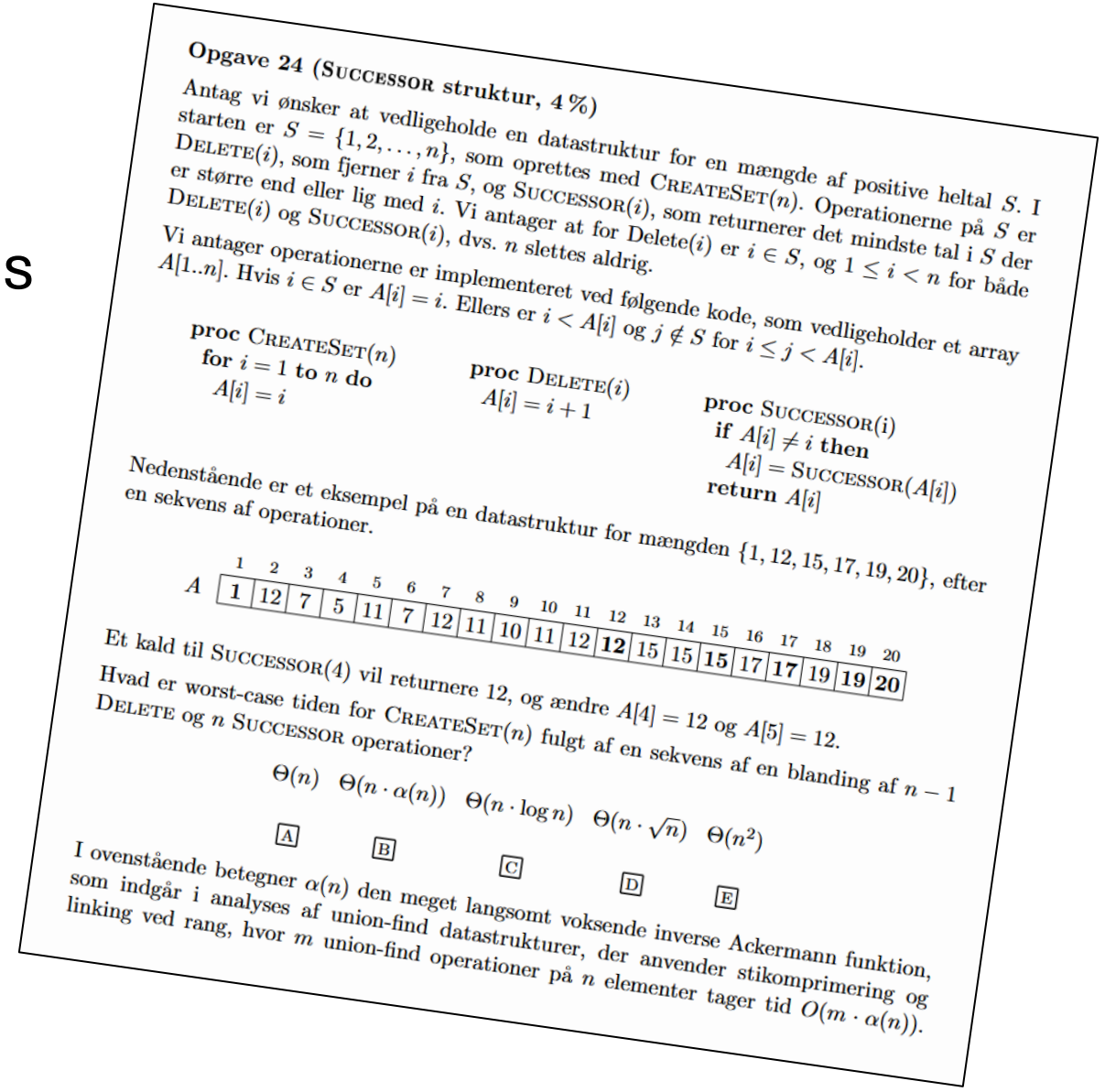
A Simple Integer Successor-Delete Data Structure

Gerth Stølting Brodal
Aarhus University

23rd International Symposium on Experimental Algorithms (SEA 2025), July 22, 2025, Venice, Italy

Background

- Algorithms and Data Structures
- Exam question
- Amortized analysis
- Simple data structure



Opgave 24 (SUCCESSOR struktur, 4 %)

Antag vi ønsker at vedligeholde en datastruktur for en mængde af positive heltal S. I starten er $S = \{1, 2, \ldots, n\}$, som oprettes med CREATESET(n). Operationerne på S er DELETE(i), som fjerner i fra S, og SUCCESSOR(i), som returnerer det mindste tal i S der er større end eller lig med i. Vi antager at for Delete(i) er $i \in S$, og $1 \leq i < n$ for både DELETE(i) og SUCCESSOR(i), dvs. n slettes aldrig.

