5 Divide & Conquer

11 November 2024

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2024/25)
Philipps-Universität Marburg
version `2024-11-11 16:01`

Learning Outcomes

Unit 5: ***Divide & Conquer***

1. Know the steps of the Divide & Conquer paradigm.
2. Be able to solve simple Divide & Conquer recurrences.
3. Be able to design and analyze new algorithms using the Divide & Conquer paradigm.
4. Know the performance characteristics of selection-by-rank algorithms.

Outline

5 Divide & Conquer

Divide and conquer

Divide and conquer *idiom*   (Latin: *divide et impera*)
to make a group of people disagree and fight with one another
so that they will not join together against one   (Merriam-Webster Dictionary)

$\rightsquigarrow$ in politics & algorithms, many independent, small problems are better than one big one!

Divide-and-conquer algorithms:

1. Break problem into smaller, independent subproblems. (Divide!)
2. Recursively solve all subproblems. (Conquer!)
3. Assemble solution for original problem from solutions for subproblems.

Divide and conquer

Divide and conquer *idiom* (Latin: *divide et impera*)
to make a group of people disagree and fight with one another
so that they will not join together against one (Merriam-Webster Dictionary)

$\rightsquigarrow$ in politics & algorithms, many independent, small problems are better than one big one!

Divide-and-conquer algorithms:

1. Break problem into smaller, independent subproblems. (Divide!)
2. Recursively solve all subproblems. (Conquer!)
3. Assemble solution for original problem from solutions for subproblems.

Examples:

- Mergesort
- Quicksort
- Binary search
- (arguably) Tower of Hanoi

5.1 Divide & Conquer Recurrences

Back-of-the-envelope analysis

- before working out the details of a D&C idea,
  it is often useful to get a quick indication of the resulting performance
  - don't want to waste time on something that's not competitive in the end anyways!
- since D&C is naturally recursive, running time often not obvious
  instead: given by a recursive equation

Back-of-the-envelope analysis

- before working out the details of a D&C idea,
  it is often useful to get a quick indication of the resulting performance
  - don't want to waste time on something that's not competitive in the end anyways!
- since D&C is naturally recursive, running time often not obvious
  instead: given by a recursive equation
- unfortunately, rigorous analysis often tricky
  - Remember mergesort?

$$C(n) = \begin{cases} 0 & n \le 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \ge 2 \end{cases}$$

$$\leadsto C(n) = 2n\lfloor \lg(n) \rfloor + 2n - 4 \cdot 2^{\lfloor \lg(n) \rfloor} \quad 😱$$

$$= \Theta(n \log n) \quad 😌$$

Back-of-the-envelope analysis

- before working out the details of a D&C idea,
  it is often useful to get a quick indication of the resulting performance
  - don't want to waste time on something that's not competitive in the end anyways!
- since D&C is naturally recursive, running time often not obvious
  instead: given by a recursive equation
- unfortunately, rigorous analysis often tricky
  - Remember mergesort?

$$C(n) = \begin{cases} 0 & n \le 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \ge 2 \end{cases}$$

$$\rightsquigarrow C(n) = 2n\lfloor \lg(n) \rfloor + 2n - 4 \cdot 2^{\lfloor \lg(n) \rfloor} \text{ 😱}$$

$$= \Theta(n \log n) \text{ 😌}$$

- the following method works for many typical cases to give the right **order of growth**

The Master Method

- Assume a stereotypical D&C algorithm
  - a recursive calls on (for some constant $a \geq 1$)
  - subproblems of size n/b (for some constant $b > 1$)
  - with non-recursive "conquer" effort $f(n)$ (for some function $f : \mathbb{R} \to \mathbb{R}$)
  - base case effort d (some constant $d > 0$)

Mergesort

$a = 2$

$b = 2$

$f(n) = 2n$

base case $(n = 1 \rightsquigarrow d = 0)$

$n = 2 \rightsquigarrow d = 2$

The Master Method

- Assume a stereotypical D&C algorithm
  - a recursive calls on (for some constant $a \geq 1$)
  - subproblems of size n/b (for some constant $b > 1$)
  - with non-recursive "conquer" effort $f(n)$ (for some function $f : \mathbb{R} \to \mathbb{R}$)
  - base case effort d (some constant $d > 0$)

$\frac{n}{b} \in \mathbb{Z}$?

