Martin Ziegler

Syllabus

2. Searching

- Linear Search
- Neighbor Search
- Binary Search
- Uniqueness
- Hashing
- Order Statistics/ Median
- 1D Range Searching/Counting
- 2D Range Searching/Counting

- Specification
- Primitives: semantics and cost
- Design
- Analysis
- (Optimality)

2. Searching

linear search

Specification: Fix set X.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$ in array $x[1 \ldots N]$
as well as $y \in X$.

Output: $n \in \{1, \ldots N\}$ such that $x_n = y$ or $n=0$ if $x_j \neq y$

Primitives: Access $\mathbb{N} \ni n \to x[n] \in X$ cost 1
Comparison "$x=y$?" cost 1
Index integer arithmetic cost 1

Algorithm:

```
for n=1…N
  if x[n]=y then return n;
return 0.
```

runtime $O(N)$

correctness: √

2. Searching *neighbor* search

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$ in array $x[1 \ldots N]$
as well as $y \in X$.

Output: $n \in \{0, \ldots N\}$ such that $x_n \leq y < x_{n+1}$ where $x_{N+1} := \infty$, $x_0 := -\infty$

Primitives: Access $\mathbb{N} \ni n \to x[n] \in X$ cost 1
ordered comparison "$x \leq y$?" cost 1
Index integer arithmetic cost 1

```
if x[1] > y then return 0;
for n=1…N
  if x[n] ⊗ y then return n-1;
return N.
```

runtime $O(N)$

correctness: √

2. Searching

binary search

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$ as well as $y \in X$.

in array $x[1 \ldots N]$ with $x_1 \leq \ldots \leq x_N$

Output: smallest $r \leq N+1$ such that $y < x_r$

where $x_{N+1} := \infty$, $x_0 := -\infty$

Primitives: Access $\mathbb{N} \ni n \to x[n] \in X$ cost 1

ordered comparison "$x \leq y$?" cost 1

Index integer arithmetic cost 1

```
l:=0; r:=N+1;
while l+1<r do
  n := ⌊(l+r)/2⌋;
  if y<x[n] then r:=n else l:=n;
```

runtime $O(\log N)$

correctness: $x[l] \leq y < x[r]$

time: $r'-l' \leq \lceil (r-l)/2 \rceil$

2. Searching

uniqueness

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$ in array $x[1 \ldots N]$ with $x_1 \leq \ldots \leq x_N$

Output: **1** if all elements are distinct: $\forall i,j: x_i = x_j \Rightarrow i = j$

0 if some element is repeated: $\exists i \neq j: x_i = x_j$

Primitives: Access $\mathbb{N} \ni n \to x[n] \in X$ cost 1

ordered comparison "$x \leq y$?" cost 1

Uniq1 ($x[1..N]$) runtime $O(N^2)$

```
For m:=2 to N do
  For k:=1 to m-1 do
    If x[m]=x[k] Return 0;
Return 1;
```

Uniq2 ($x[1..N]$) runtime $O(N)$

```
For m:=1 to N−1 do
  If x[m]=x[m+1] Return 0;
Return 1;
```

2. Searching

uniqueness

Specification: Fix set $X=\{0,\ldots M\text{-}1\}$

Input: $N\in\mathbb{N}$ and finite sequence $x_1,\ldots x_N\in X$ in array $x[1\ldots N]$

Output: **1** if all elements are distinct: $\forall i,j:\ x_i=x_j \ \Rightarrow\ i=j$

0 if some element is repeated: $\exists i\neq j:\ x_i=x_j$

Primitives: Access $\mathbb{N}\ni n\to x[n]\in X$ cost 1

Uniq3 ($x[1..N]$)

Boolean array *present*[0…*M*-1]; // initialized with **0/false**

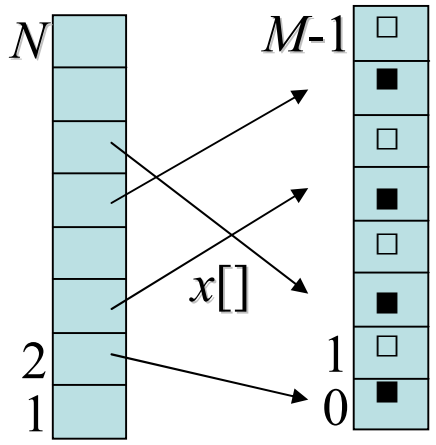
For *n*:=1 to *N* do

  If *present*[*x*[*n*]] then return **0**;

  present[*x*[*n*]] := **1**;