Vi antager operationerne er implementeret ved følgende kode, som vedligeholder et array $A[1..n]$. Hvis $i \in S$ er $A[i] = i$. Ellers er $i < A[i]$ og $j \notin S$ for $i \leq j < A[i]$.

```
proc CreateSet(n)
  for i = 1 to n do
    A[i] = i

proc Delete(i)
  A[i] = i + 1

proc Successor(i)
  if A[i] ≠ i then
    A[i] = Successor(A[i])
  return A[i]
```

Nedenstående er et eksempel på en datastruktur for mængden $\{1, 12, 15, 17, 19, 20\}$, efter en sekvens af operationer.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	**1**	12	7	5	11	7	12	11	10	11	12	**12**	15	15	**15**	17	**17**	19	**19**	**20**

Et kald til SUCCESSOR(4) vil returnere 12, og ændre $A[4] = 12$ og $A[5] = 12$.

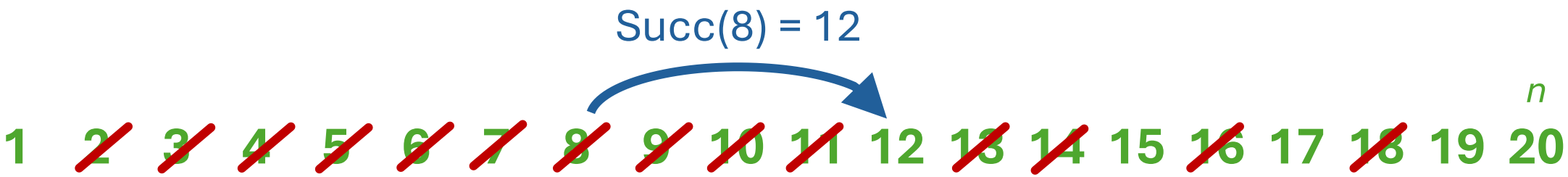
Hvad er worst-case tiden for CREATESET(n) fulgt af en sekvens af en blanding af $n - 1$ DELETE og n SUCCESSOR operationer?

$\Theta(n)$ (A) $\Theta(n \cdot \alpha(n))$ (B) $\Theta(n \cdot \log n)$ (C) $\Theta(n \cdot \sqrt{n})$ (D) $\Theta(n^2)$ (E)

I ovenstående betegner $\alpha(n)$ den meget langsomt voksende inverse Ackermann funktion, som indgår i analyses af union-find datastrukturer, der anvender stikomprimering og linking ved rang, hvor m union-find operationer på n elementer tager tid $O(m \cdot \alpha(n))$.

