Sorting in Linear Time

Alexandre David
B2-206

Linear Sort? But…

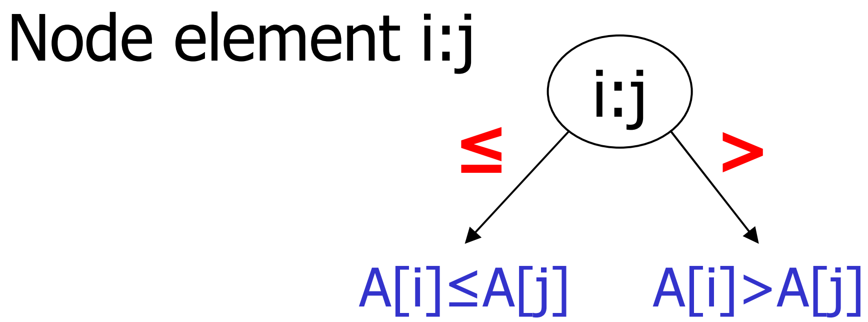
- Best algorithms so far perform in $\Theta(n \lg n)$. But they are comparison sorts.
  - We do not inspect the value or use other information.
  - Only comparisons between keys.
- Comparison sorts need at least $\Omega(n \lg n)$. Previous algorithms were optimal!

Optimality

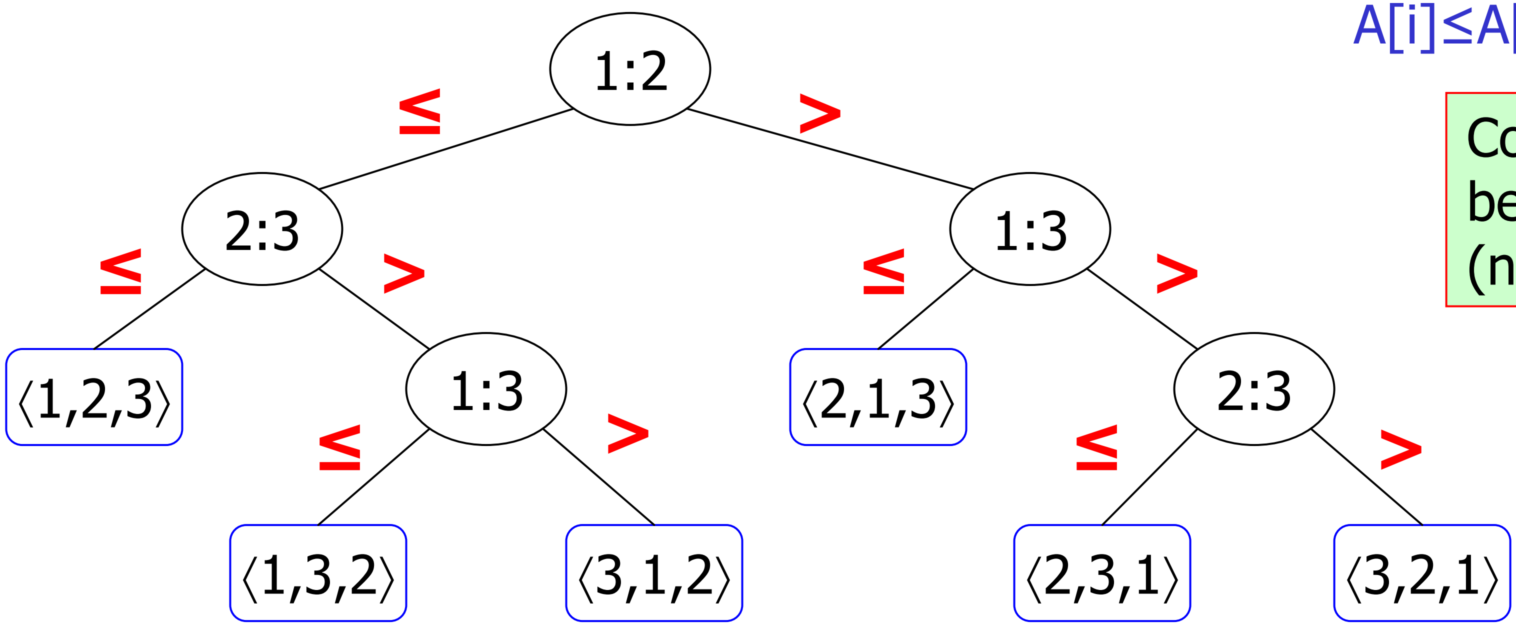
- How to prove that $n \lg n$ is the lower bound for *all possible* comparison sort algorithms?
- Use decision-tree.
  - Binary trees representing comparisons.
  - All possible permutations represented.
  - $n!$ permutations, thus $n!$ leaves.
  - Sorting algorithms find an ordering, i.e., a path.