Endfor; return **1**

runtime $O(N)$
memory $O(M)$



2. Searching

hash functions
$\varphi_k(y) = y \cdot k \bmod P$

(division) hashing

Specification: Fix set $X=\{1,\ldots M\text{-}1\}$

Input: $N \in \mathbb{N}$ and finite sequence $x_1,\ldots x_N \in X$ in array $x[1\ldots N]$

Output: **1** if all elements are distinct: $\forall i,j:\ x_i=x_j \ \Rightarrow\ i=j$

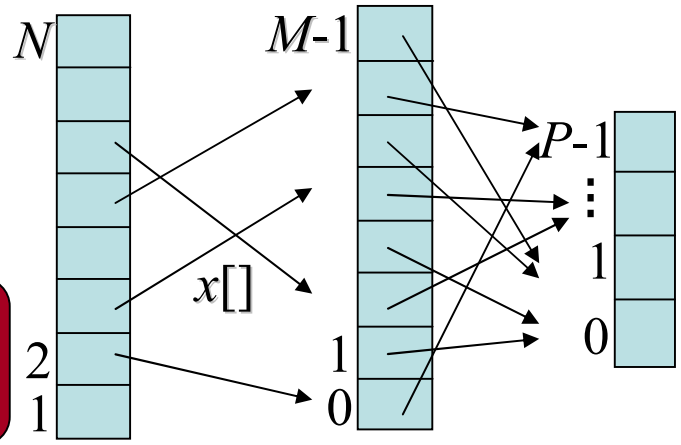
0 if some element is repeated: $\exists i\neq j:\ x_i=x_j$

Primitives: Arithmetic (on values!)

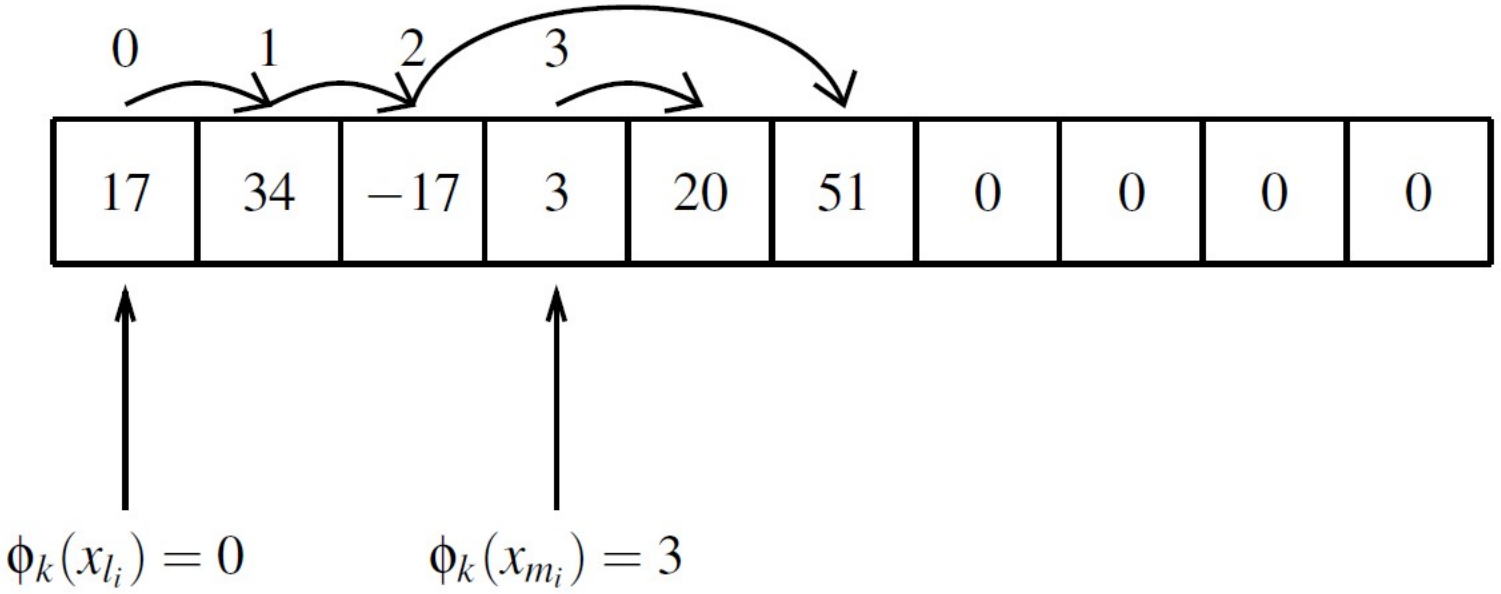
Let $P \geq N$ be prime.
Guess random
$k \in \{1,\ldots P\text{-}1\}$