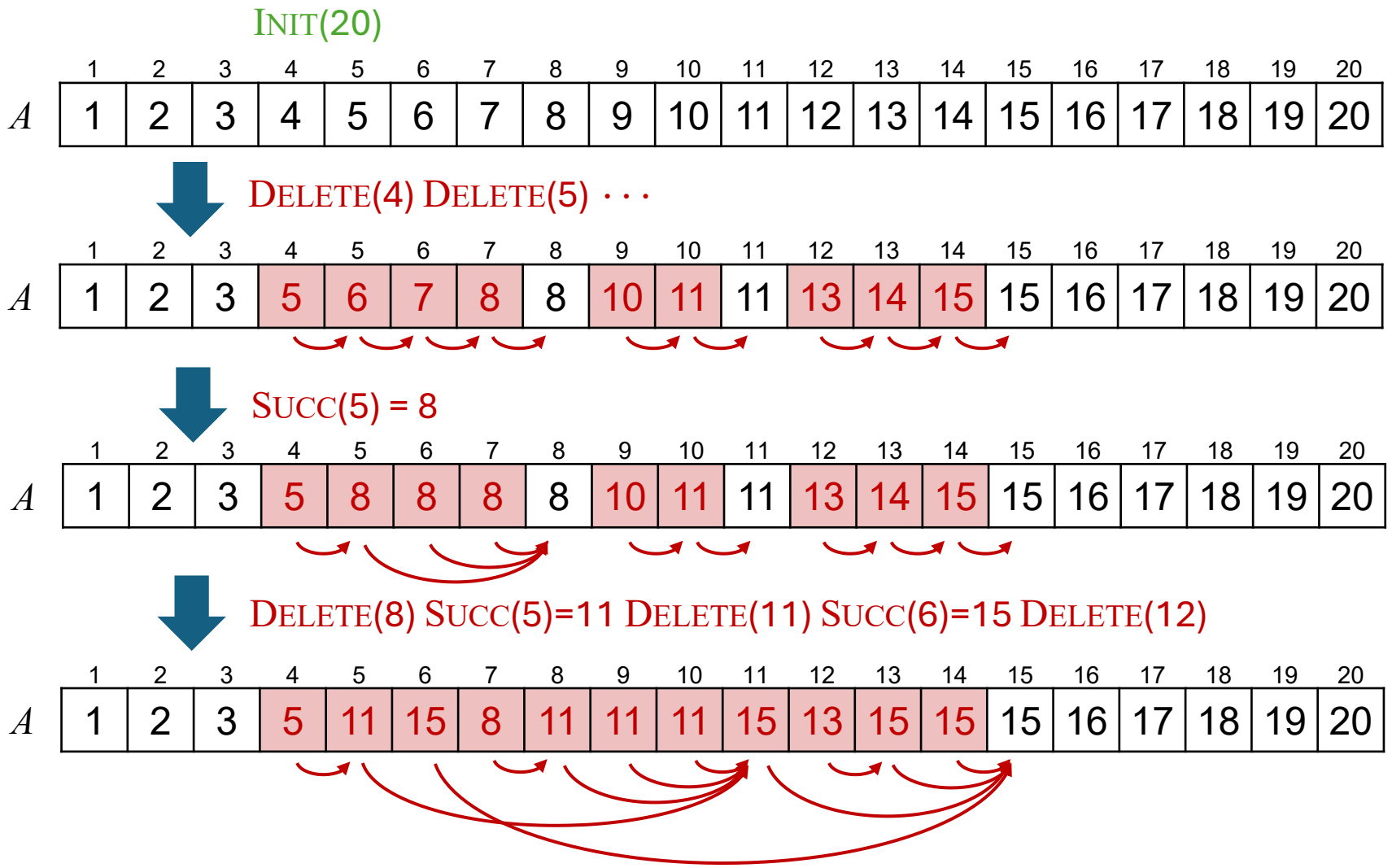
Problem – Internal Union-Find

- Main a subset of $S \subseteq \{1, 2, \ldots, n\}$
- Operations: Init(n) Delete(i) Succ(i)
- [Gabow, Tarjan JCSS 1981] $O(n)$ init, O(1) Succ, amortized $O(1)$ Delete



- **Topic of this talk**
  A very simple data structure (with slightly worse theoretical bounds)

The Data Structure



Proc INIT(n)
  Create $A[1..n]$
  for $i \leftarrow 1$ **to** n **do**
    $A[i] \leftarrow i$

Proc DELETE(i)
  $A[i] \leftarrow i+1$

Proc SUCC(i)
  if $A[i] = i$ **then**
    return i
  $A[i] \leftarrow$ SUCC$(A[i])$
  return $A[i]$

path compression

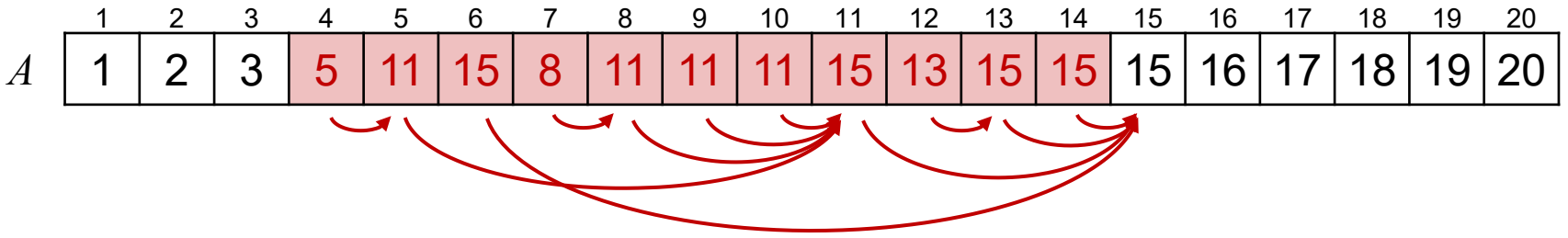
Invariants

- If i not deleted, $A[i] = i$
- If i deleted, $i < A[i] \leq \text{succ}(S, i)$
- (1 and n are never deleted)

Proc INIT(n)
  Create $A[1..n]$
  for $i \leftarrow 1$ **to** n **do**
    $A[i] \leftarrow i$

Proc DELETE(i)
  $A[i] \leftarrow i+1$

Proc SUCC(i)
  if $A[i] = i$ **then**
    return i
  $A[i] \leftarrow$ SUCC$(A[i])$
  return $A[i]$



path compression

Variants

- Deletions can check if value already deleted

```
Proc DELETE(i)
  A[i] ← i + 1
```

```
Proc DELETE(i)
  if A[i] = i then
    A[i] ← i + 1
```

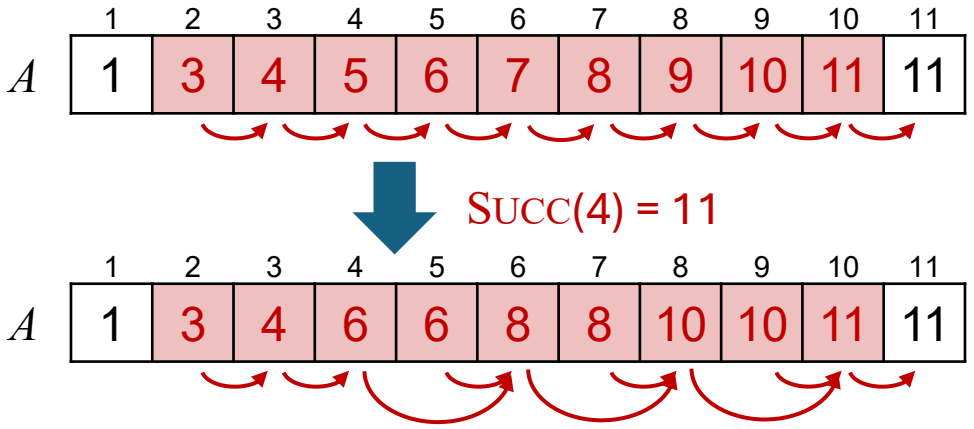
- Path compression can be recursive or 2-pass iterative

```
Proc SUCC(i)
  if A[i] = i then
    return i
  A[i] ← SUCC(A[i])
  return A[i]
```