Decision Tree

"The tree represents the comparisons done by a sorting algorithm."



Comparisons between all (needed) pairs.

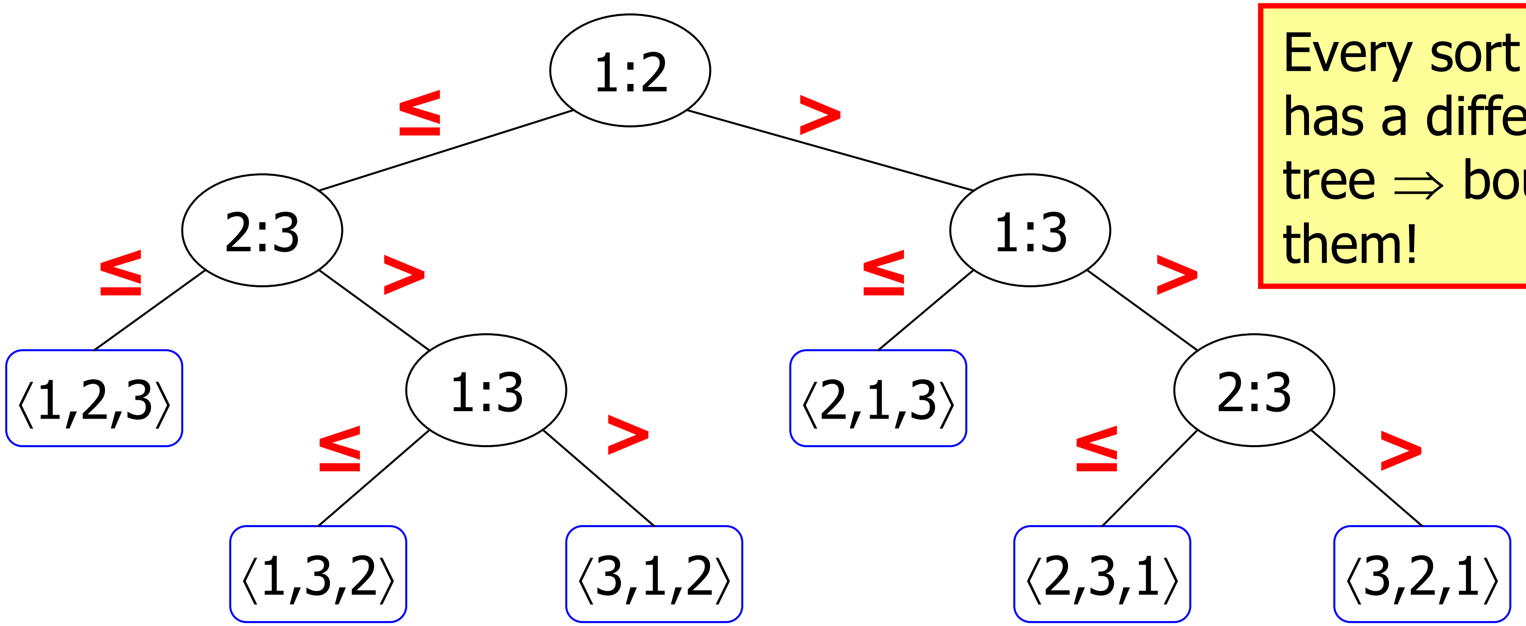


Decision Tree

The point: Lower bound on the heights of all decision trees = lower bound of running time of any comparison sort algorithm.

