Repeatedly find and remove the minimum $\Theta(k \cdot n)$.
→ Median in $\Theta(n^2)$

Better Approaches

Better Approaches

- Sorting (covered soon): $\Theta(n \log n)$

Better Approaches

- Sorting (covered soon): $\Theta(n \log n)$
- Use a pivot: $\Theta(n)$!

Use a pivot



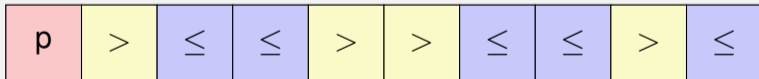
Use a pivot

- 1 Choose a (an arbitrary) *pivot* p



Use a pivot

- 1 Choose a (an arbitrary) *pivot* p
- 2 Partition A in two parts, and determine the rank of p by counting the indices i with $A[i] \leq p$.



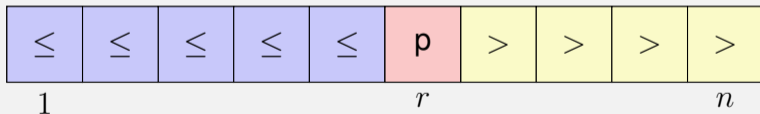
Use a pivot

- 1 Choose a (an arbitrary) *pivot* p
- 2 Partition A in two parts, and determine the rank of p by counting the indices i with $A[i] \leq p$.



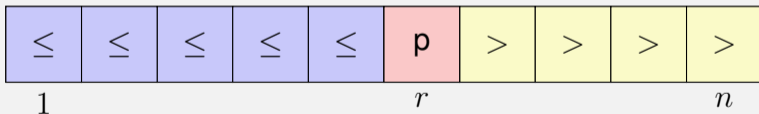
Use a pivot

- 1 Choose a (an arbitrary) *pivot* p
- 2 Partition A in two parts, and determine the rank of p by counting the indices i with $A[i] \leq p$.



Use a pivot

- 1 Choose a (an arbitrary) *pivot* p
- 2 Partition A in two parts, and determine the rank of p by counting the indices i with $A[i] \leq p$.
- 3 Recursion on the relevant part. If $k = r$ then found.



Algorithmus Partition($A[l..r], p$)

Input: Array A , that contains the pivot p in the interval $[l, r]$ at least once.

Output: Array A partitioned in $[l..r]$ around p . Returns position of p .

while $l \leq r$ **do**

while $A[l] < p$ **do**

$l \leftarrow l + 1$

while $A[r] > p$ **do**

$r \leftarrow r - 1$

 swap($A[l], A[r]$)

if $A[l] = A[r]$ **then**

$l \leftarrow l + 1$

return $l-1$

Correctness: Invariant

Invariant I : $A_i \leq p \forall i \in [0, l), A_i \geq p \forall i \in (r, n], \exists k \in [l, r] : A_k = p$.

while $l \leq r$ **do**

while $A[l] < p$ **do**

$l \leftarrow l + 1$

while $A[r] > p$ **do**

$r \leftarrow r - 1$

$\text{swap}(A[l], A[r])$

if $A[l] = A[r]$ **then**

$l \leftarrow l + 1$

return $l-1$

I

I und $A[l] \geq p$

I und $A[r] \leq p$

I und $A[l] \leq p \leq A[r]$

I

Correctness: progress

```
while  $l \leq r$  do  
  while  $A[l] < p$  do      progress if  $A[l] < p$   
     $l \leftarrow l + 1$   
  while  $A[r] > p$  do      progress if  $A[r] > p$   
     $r \leftarrow r - 1$   
  swap( $A[l], A[r]$ )          progress if  $A[l] > p$  oder  $A[r] < p$   
  if  $A[l] = A[r]$  then      progress if  $A[l] = A[r] = p$   
     $l \leftarrow l + 1$   
return  $l-1$ 
```

Choice of the pivot.

The minimum is a bad pivot: worst case $\Theta(n^2)$



Choice of the pivot.

The minimum is a bad pivot: worst case $\Theta(?)$



Choice of the pivot.