$\rightsquigarrow$ running time $T(n)$ satisfies

$$\boxed{T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ d & n \leq 1 \end{cases}}$$

The Master Method

- Assume a stereotypical D&C algorithm
  - a recursive calls on (for some constant $a \geq 1$)
  - subproblems of size n/b (for some constant $b > 1$)
  - with non-recursive "conquer" effort $f(n)$ (for some function $f : \mathbb{R} \to \mathbb{R}$)
  - base case effort d (some constant $d > 0$)

$\rightsquigarrow$ running time $T(n)$ satisfies

$$\boxed{T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ d & n \leq 1 \end{cases}}$$

Theorem 5.1 (Master Theorem)

With $\underline{c := \log_b(a)}$, we have for the above recurrence:

(a) $T(n) = \Theta(n^c)$ if $f(n) = O(n^{c-\varepsilon})$ for constant $\varepsilon > 0$.

(b) $T(n) = \Theta(n^c \log n)$ if $f(n) = \Theta(n^c)$.

(c) $T(n) = \Theta(f(n))$ if $f(n) = \Omega(n^{c+\varepsilon})$ for constant $\varepsilon > 0$ **and** f satisfies the regularity condition $\exists n_0, \alpha < 1\ \forall n \geq n_0 : a \cdot f\left(\frac{n}{b}\right) \leq \alpha f(n)$.

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

$$= a\left(a T\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)\right) + f(n)$$

$$= a^2 \cdot T\left(\frac{n}{b^2}\right) + f(n) + a\,f\left(\frac{n}{b}\right)$$

$$= a^3\,T\left(\frac{n}{b^3}\right) + f(n) + a\,f\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right)$$

$$\vdots$$

$\ell = \log_b(n)$

$$= a^\ell \underbrace{T(1)}_{=d} + \sum_{i=0}^{\ell} a^i f\left(\frac{n}{b^i}\right)$$

$$= \underbrace{a^{\log_b(n)}}_{\substack{\shortparallel \\ e^{\ln a \frac{\ln(n)}{\ln(b)}} \\ = n^{\log_b(a)}}} \cdot d + \sum_{i=0}^{\log_b(n)} a^i f\left(\frac{n}{b^i}\right)$$

Case 1: terms with i near $\log_b(n)$ dominate

Master Theorem – Intuition & Proof Idea

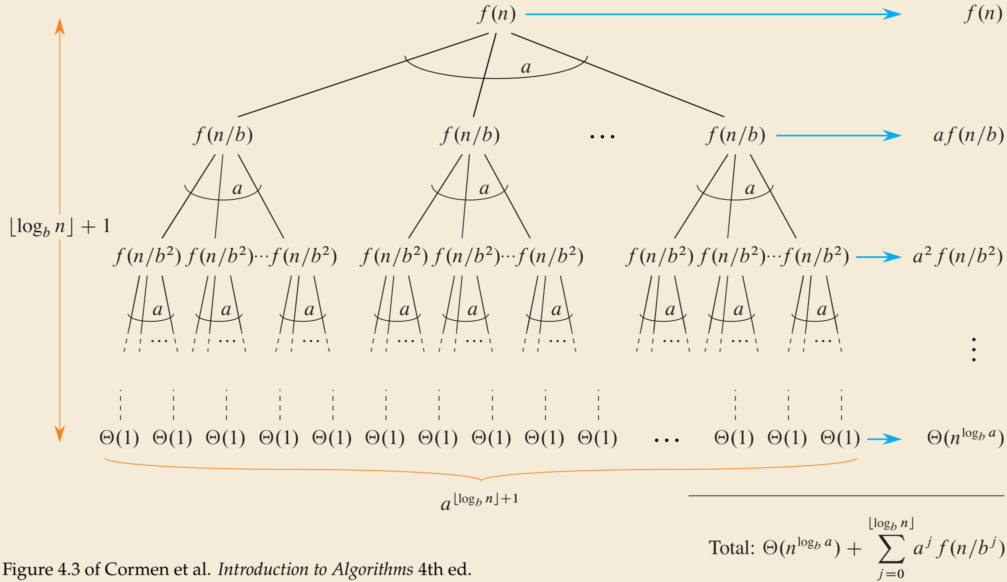


Figure 4.3 of Cormen et al. *Introduction to Algorithms* 4th ed.