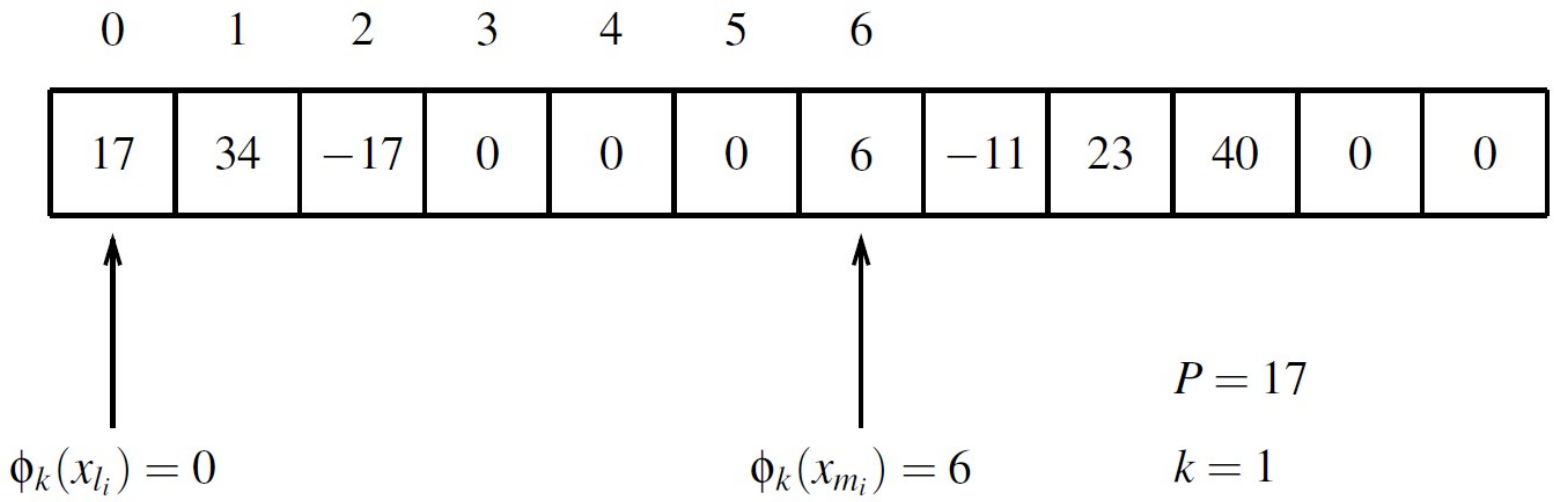
```
Integer array y[0…P-1];  // initialized with 0s
For n:=1 to N do ;   p := x[n]·k  mod  P;
  While  y[p]≠0  do
     If  y[p]=x[n]  then return 0;
     p := (p + 1) mod P ;  Endwhile ;
  y[p] := x[n] ;
Endfor;  return 1
```

runtime $O(?)$
memory $O(P)$



Hashing: Illustrated Example





2. Searching

order statistics/ median

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$ as well as $K \in \{1, \ldots, N\}$.

in array $x[1 \ldots N]$ with $x_1 \leq \ldots \leq x_N$

Output: $m \in \{1, \ldots N\}$ s.t. $\#\{n : x_n < x_m\} < K \leq \#\{n : x_n \leq x_m\}$

Primitives: Access $\mathbb{N} \ni n \to x[n] \in X$ cost 1

ordered comparison "$x \leq y$?" cost 1

OrderStat1 ($x[1..N]$; K) runtime $O(N^2)$

For m:=1 to N do

  a:=0; b:=0; For n:=1 to N do

    If $x[n] < x[m]$ then a:=a+1;

    If $x[n] \leq x[m]$ then b:=b+1;

  If $a < K \leq b$ then Return m;

OrderStat2 ($x[1..N]$; K)