The minimum is a bad pivot: worst case $\Theta(?)$



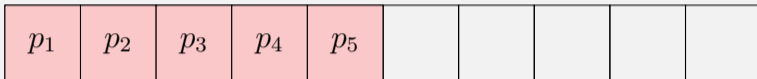
Choice of the pivot.

The minimum is a bad pivot: worst case $\Theta(?)$



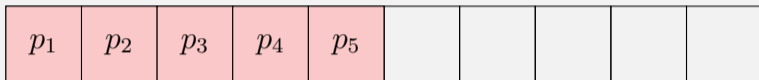
Choice of the pivot.

The minimum is a bad pivot: worst case $\Theta(n^2)$

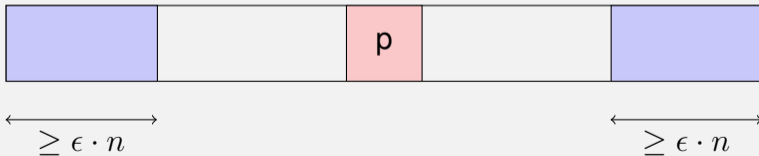


Choice of the pivot.

The minimum is a bad pivot: worst case $\Theta(n^2)$



A good pivot has a linear number of elements on both sides.



Analysis

Partitioning with factor q ($0 < q < 1$): two groups with $q \cdot n$ and $(1 - q) \cdot n$ elements (without loss of generality $g \geq 1 - q$).

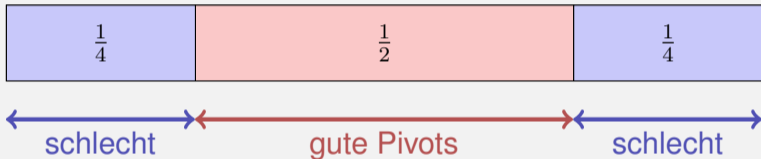
$$T(n) \leq T(q \cdot n) + c \cdot n$$

$$= c \cdot n + q \cdot c \cdot n + T(q^2 \cdot n) = \dots = c \cdot n \sum_{i=0}^{\log_q(n)-1} q^i + T(1)$$

$$\leq c \cdot n \underbrace{\sum_{i=0}^{\infty} q^i}_{\text{geom. Reihe}} + d = c \cdot n \cdot \frac{1}{1 - q} + d = \mathcal{O}(n)$$

How can we achieve this?

Randomness to our rescue (Tony Hoare, 1961). In each step choose a random pivot.



Probability for a good pivot in one trial: $\frac{1}{2} =: \rho$.

Probability for a good pivot after k trials: $(1 - \rho)^{k-1} \cdot \rho$.

Expected number of trials: $1/\rho = 2$ (Expected value of the geometric distribution:)

Algorithm Quickselect ($A[l..r], k$)

Input: Array A with length n . Indices $1 \leq l \leq k \leq r \leq n$, such that for all $x \in A[l..r] : |\{j | A[j] \leq x\}| \geq l$ and $|\{j | A[j] \leq x\}| \leq r$.

Output: Value $x \in A[l..r]$ with $|\{j | A[j] \leq x\}| \geq k$ and $|\{j | x \leq A[j]\}| \geq n - k + 1$

if $l=r$ **then**

 | return $A[l]$;

$x \leftarrow \text{RandomPivot}(A[l..r])$

$m \leftarrow \text{Partition}(A[l..r], x)$

if $k < m$ **then**

 | return QuickSelect($A[l..m - 1], k$)

else if $k > m$ **then**

 | return QuickSelect($A[m + 1..r], k$)

else

 | **return** $A[k]$

Algorithm RandomPivot ($A[l..r]$)

Input: Array A with length n . Indices $1 \leq l \leq i \leq r \leq n$

Output: Random “good” pivot $x \in A[l..r]$

repeat

 choose a random pivot $x \in A[l..r]$

$p \leftarrow l$

for $j = l$ **to** r **do**

if $A[j] \leq x$ **then** $p \leftarrow p + 1$

until $\lfloor \frac{3l+r}{4} \rfloor \leq p \leq \lceil \frac{l+3r}{4} \rceil$

return x

This algorithm is only of theoretical interest and delivers a good pivot in 2 expected iterations. Practically, in algorithm QuickSelect a uniformly chosen random pivot can be chosen or a deterministic one such as the median of three elements.