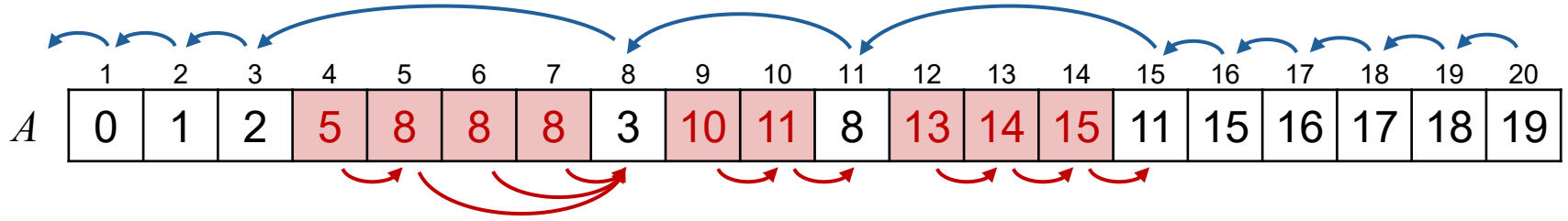
```
Proc SUCC(i)
  j ← i
  while j < A[j] do
    j ← A[j]
  while i < A[i] do
    i' ← A[i]
    A[i] ← j
    i ← i'
  return j
```

Variant – 1-pass Path Halving

```
Proc SUCC(i)
    while A[i] > i do
        A[i] ← A[A[i]]
        i ← A[i]
    return i
```



Supporting Succ, Pred, RangeReport



Invariant If i not deleted, $A[i] = \text{pred}(S, i - 1)$; otherwise, $i < A[i] \leq \text{succ}(S, i)$

Proc INIT(n)
  Create $A[1..n]$
  for $i \leftarrow 1$ **to** n **do**
    $A[i] \leftarrow i - 1$

Proc DELETE(i)
  if $A[i] \leq i$ **then**
    $j \leftarrow$ SUCC$(i + 1)$
    $A[j] \leftarrow A[i]$
    $A[i] \leftarrow j$

Proc PRED(i)
  if $A[i] \leq i$ **then**
    return i
  return $A[$SUCC$(i)]$

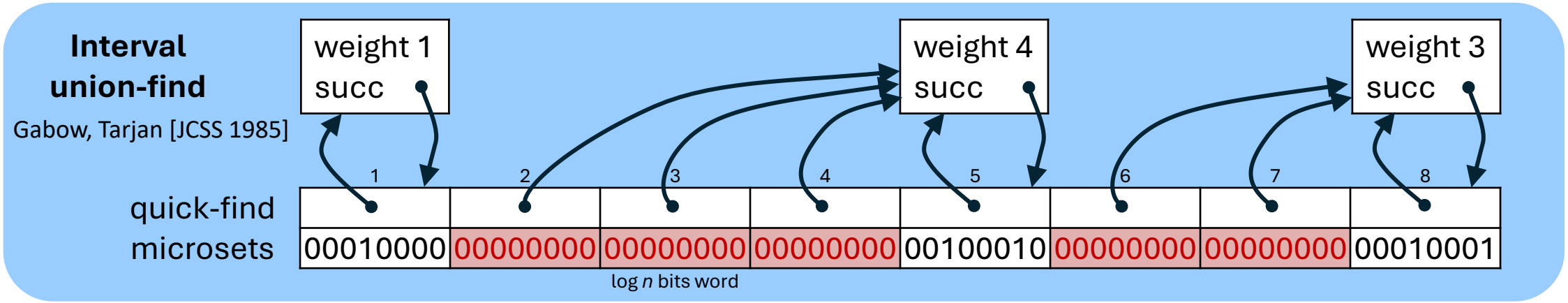
Proc SUCC(i)
  $j \leftarrow i$
  while $j < A[j]$ **do**
    $j \leftarrow A[j]$
  while $i < A[i]$ **do**
    $i' \leftarrow A[i]$
    $A[i] \leftarrow j$
    $i \leftarrow i'$
  return j

Proc RANGEREPORT(i, j)
  $j \leftarrow$ PRED(j)
  while $i \leq j$ **do**
    report j
    $j \leftarrow A[j]$