Example: Mergesort

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

$$a = b = 2$$

$$f(n) = 2n$$

$$c = \log_b(a) = 1$$

$$f(n) \quad \text{vs.} \quad n^c$$

$$2n = \Theta(n^c)$$

$\Rightarrow$ Case 2 applies

$$T(n) = \Theta(f(n) \cdot \log n)$$

$$= \Theta(n \log n)$$

When it's fine to ignore floors and ceilings

The ***polynomial-growth condition***

- $f : \mathbb{R}_{>0} \to \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \; \forall C \geq 1 \; \exists D > 1 \quad \forall n \geq n_0 \; \forall c \in [1, C] \;\; : \;\; \tfrac{1}{D} f(n) \;\leq\; f(cn) \;\leq\; D f(n)$$

When it's fine to ignore floors and ceilings

The ***polynomial-growth condition***

- $f : \mathbb{R}_{>0} \to \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \; \forall C \geq 1 \; \exists D > 1 \quad \forall n \geq n_0 \; \forall c \in [1, C] \;\; : \;\; \frac{1}{D} f(n) \;\leq\; f(cn) \;\leq\; D f(n)$$

- intuitively: increasing n by up to a factor C (and anywhere in between!)
  changes the function value by at most a factor $D = D(C)$
  (for sufficiently large n)

- examples: $f(n) = \Theta(n^{\alpha} \log^{\beta}(n) \log\log^{\gamma}(n))$ for constants α, β, γ (zero allowed)
  $\rightsquigarrow$ f satisfies the polynomial-growth condition

When it's fine to ignore floors and ceilings

The ***polynomial-growth condition***

- $f : \mathbb{R}_{>0} \to \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \ \forall C \geq 1 \ \exists D > 1 \quad \forall n \geq n_0 \ \forall c \in [1, C] \ : \ \frac{1}{D} f(n) \ \leq \ f(cn) \ \leq \ D f(n)$$

- intuitively: increasing n by up to a factor C (and anywhere in between!)
  changes the function value by at most a factor $D = D(C)$
  (for sufficiently large n)

- examples: $f(n) = \Theta(n^\alpha \log^\beta(n) \log\log^\gamma(n))$ for constants α, β, γ zero allowed
  $\rightsquigarrow$ f satisfies the polynomial-growth condition

Lemma 5.2 (Polynomial-growth master method)

If the toll function $f(n)$ satisfies the polynomial-growth condition,
then the Θ-class of the solution of a D&C recurrence remains the same
when ignoring floors and ceilings on subproblem sizes.

A Rigorous and Stronger Meta Theorem

∉ exam

Theorem 5.3 (Roura's Discrete Master Theorem)

Let $T(n)$ be recursively defined as

$$T(n) = \begin{cases} b_n & 0 \le n < n_0, \\ f(n) + \sum_{d=1}^{D} a_d \cdot T\left(\frac{n}{b_d} + r_{n,d}\right) & n \ge n_0, \end{cases}$$

where $D \in \mathbb{N}$, $a_d > 0$, $b_d > 1$, for $d = 1, \ldots, D$ are constants, functions $r_{n,d}$ satisfy $|r_{n,d}| = O(1)$ as $n \to \infty$, and function $f(n)$ satisfies $\underline{f(n) \sim B \cdot n^\alpha (\ln n)^\gamma}$ for constants $B > 0$, α, γ.
Set $H = 1 - \sum_{d=1}^{D} a_d \, (1/b_d)^\alpha$; then we have:

(a) If $H < 0$, then $T(n) = O(n^{\tilde{\alpha}})$, for $\tilde{\alpha}$ the unique value of α that would make $H = 0$.