Return K;

runtime $O(1)$

2. Searching

1D Range Counting

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$ in array $x[1 \ldots N]$
as well as $x < x' \in X$. with $x_1 \leq \ldots \leq x_N$

Output: $\#\{ m : x < x_m \leq x' \} \in \mathbb{N}$

Primitives: Access $\mathbb{N} \ni n \to x[n] \in X$
ordered comparison

r=**BinSearch**($x[], y$):
smallest $r \leq N+1$
such that $y < x_r$

Range1 ($x[]$; x , x')
t:=0; For m:=1 to N do
If $x < x[m] \leq x'$
then $t := t+1$;

runtime $O(N)$

Range2 ($x[]$; x , x')
l := BinSearch($x[]$; x)
r := BinSearch($x[]$; x')
If r-$l \leq 0$ then Return 0
else Return r-l;

runtime $O(\log N)$

2. Searching

1D Range Counting,

Search Tree

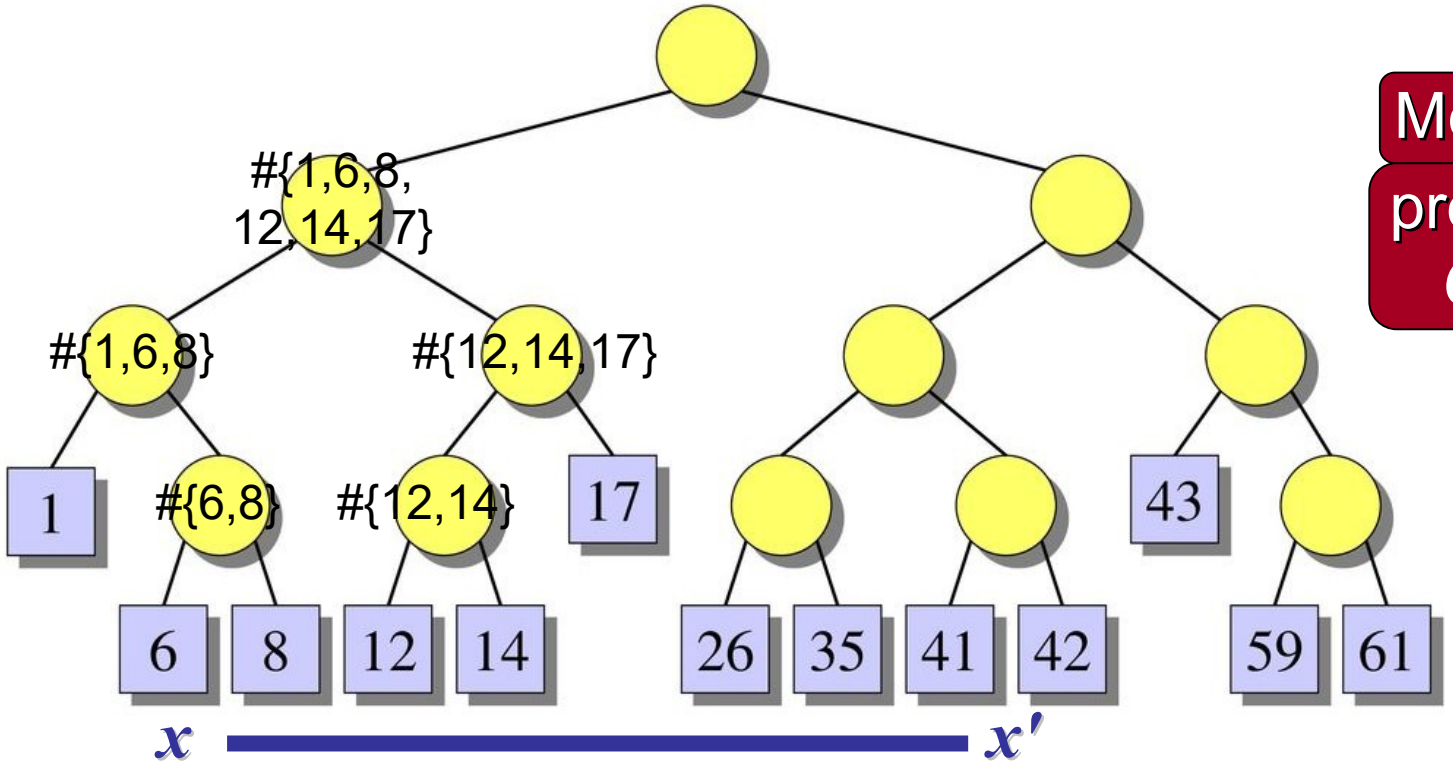
(=balanced binary tree)