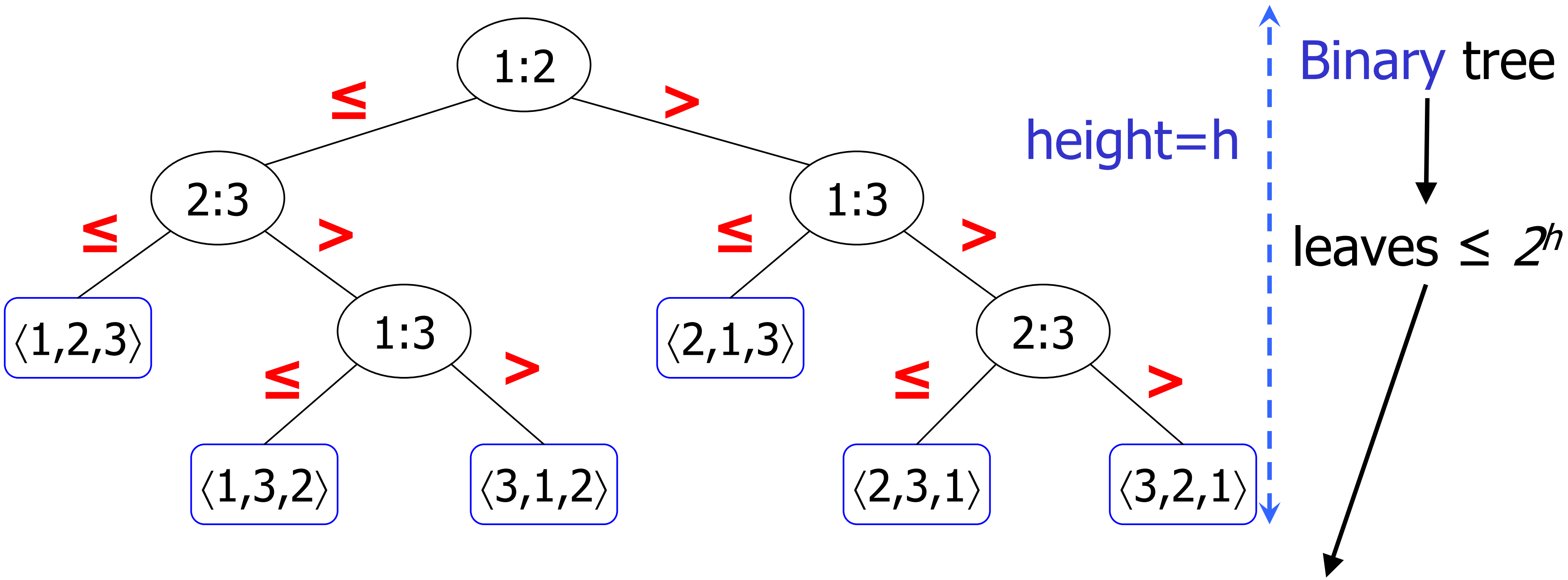
Because to find one ordering one must go on a path from the root to a leaf.

Every sort algorithm has a different decision tree ⇒ bound all of them!



Bound on the height.