(b) If $H = 0$ and $\gamma > -1$, then $T(n) \sim f(n) \ln(n)/\tilde{H}$ with constant $\tilde{H} = (\gamma + 1) \sum_{d=1}^{D} a_d \, b_d^{-\alpha} \ln(b_d)$.

(c) If $H = 0$ and $\gamma = -1$, then $T(n) \sim f(n) \ln(n) \ln(\ln(n))/\hat{H}$ with constant $\hat{H} = \sum_{d=1}^{D} a_d \, b_d^{-\alpha} \ln(b_d)$.

(d) If $H = 0$ and $\gamma < -1$, then $T(n) = O(n^\alpha)$.

(e) If $H > 0$, then $T(n) \sim f(n)/H$.

5.2 Order Statistics

Selection by Rank

- Standard data summary of numerical data: (Data scientists, listen up!)
  - mean, standard deviation
  - min/max (range)
  - histograms

  easy to compute in $\Theta(n)$ time

  - median, quartiles, other quantiles (a.k.a. order statistics)

computable in $\Theta(n)$ time?

Selection by Rank

- Standard data summary of numerical data: (Data scientists, listen up!)
  - mean, standard deviation
  - min/max (range)
  - histograms

  } easy to compute in $\Theta(n)$ time
  - median, quartiles, other quantiles (a.k.a. order statistics)

computable in $\Theta(n)$ time?

General form of problem: **Selection by Rank**

- **Given:** array $A[0..n)$ of numbers and number $k \in [0..n)$.
- **Goal:** find element that would be in position k if A was sorted (kth smallest element).

  but 0-based & counting dups
- $k = \lfloor n/2 \rfloor \rightsquigarrow$ median; $k = \lfloor n/4 \rfloor \rightsquigarrow$ lower quartile
  $k = 0 \rightsquigarrow$ minimum; $k = n - \ell \rightsquigarrow$ ℓth largest

Quickselect

- Key observation: Finding the element of rank k seems hard.
  But computing the rank of a given element is easy!
  count smaller elements

$\rightsquigarrow$ Pick any element $A[b]$ and find its rank j.

- $j = k$? $\rightsquigarrow$ Lucky Duck! Return chosen element and stop
- $j < k$? $\rightsquigarrow$ ... not done yet. But: The $j + 1$ elements smaller than $\le A[b]$ can be excluded!
- $j > k$? $\rightsquigarrow$ similarly exclude the $n - j$ elements $\ge A[b]$

Quickselect

- Key observation: Finding the element of rank k seems hard.
  But computing the rank of a given element is easy!
  count smaller elements

$\rightsquigarrow$ Pick any element $A[b]$ and find its rank j.

- $j = k$? $\rightsquigarrow$ Lucky Duck! Return chosen element and stop
- $j < k$? $\rightsquigarrow$ …not done yet. But: The $j + 1$ elements smaller than $\le A[b]$ can be excluded!
- $j > k$? $\rightsquigarrow$ similarly exclude the $n - j$ elements $\ge A[b]$

- partition function from Quicksort:
  - returns the rank of pivot
  - separates elements into smaller/larger

$\rightsquigarrow$ can use same building blocks

```
procedure quickselect(A[l..r], k)
    if r − ℓ ≤ 1 then return A[l]
    b := choosePivot(A[l..r])
    j := partition(A[l..r], b)
    if j == k
        return A[j]
    else if j < k
        quickselect(A[j + 1..n], k ~~− j − 1~~)
    else // j > k
        quickselect(A[0..j], k)
```

Quickselect – Iterative Code

Recursion can be replaced by loop (*tail-recursion elimination*)

procedure quickselect($A[l..r]$, k)
    if $r - \ell \le 1$ **then return** $A[l]$
    b := choosePivot($A[l..r]$)
    j := partition($A[l..r]$, b)
    if $j == k$
        return $A[j]$
    else if $j < k$
        quickselect($A[j+1..n]$, k)
    else // $j > k$
        quickselect($A[0..j]$, k)

procedure quickselectIterative($A[0..n)$, k)
    l := 0; r := n
    while $r - l > 1$
        b := choosePivot($A[l..r)$)
        j := partition($A[l..r)$, b)
        if $j \ge k$ **then** r := $j - 1$
        if $j \le k$ **then** l := $j + 1$
    return $A[k]$