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$
as well as $x < x' \in X$.

with $x_1 \leq \ldots \leq x_N$

Output: $\#\{ m : x < x_m \leq x' \} =: K$



Memory $O(N)$

preprocessing $O(N \cdot \log N)$

Depth/
runtime $O(\log N)$

2. Searching

1D Range Searching, *Search Tree*

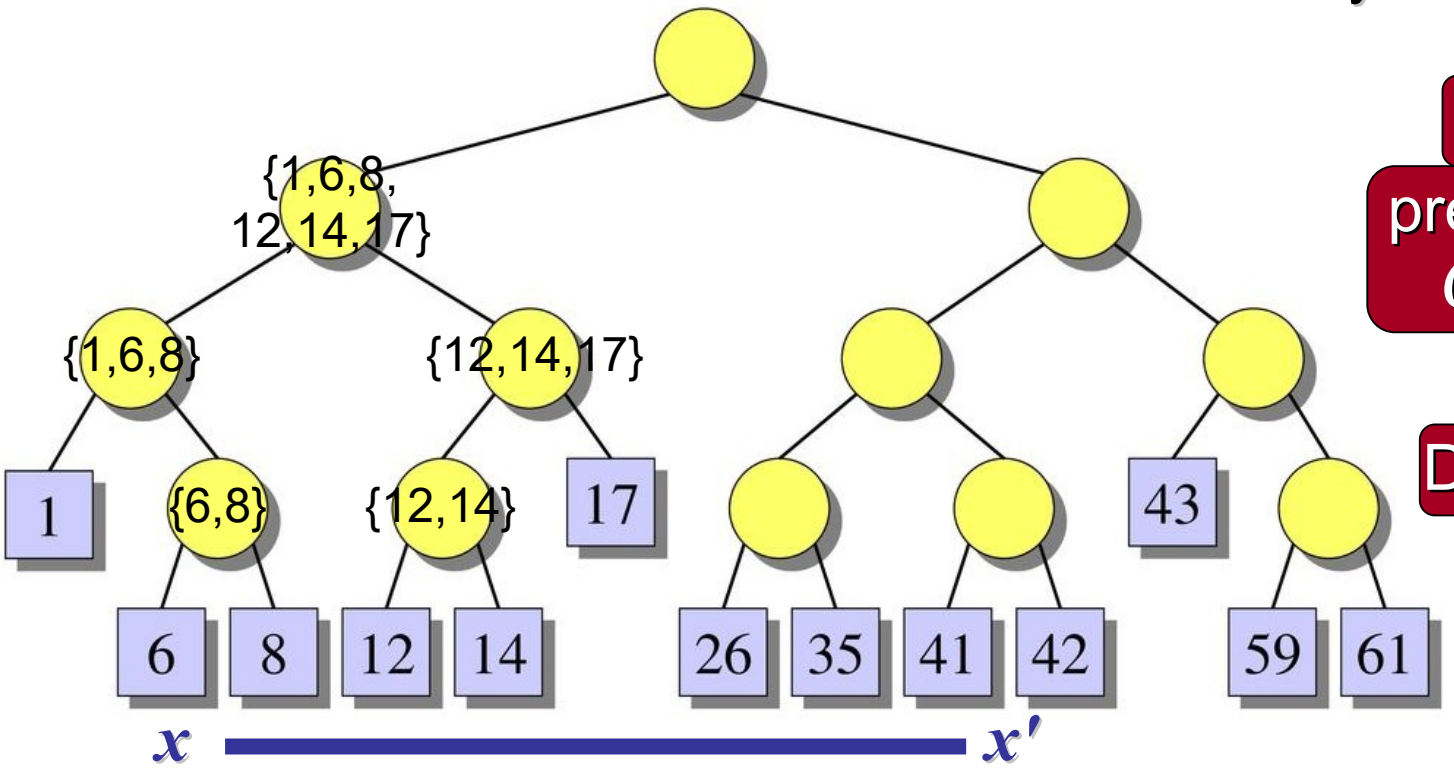
Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1, \ldots x_N \in X$
as well as $x < x' \in X$.