Median of medians

Goal: find an algorithm that even in worst case requires only linearly many steps.

Algorithm Select (k -smallest)

- Consider groups of five elements.
- Compute the median of each group (straightforward)
- Apply Select recursively on the group medians.
- Partition the array around the found median of medians. Result: i
- If $i = k$ then result. Otherwise: select recursively on the proper side.

Median of medians



Median of medians



1 groups of five

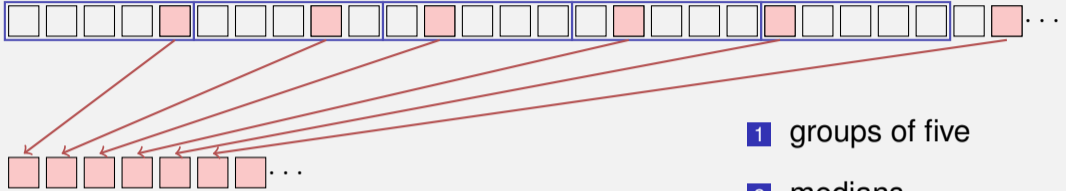
Median of medians



1 groups of five

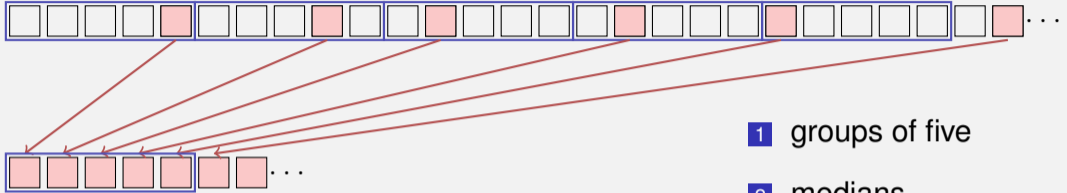
2 medians

Median of medians

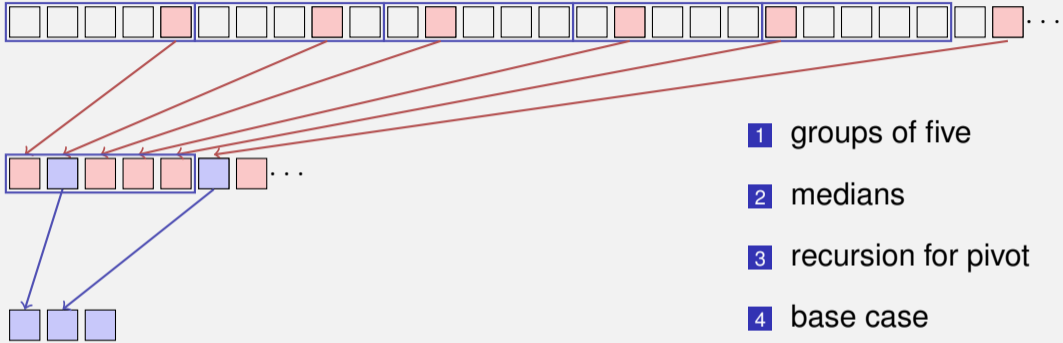


- 1 groups of five
- 2 medians
- 3 recursion for pivot

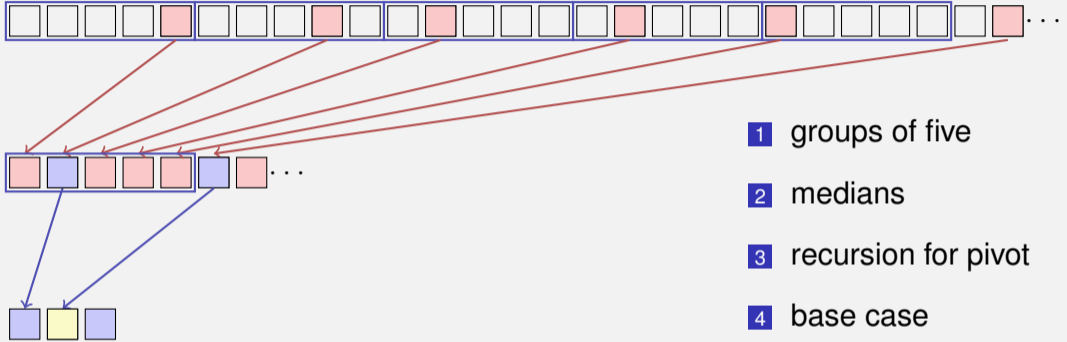
Median of medians



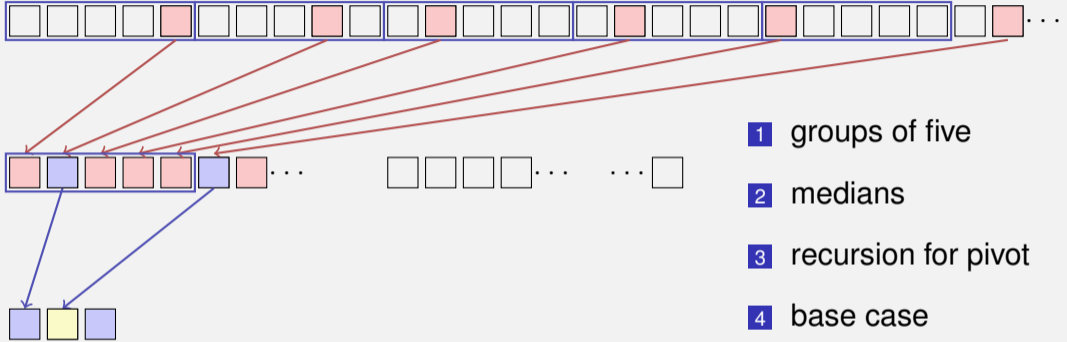
Median of medians



Median of medians

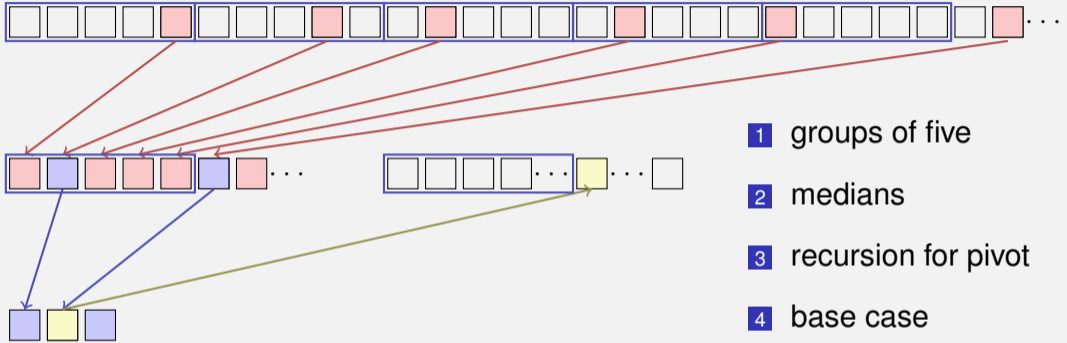


Median of medians



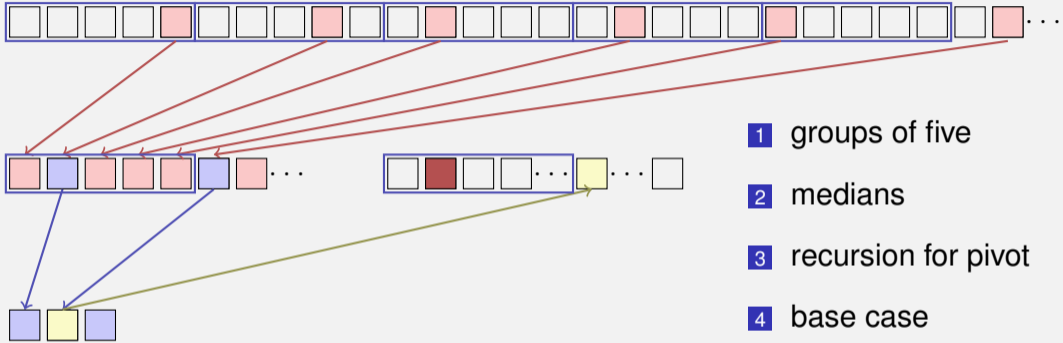
- 1 groups of five
- 2 medians
- 3 recursion for pivot
- 4 base case
- 5 pivot (level 1)