- implementations should usually prefer iterative version
- analysis more intuitive with recursive version

Quickselect – Analysis

```
procedure quickselect(A[l..r], k)
    if r − ℓ ≤ 1 then return A[l]
    b := choosePivot(A[l..r])
    j := partition(A[l..r], b)
    if j == k
        return A[j]
    else if j < k
        quickselect(A[j + 1..n], k − j − 1)
    else // j > k
        quickselect(A[0..j), k)
```

- cost = #cmps
- costs depend on n **and** k

Quickselect – Analysis

```
procedure quickselect(A[l..r], k)
    if r − ℓ ≤ 1 then return A[l]
    b := choosePivot(A[l..r])
    j := partition(A[l..r], b)
    if j == k
        return A[j]
    else if j < k
        quickselect(A[j + 1..n], k − j − 1)
    else // j > k
        quickselect(A[0..j), k)
```

- cost = #cmps
- costs depend on n **and** k
- **worst case:** $k = 0$, but always $j = n - 2$
  - $\rightsquigarrow$ each recursive call makes n one smaller at cost $\Theta(n)$
  - $\rightsquigarrow$ $T(n, k) = \Theta(n^2)$ worst case cost

Quickselect – Analysis

```
procedure quickselect(A[l..r), k)
    if r − ℓ ≤ 1 then return A[l]
    b := choosePivot(A[l..r))
    j := partition(A[l..r), b)
    if j == k
        return A[j]
    else if j < k
        quickselect(A[j + 1..n), k − j − 1)
    else // j > k
        quickselect(A[0..j), k)
```

- cost = #cmps
- costs depend on n **and** k
- **worst case:** $k = 0$, but always $j = n - 2$
  - $\rightsquigarrow$ each recursive call makes n one smaller at cost $\Theta(n)$
  - $\rightsquigarrow$ $T(n,k) = \Theta(n^2)$ worst case cost

average case:

- let $T(n,k)$ expected cost when we choose a pivot uniformly from $A[0..n)$

$\rightsquigarrow$ formulate recurrence for $T(n,k)$ similar to BST/Quicksort recurrence

$$T(n,k) = \underbrace{n}_{\text{partition}} + \underset{\uparrow \atop \Pr[\text{pivot rank } r]}{\frac{1}{n}} \sum_{r=0}^{n-1} \underbrace{[r = k]}_{= \begin{cases} 1 & r=k \\ 0 & \text{else} \end{cases}} \cdot 0 + [k < r] \cdot T(r,k) + [k > r] \cdot T(n-r-1, k-r-1)$$

Quickselect – Average Case Analysis

- $T(n,k) \;=\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}[r=k]\cdot 0 \;+\; [k<r]\cdot T(r,k) \;+\; [k>r]\cdot T(n-r-1,k-r-1)$

- Set $\underline{\hat{T}(n)} \;=\; \max_{\underline{k\in[0..n)}} T(n,k)$

Quickselect – Average Case Analysis

- $T(n,k) \;=\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1} \underbrace{[r=k]\cdot 0}_{=0} \;+\; \underbrace{[k<r]\cdot T(r,k) \;+\; [k>r]\cdot T(n-r-1,k-r-1)}$

$\le \max\{T(r,k),\, T(n-r-1,k-r-1)\}$

- Set $\hat{T}(n) \;=\; \max_{k\in[0..n)} T(n,k)$

$\leadsto\; \hat{T}(n) \;\le\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1} \max\{\hat{T}(r), \hat{T}(n-r-1)\}$

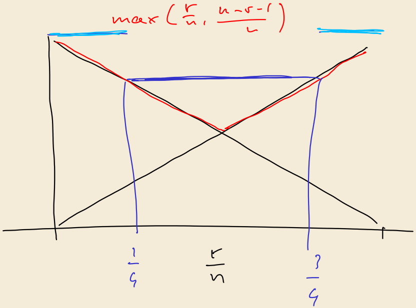
$\le \max\{\hat{T}(r),\, \hat{T}(n-r-1)\}$