with $x_1 \leq \ldots \leq x_N$

Output: $\{ m : x < x_m \leq x' \}$

not a number, but a set
cardinality =: $K \in \{0 \ldots N\}$



Memory / preprocessing $O(N \cdot \log N)$

Depth $O(\log N)$
runtime $O(K + \log N)$

2. Searching

2D Range Counting

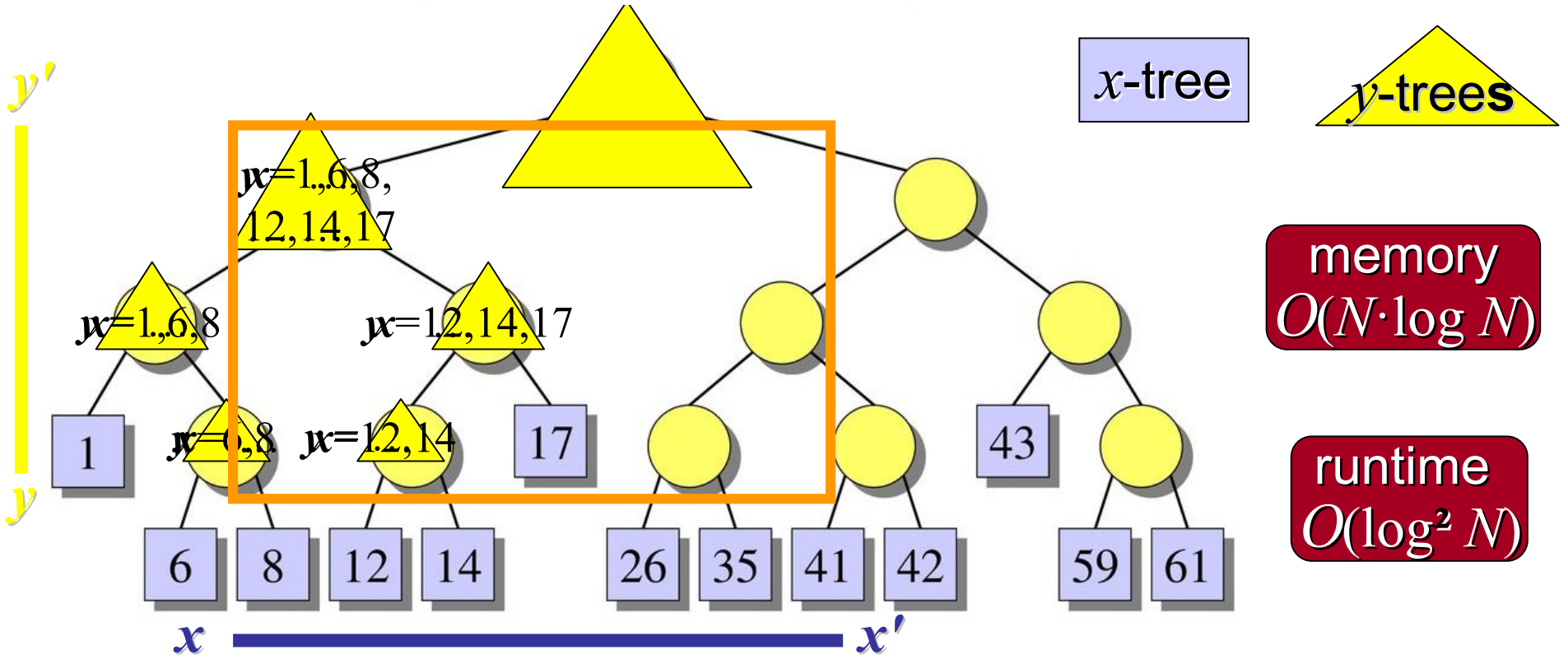
Nested Trees

Specification: Fix sets X, Y with total orders $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $(x_1, y_1), \ldots (x_N, y_N) \in X \times Y$
as well as $(x, y) < (x', y')$.