Decision Tree



Leaves=permutations ⟶ $n!$ = leaves ⟶ $h \geq \lg(n!) = \Omega(n \lg n)$

(Any correct algorithm must be able to produce any permutation)

(3.18)

Optimality

- Conclusion:
  - Any correct algorithm must go through $\Omega(n \lg n)$ to produce any ordering.
  - We have sorting algorithms that have a bound of $O(n \lg n)$.
  - These (comparison) sorting algorithm are optimal!
  - You can't do better. If you do better, then your algorithm cannot generate all the permutations and is incorrect.

Counting Sort

- Assume that the input is made of integers with a small range (k):
  - $0 \le a_{1..n} \le k$.
  - When $k = O(n)$, counting-sort runs in $\Theta(n)$.
- Idea: For every x
  - count how many elements are ≤x, say *t*,
  - put x at position *t*, the *right* position.

Counting Sort

```
Count_sort(A,B,k):                          // B = output
for i = 0 to k do C[i] = 0                  // initialize
for i = 1 to length(A) do C[A[i]]++         // count x_i
for i = 1 to k do C[i] += C[i-1]            // count ≤x_i
for i = length(A) downto 1 do
    B[C[A[i]]] = A[i]                       // write x at t
    C[A[i]]--                               // update counter
done
```

Needs extra memory with k elements: C[0..k] for counting.

Counting Sort

```
Count_sort(A,B,k):
1 for i = 0 to k do C[i] = 0
2 for i = 1 to length(A) do C[A[i]]++
3 for i = 1 to k do C[i] += C[i-1]
4 for i = length(A) downto 1 do
      B[C[A[i]]] = A[i]
      C[A[i]]--
done
```

	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

	0	1	2	3	4	5	
C:	0	0	0	0	0	0	1
C:	2	0	2	3	0	1	2
C:	2	2	4	7	7	8	3
C:	2	2	4	7	7	8	4
C:	2	2	4	6	7	8	
C:	1	2	4	6	7	8	
C:	1	2	4	5	7	8	

A[8] 3 at C[3] 7 — one less 3: C[3]--

A[7] 0 at C[0] 2 — one less 0: C[0]--

A[6] 3 at C[3] 6 — one less 3: C[3]--

	1	2	3	4	5	6	7	8
B:		0				3	3	

Counting Sort

```
Count_sort(A,B,k):
for i = 0 to k do C[i] = 0
for i = 1 to n do C[A[i]]++
for i = 1 to k do C[i] += C[i-1]
for i = n downto 1 do
    B[C[A[i]]] = A[i]
    C[A[i]]--
done
```

- Running time $T(n)=\Theta(n+k)$.
  For $k=O(n)$, $T(n)=\Theta(n)$.
- There is no comparison.
- The sort is stable (order kept for $a_i==a_j$).
- The sort is not in-place.
- Problem: Range of numbers translates into the size of the working array (counters).