Previous and New Results

	Successor	Insert	Delete
Binary search trees	$O(\log n)$	$O(\log n)$	$O(\log n)$
van Emde Boas trees	$O(\text{loglog } n)$	$O(\text{loglog } n)$	$O(\text{loglog } n)$
Weighted quick-find	$O(1)$		$O_A(\log n)$
Interval union-find	$O(1)$		$O_A(1)$
This paper*	$O_A(\log n)$		$O(1)$

Theorem* Init(n), d Delete, s Succ operations : $O(n + d + s \cdot (1 + \log_{\max(2,s/n)} \min(s, n)))$



Theoretical Analysis

Theorem: Init(n), d Delete, s Succ operations takes time

$$O(n + d + s \cdot (1 + \log_{\max(2,s/n)} \min(s, n)))$$

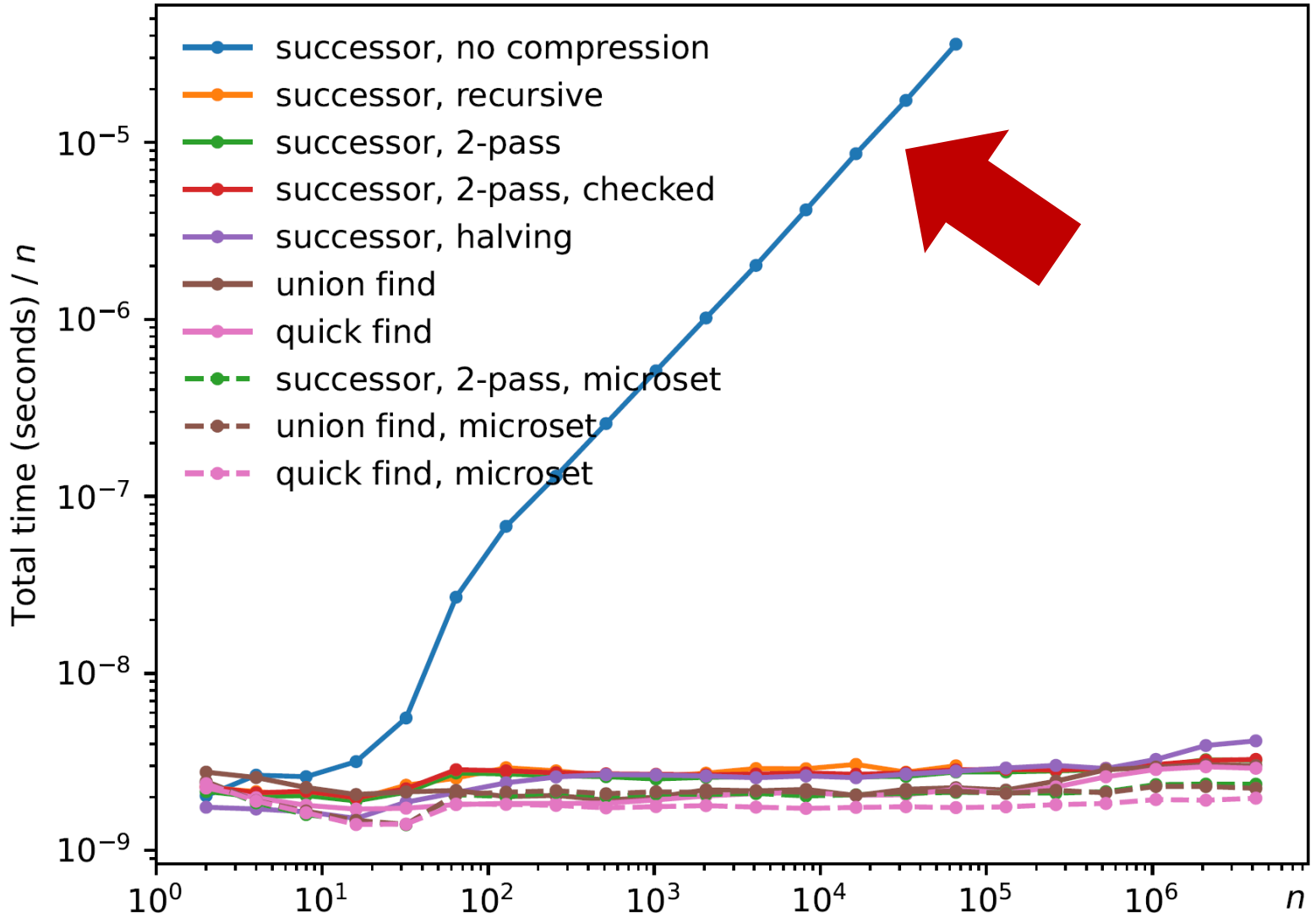
Note: Time is $O(n + d + s)$ if $s = O(n / \log n)$ or $s = \Omega(n^{1+\varepsilon})$

- Our data structure is a special case of Rem's general union-find data structure described by Dijkstra [1976]
- Tarjan, van Leeuwen [1984] analyzed Union-Find with naïve linking

Experimental Results

- Intel Core i7-1365U, Windows 11 (23H2), GCC 14.2.0 (MSYS2), A = 64 bit ints
- Data structures
  1) no path compression
  2) recursive path compression
  3) 2-pass path compression
  4) 2-pass path compression and checked deletions
  5) path halving
  6) union-find (weighted linking, path compression)
  7) weighted quick-find
- Microset-macroset (microset = 64 bits, __builtin_ctzll)
  8) 2-pass path compression
  9) union-find
  10) weighted quick-find (interval union-find)

Bad Idea – Skip Path Compression...