componentwise

Output: $\#\{ m : (x, y) < (x_m, y_m) \leq (x', y') \} \in \mathbb{N}$



2. Searching

2D Range Searching

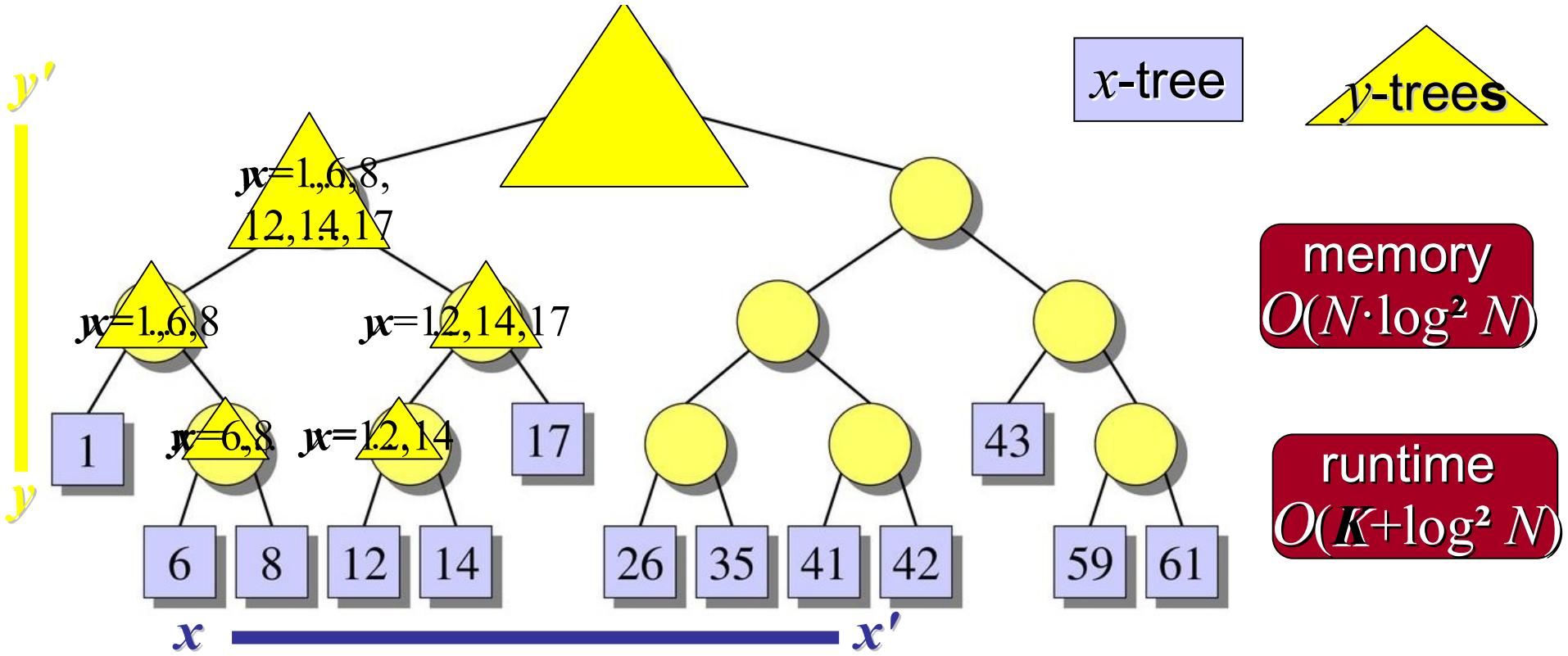
Nested Trees

Specification: Fix sets X, Y with total orders $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $(x_1, y_1), \ldots (x_N, y_N) \in X \times Y$
as well as $(x, y) < (x', y')$.

componentwise

Output: $\#\{\, m : (x, y) < (x_m, y_m) \leq (x', y') \,\} \in \mathbb{N}$



memory
$O(N \cdot \log^2 N)$

runtime
$O(K + \log^2 N)$

Recap

2. Searching

- Linear Search
- Neighbor Search
- Binary Search
- Uniqueness
- Hashing
- Order Statistics/ Median
- 1D Range Searching/Counting
- 2D Range Searching/Counting

- Specification
- Primitives: semantics and cost
- Design
- Analysis
- (Optimality)

2. Searching

Martin Ziegler

Quiz

a) What is the purpose of *algorithm design*?

1) To understand what the algorithm is supposed to do?
2) To count the number of primitive operations needed?
3) To prove that the algorithm is correct?
4) To have the right idea how to accomplish the specification of the algorithm?

b) What is true about *neighbor search*?

1) When the array is sorted, then the run time is logarithmic.
2) When the array is sorted, then the output is unique.
3) It is slower than sorting the array and using binary search.
4) When the array is not sorted, then it does not terminate

c) What is a *necessary* condition for the construction of the hash function (so that the algorithm terminates)?

1) the number P is prime
2) k is relatively prime to P
3) $P \geq N$
4) k is not zero