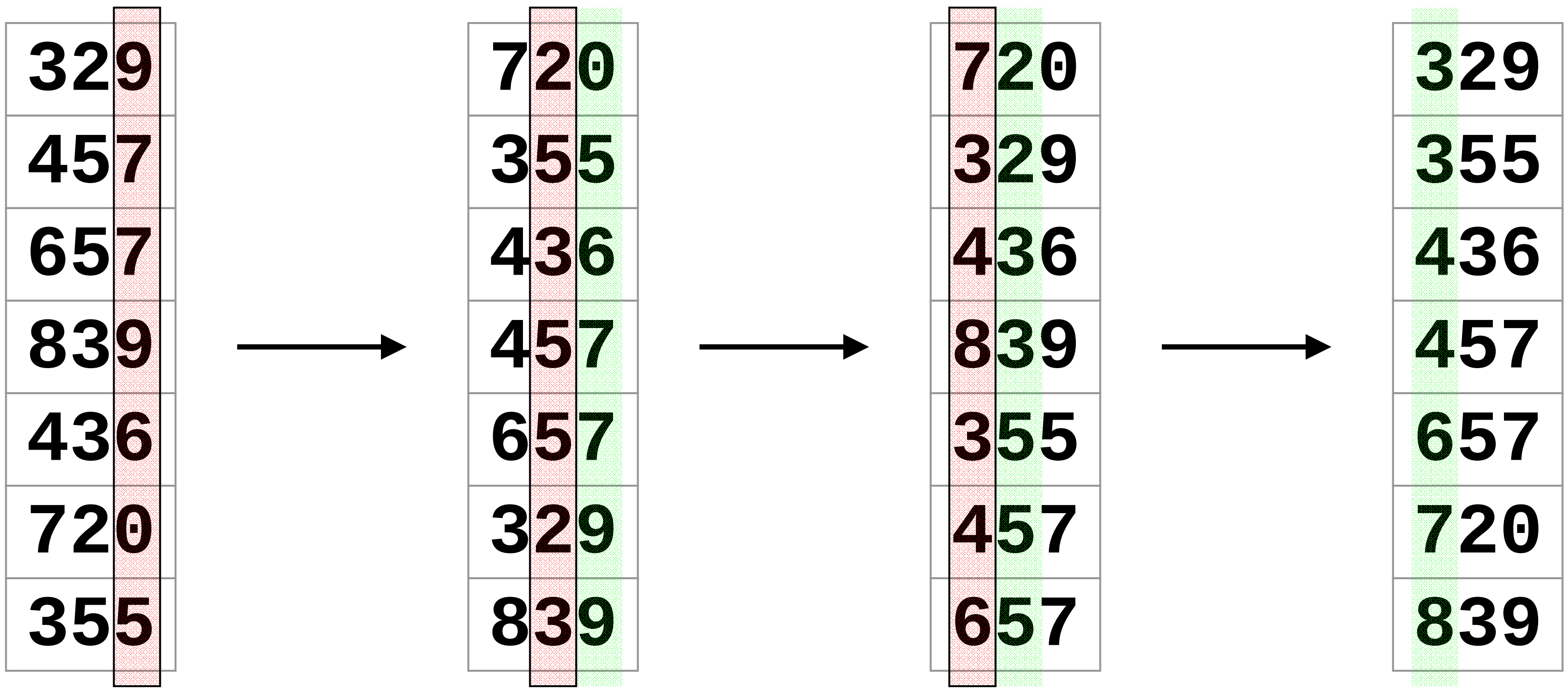
Radix Sort

- Sort on the digits of the numbers.
  - Least significant digits first.
  - Use a **stable** sort (like counting sort).

```
Radix_sort(A,d):
for i = 1 to d do
    stable_sort A with keys=digit i
done
```

- Sort n-digits numbers with each digit taking k values (base k), running time is $T(n)=\Theta(d(n+k))$.

Radix Sort


329
457
657
839
436
720
355
720
355
436
457
657
329
839
720
329
436
839
355
457
657
329
355
436
457
657
720
839

Bucket Sort

- Assume the input is uniformly distributed.
- Assume values in [0,1).

```
Bucket_sort(A):
n = length(A)
for i = 1 to n do insert A[i] into list B[n*A[i]]
for i = 0 to n-1 do insertion_sort list B[i]
concatenate lists B[0],B[1],…B[n-1]
```

- Running time: $T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$
- Expected running time: *Θ(n)*.