Median of medians



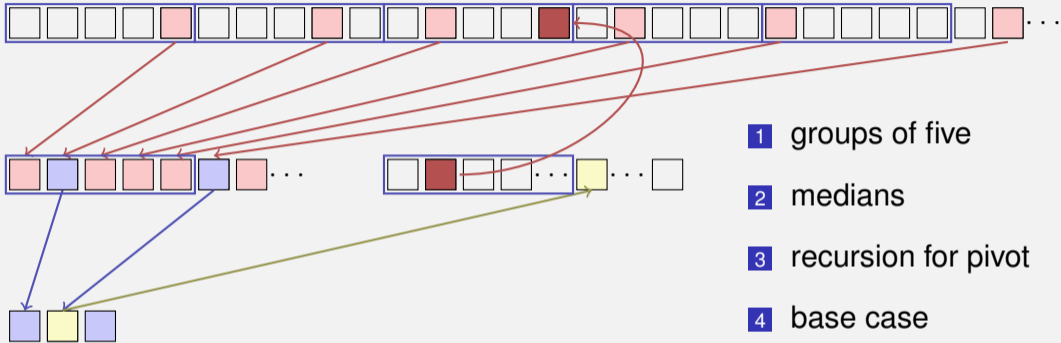
- 1 groups of five
- 2 medians
- 3 recursion for pivot
- 4 base case
- 5 pivot (level 1)
- 6 partition (level 1)

Median of medians



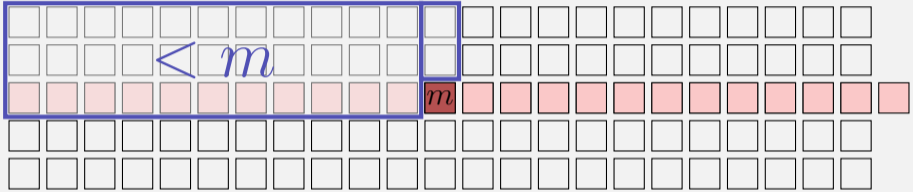
- 1 groups of five
- 2 medians
- 3 recursion for pivot
- 4 base case
- 5 pivot (level 1)
- 6 partition (level 1)
- 7 median = pivot level 0

Median of medians

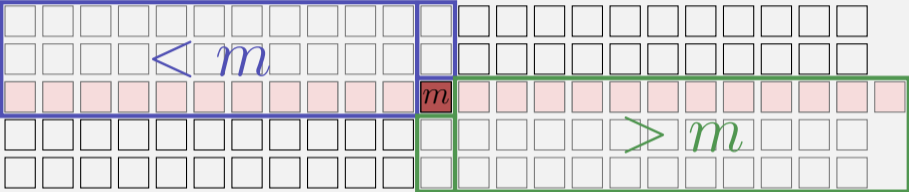


- 1 groups of five
- 2 medians
- 3 recursion for pivot
- 4 base case
- 5 pivot (level 1)
- 6 partition (level 1)
- 7 median = pivot level 0
- 8 2. recursion starts

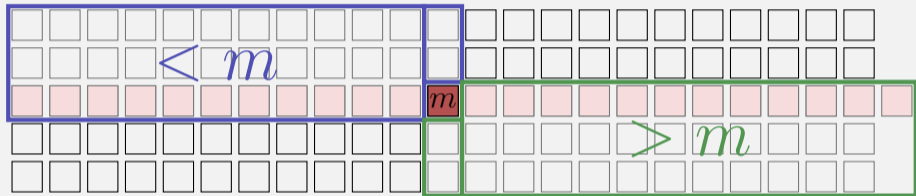
How good is this?



How good is this?