Quickselect – Average Case Analysis

- $T(n,k) \;=\; n + \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}[r=k]\cdot 0 + [k<r]\cdot T(r,k) + [k>r]\cdot T(n-r-1,k-r-1)$

- Set $\hat{T}(n) \;=\; \max_{k\in[0..n)} T(n,k)$

$\rightsquigarrow$ $\hat{T}(n) \;\le\; n + \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}\max\{\hat{T}(r),\hat{T}(n-r-1)\}$



- analyze hypothetical, worse algorithm:
  if $r \notin [\frac{1}{4}n, \frac{3}{4}n)$, discard pivot and repeat with new one!

$\rightsquigarrow$ $\hat{T}(n) \le \tilde{T}(n)$ defined by $\quad \tilde{T}(n) \;\le\; n + \underline{\tfrac{1}{2}\tilde{T}(n)} + \underline{\tfrac{1}{2}\tilde{T}(\tfrac{3}{4}n)}$

Quickselect – Average Case Analysis

- $T(n,k) \;=\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}[r=k]\cdot 0 \;+\; [k<r]\cdot T(r,k) \;+\; [k>r]\cdot T(n-r-1, k-r-1)$

- Set $\hat{T}(n) \;=\; \max_{k\in[0..n)} T(n,k)$

$\rightsquigarrow\; \hat{T}(n) \;\le\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}\max\{\hat{T}(r), \hat{T}(n-r-1)\}$

- analyze hypothetical, worse algorithm:
  if $r \notin [\frac{1}{4}n, \frac{3}{4}n)$, discard pivot and repeat with new one!

$\rightsquigarrow\; \hat{T}(n) \le \tilde{T}(n)$ defined by $\quad \tilde{T}(n) \;\le\; n \;+\; \frac{1}{2}\tilde{T}(n) \;+\; \frac{1}{2}\tilde{T}(\frac{3}{4}n) \quad \mid \cdot 2 \quad - \tilde{T}(n)$

$\rightsquigarrow\; \tilde{T}(n) \;\le\; 2n \;+\; \tilde{T}(\frac{3}{4}n)$

$a = 1$

$b = \frac{4}{3}$

Quickselect – Average Case Analysis

- $T(n,k) \;=\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}[r=k]\cdot 0 \;+\; [k<r]\cdot T(r,k) \;+\; [k>r]\cdot T(n-r-1,k-r-1)$
- Set $\hat{T}(n) \;=\; \max_{k\in[0..n)} T(n,k)$

$\rightsquigarrow$ $\hat{T}(n) \;\le\; n \;+\; \dfrac{1}{n}\displaystyle\sum_{r=0}^{n-1}\max\{\hat{T}(r),\hat{T}(n-r-1)\}$

- analyze hypothetical, worse algorithm:
  if $r \notin [\frac{1}{4}n, \frac{3}{4}n)$, discard pivot and repeat with new one!

$\rightsquigarrow$ $\hat{T}(n) \le \tilde{T}(n)$ defined by $\quad \tilde{T}(n) \;\le\; n \;+\; \frac{1}{2}\tilde{T}(n) \;+\; \frac{1}{2}\tilde{T}(\frac{3}{4}n)$

$\rightsquigarrow$ $\tilde{T}(n) \;\le\; 2n \;+\; \tilde{T}(\frac{3}{4}n)$

- Master Theorem Case 3: $\tilde{T}(n) = \Theta(n)$

Quickselect Discussion

- $\Theta(n^2)$ worst case (like Quicksort)
- expected cost $\Theta(n)$ (best possible)
- no extra space needed
- adaptations possible to find several order statistics at once

Quickselect Discussion

- $\Theta(n^2)$ worst case (like Quicksort)
- expected cost $\Theta(n)$ (best possible)
- no extra space needed
- adaptations possible to find several order statistics at once

- expected cost can be further improved by choosing pivot from a small sorted sample
  - $\rightsquigarrow$ asymptotically optimal randomized cost: $n + \min\{k, n-k\}$ comparisons in expectation achieved asymptotically by the *Floyd-Rivest algorithm*