INIT(n)

DELETE(1)

DELETE(2)

⋮

DELETE(n)

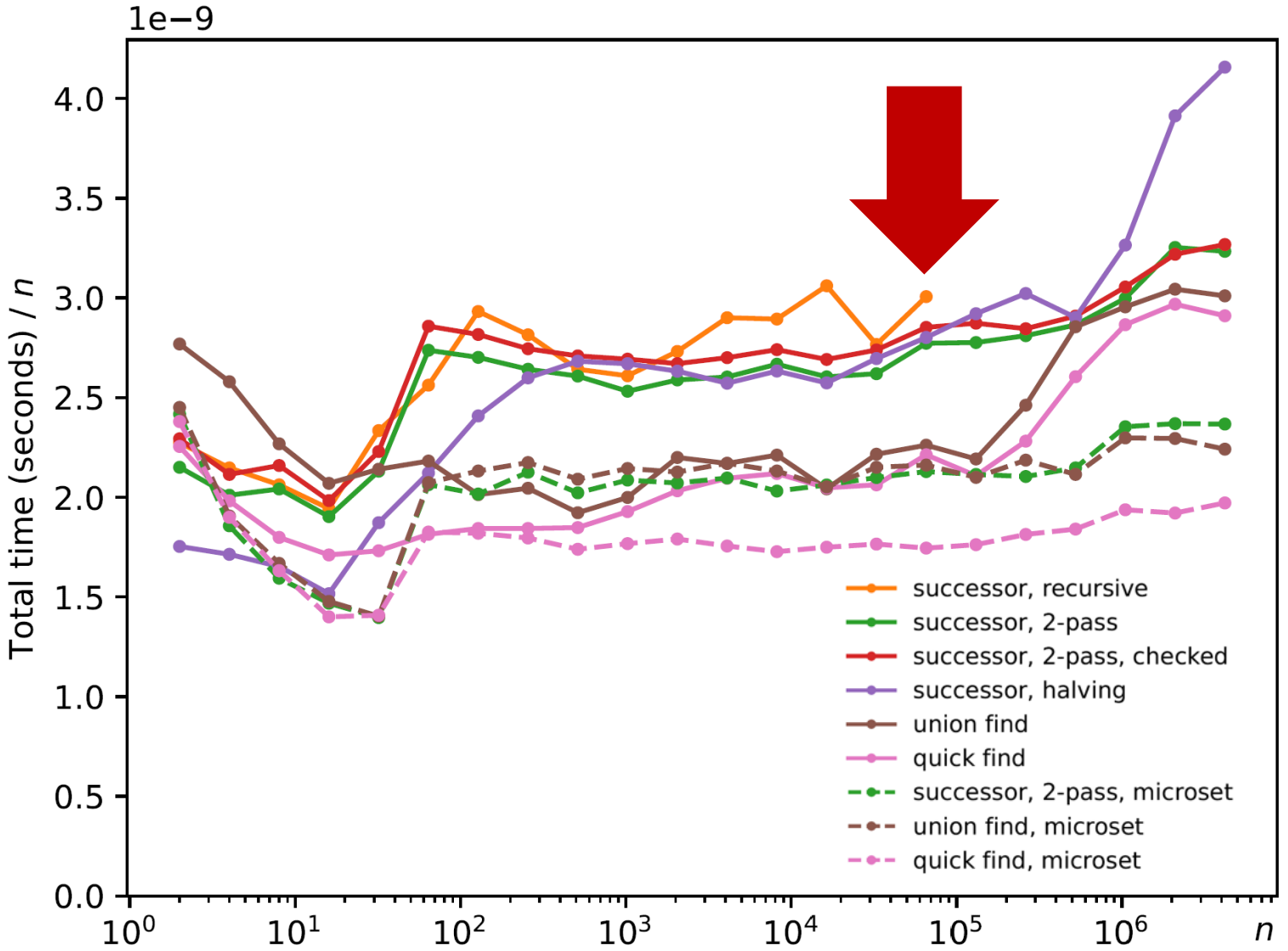
SUCC(1)

SUCC(1)

⋮

SUCC(1)

Recursion – Stack Overflow...