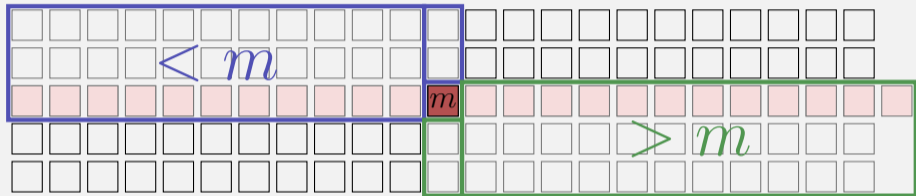


How good is this?



Number points left / right of the median of medians (without median group and the rest group) $\geq 3 \cdot (\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2) \geq \frac{3n}{10} - 6$

How good is this?



Number points left / right of the median of medians (without median group and the rest group) $\geq 3 \cdot (\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2) \geq \frac{3n}{10} - 6$

Second call with maximally $\lceil \frac{7n}{10} + 6 \rceil$ elements.

Analysis

Recursion inequality:

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lceil \frac{7n}{10} + 6 \right\rceil\right) + d \cdot n.$$

with some constant d .

Claim:

$$T(n) = \mathcal{O}(n).$$

Proof

Base clause: choose c large enough such that

$$T(n) \leq c \cdot n \text{ für alle } n \leq n_0.$$

Induction hypothesis:

$$T(i) \leq c \cdot i \text{ für alle } i < n.$$

Induction step:

$$\begin{aligned} T(n) &\leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lceil \frac{7n}{10} + 6 \right\rceil\right) + d \cdot n \\ &= c \cdot \left\lceil \frac{n}{5} \right\rceil + c \cdot \left\lceil \frac{7n}{10} + 6 \right\rceil + d \cdot n. \end{aligned}$$

Proof

Induction step:

$$\begin{aligned} T(n) &\leq c \cdot \left\lceil \frac{n}{5} \right\rceil + c \cdot \left\lceil \frac{7n}{10} + 6 \right\rceil + d \cdot n \\ &\leq c \cdot \frac{n}{5} + c + c \cdot \frac{7n}{10} + 6c + c + d \cdot n = \frac{9}{10} \cdot c \cdot n + 8c + d \cdot n. \end{aligned}$$

Choose $c \geq 80 \cdot d$ and $n_0 = 91$.

$$T(n) \leq \frac{72}{80} \cdot c \cdot n + 8c + \frac{1}{80} \cdot c \cdot n = c \cdot \underbrace{\left(\frac{73}{80}n + 8 \right)}_{\leq n \text{ für } n > n_0} \leq c \cdot n.$$

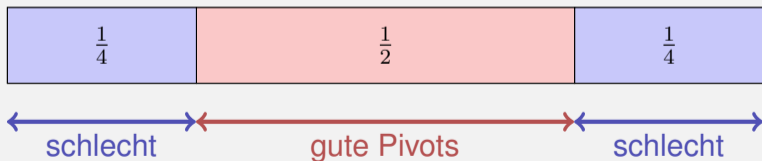
Result

Theorem

The k -th element of a sequence of n elements can, in the worst case, be found in $\Theta(n)$ steps.

Overview

- | | |
|----------------------------------|-----------------------------|
| 1. Repeatedly find minimum | $\mathcal{O}(n^2)$ |
| 2. Sorting and choosing $A[i]$ | $\mathcal{O}(n \log n)$ |
| 3. Quickselect with random pivot | $\mathcal{O}(n)$ expected |
| 4. Median of Medians (Blum) | $\mathcal{O}(n)$ worst case |



5.1 Appendix

Derivation of some mathematical formulas

[Expected value of the Geometric Distribution]

Random variable $X \in \mathbb{N}^+$ with $\mathbb{P}(X = k) = (1 - p)^{k-1} \cdot p$.

Expected value

$$\begin{aligned}\mathbb{E}(X) &= \sum_{k=1}^{\infty} k \cdot (1 - p)^{k-1} \cdot p = \sum_{k=1}^{\infty} k \cdot q^{k-1} \cdot (1 - q) \\ &= \sum_{k=1}^{\infty} k \cdot q^{k-1} - k \cdot q^k = \sum_{k=0}^{\infty} (k + 1) \cdot q^k - k \cdot q^k \\ &= \sum_{k=0}^{\infty} q^k = \frac{1}{1 - q} = \frac{1}{p}.\end{aligned}$$