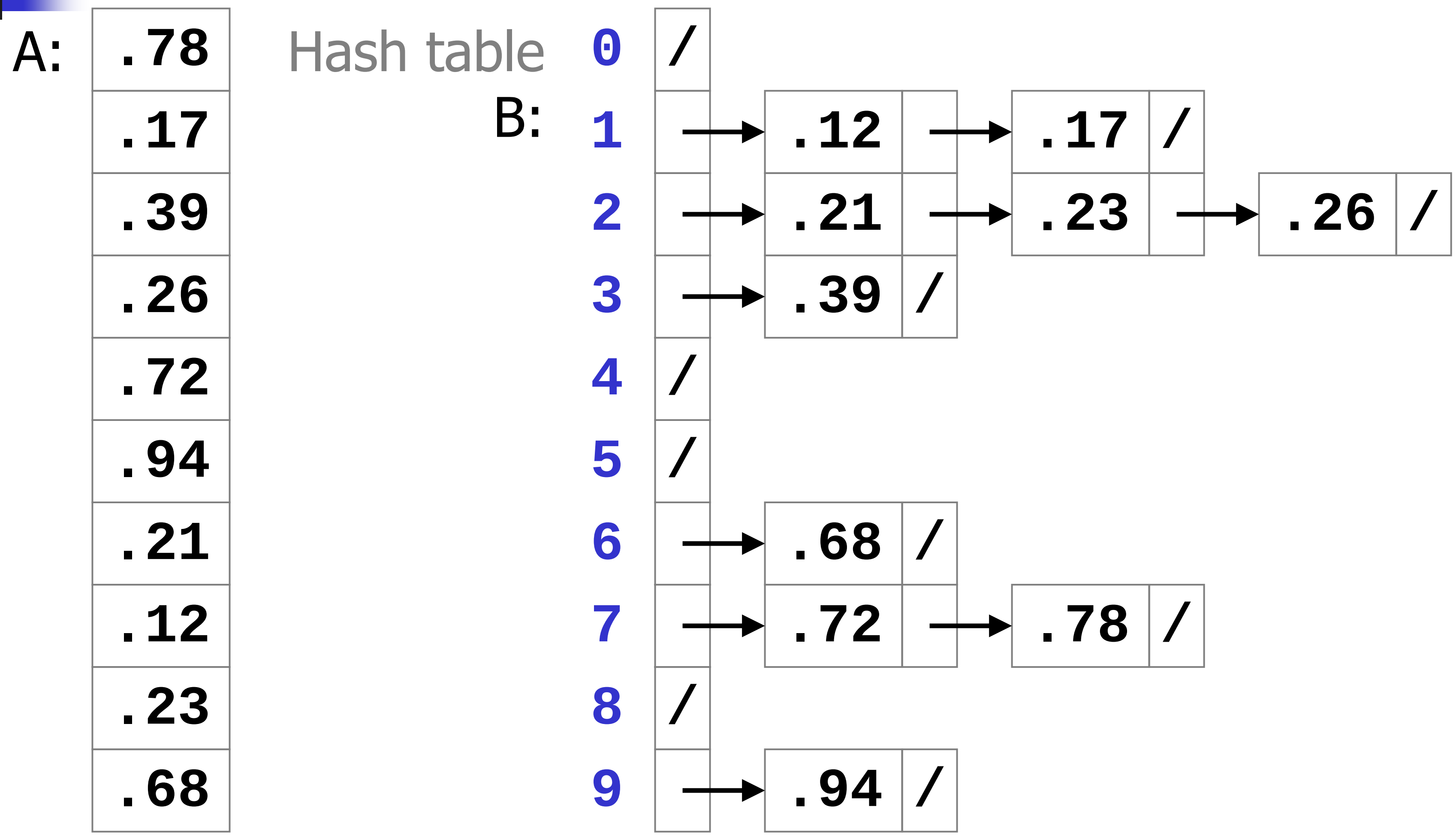
Bucket Sort

- Expected time:
  - Use $E[n_i^2]=2-1/n$.
    See book for technicality.

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) = \Theta(n)$$

  - Idea: *n* elements distributed uniformly in *n* entries $\Rightarrow$ 1 element per entry in average.
    But not *O(1)* for sorting…

Bucket Sort


A:
.78
.17
.39
.26
.72
.94
.21
.12
.23
.68
Hash table
B:
0 /
1 .12 .17 /
2 .21 .23 .26 /
3 .39 /
4 /
5 /
6 .68 /
7 .72 .78 /
8 /
9 .94 /