INIT(n)

DELETE(1)

DELETE(2)

⋮

DELETE(n)

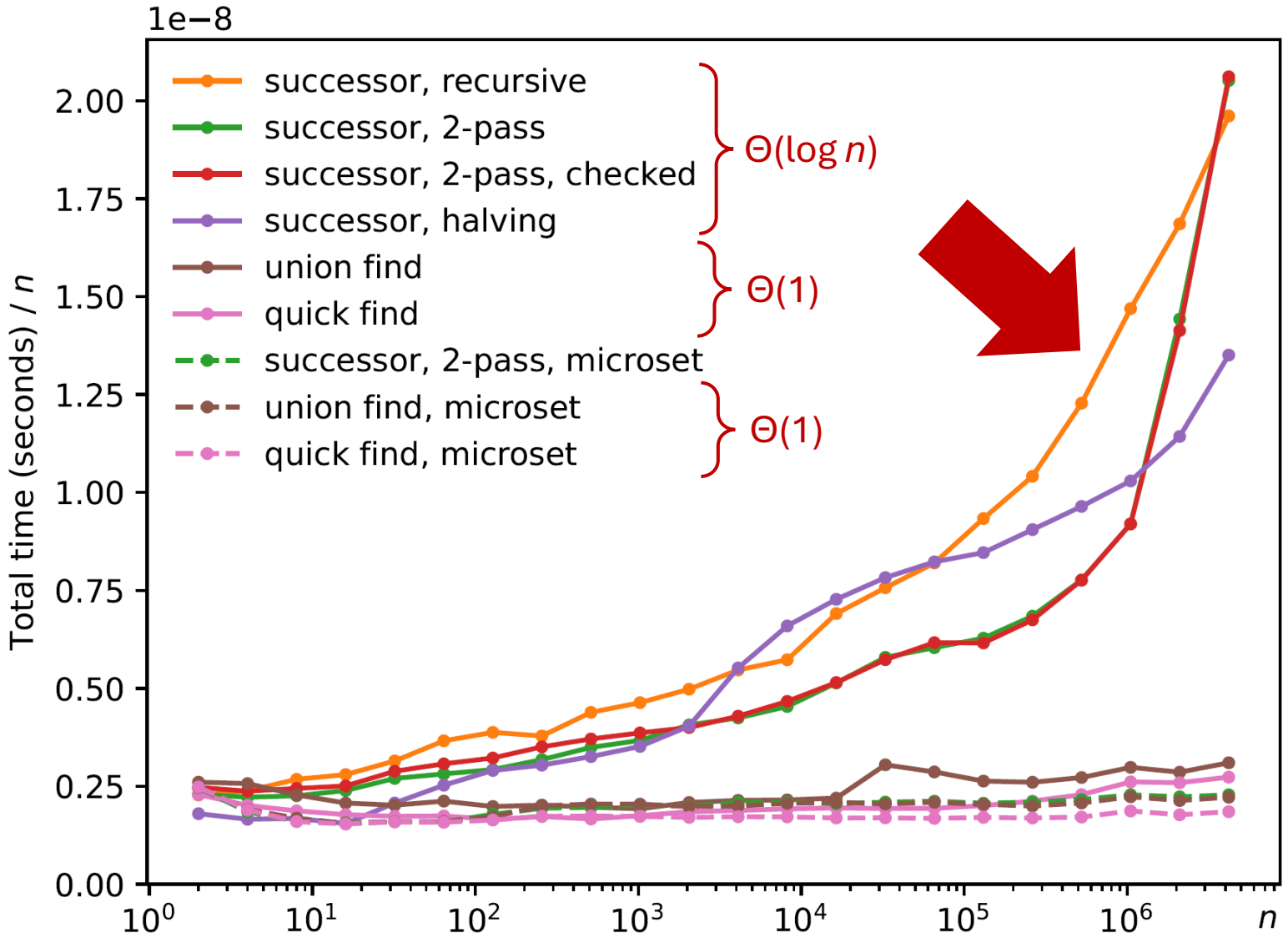
SUCC(1)

SUCC(1)

⋮

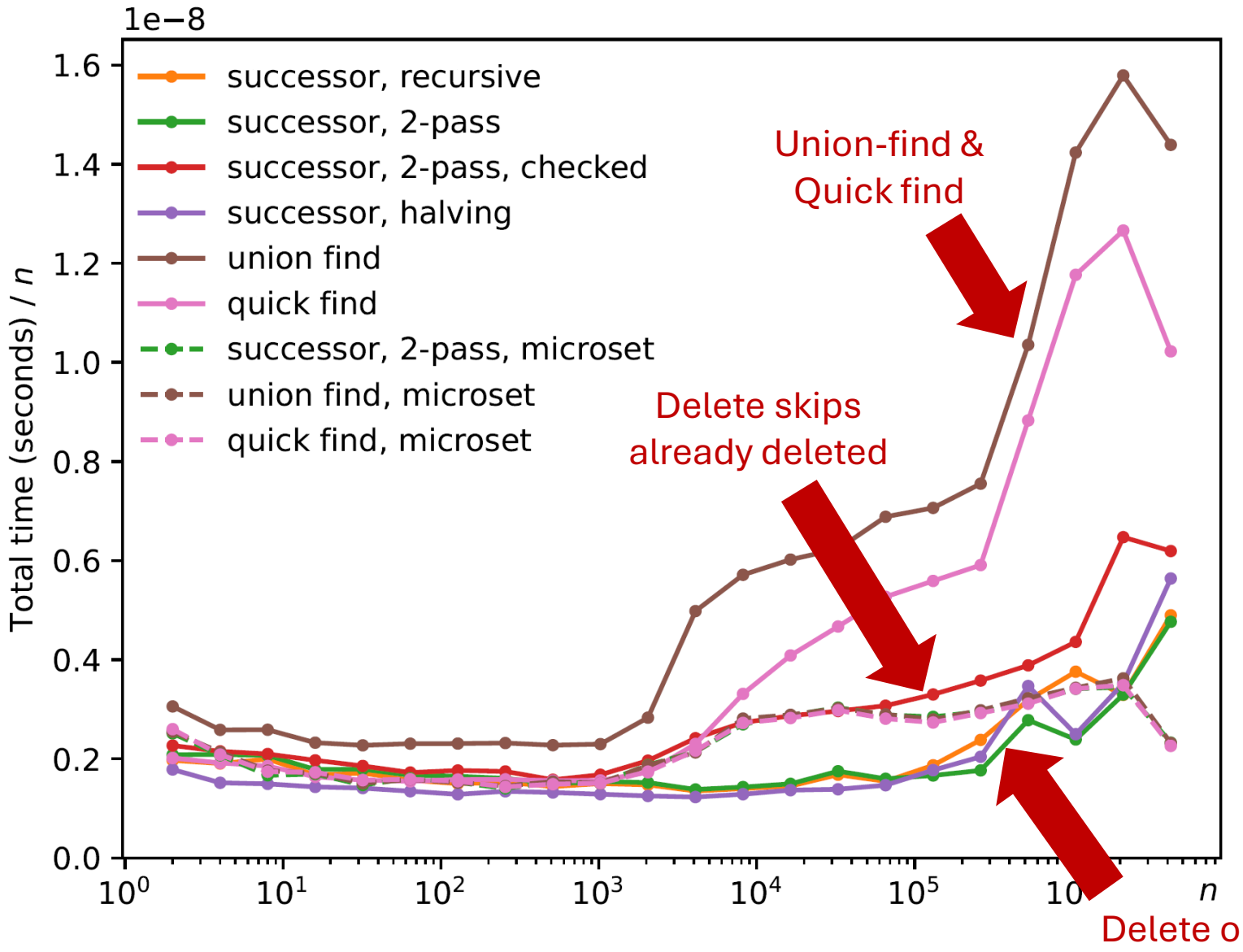
SUCC(1)

Worst-Case Sequence ("add new root")



INIT(*n*)
DELETE(1)
SUCC(*worst*)
DELETE(2)
SUCC(*worst*)
DELETE(3)
SUCC(*worst*)
⋮
DELETE(*n*)
SUCC(*worst*)

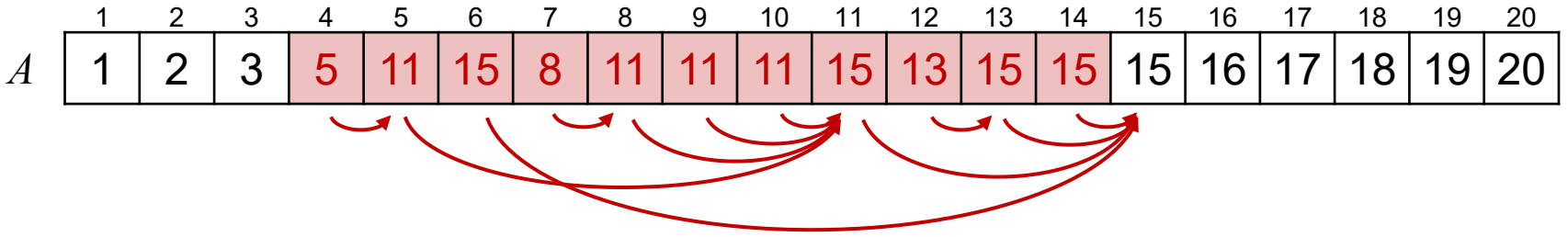
Random Sequence (*n* Delete & Succ)



INIT(*n*)
DELETE(*random*)
SUCC(*worst*)
DELETE(*random*)
SUCC(*worst*)
DELETE(*random*)
SUCC(*worst*)
⋮
DELETE(*random*)
SUCC(*worst*)

Conclusion

- Simple internal union-find data structure
- Theoretical better structures exist
- Performance in same ballpark
- Performs very well with microset-macroset idea



Proc INIT(n)
  Create $A[1..n]$
  for $i \leftarrow 1$ **to** n **do**
    $A[i] \leftarrow i$

Proc DELETE(i)
  $A[i] \leftarrow i+1$

Proc SUCC(i)
  $j \leftarrow i$
  while $j < A[j]$ **do**
    $j \leftarrow A[j]$
  while $i < A[i]$ **do**
    $i' \leftarrow A[i]$
    $A[i] \leftarrow j$
    $i \leftarrow i'$
  return j