Randomization

Quicksort

Sorting

Input:
An array $A[n]$ of n integers.

Output:
An array $B[n]$ such that B contains the entries of $A[n]$ in an ascending order.

Sorting

Input:
An array $A[n]$ of n integers.

For simplicity, distinct values.

Output:
An array $B[n]$ such that B contains the entries of $A[n]$ in an ascending order.

We don't actually need a new array.

Outline

- Quicksort
  - Pivot
  - Paranoid

- Runtime
  - $O(n \log n)$ in expectation
  - With high probability in the tutorial exercises

Learning objectives:
You are able to
- describe the paranoid quicksort algorithm
- bound of pivot candidates per recursive call in expectation

Quicksort

- Works in expected $O(n \log n)$ time

Mergesort is also
$O(n \log n)$!?

Quicksort

- Works in expected $O(n \log n)$ time
  - In practice, tends to be faster than Mergesort
  - Behaves well with caching
  - can be implemented without extra space

- Can also be shown to work in time $O(n \log n)$ with high probability
  - We will discuss a simpler $O(n \log^2 n)$ analysis in the tutorial session

Paranoid Quicksort

Classic Quicksort:

1. Pick a pivot p randomly.
2. Elements smaller than p to the left and greater to the right.
3. Recurse.

Paranoid Quicksort

Classic Quicksort:
1. Pick a pivot p randomly.
2. Elements smaller than p to the left and greater to the right.
3. Recurse.

Simpler to analyse

Paranoid Quicksort:
1. Pick a pivot p randomly.
2. If less than $1/10$ elements smaller or larger, pick a new pivot.
3. Elements smaller than p to the left and greater to the right.
4. Recurse.

No split is too bad

Paranoid Quicksort

Quicksort(A, ℓ, r)**:**
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \dots, r$

Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p-1)$
Quicksort$(A, p+1, r)$

Pivot p is *good* if at least $|A|/10$ elements of A are larger and smaller than p.

Any pivot is good if $|A| < 10$.

Paranoid Quicksort

Quicksort(A, ℓ, r):
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \dots, r$

Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p-1)$
Quicksort$(A, p+1, r)$

Pivot p is *good* if at least $|A|/10$ elements of A are larger and smaller than p.

Any pivot is good if $|A| < 10$.

Paranoid Quicksort

Quicksort(A, ℓ, r)**:**
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \ldots, r$

Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p-1)$
Quicksort$(A, p+1, r)$

Pivot p is *good* if at least $|A|/10$ elements of A are larger and smaller than p.

Any pivot is good if $|A| < 10$.

Paranoid Quicksort

Quicksort(A, ℓ, r)**:**
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \ldots, r$

Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p-1)$
Quicksort$(A, p+1, r)$

Pivot p is *good* if at least $|A|/10$ elements of A are as large and as small as p.

Any pivot is good if $|A| < 10$.

Only touch the relevant entries.

Paranoid Quicksort

Quicksort(A, ℓ, r)**:**
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \dots, r$

Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p - 1)$
Quicksort$(A, p + 1, r)$

Let $|A| = n$.
Pivot p is *good* if at least n/10 elements of A are at least as large and small as p.

Since elements are distinct, at least $8n/10$ elements are good pivots.

$P(p \text{ is a good pivot}) \geq 8/10$

Paranoid Quicksort

10	15	5	0	11	65	4	2	9
	i j							

Partition(A, p)**:**
Swap($A[0], A[p]$)
$i \coloneqq 1$
for($j = i, \ldots$)
  if($A[j] < A[0]$)
    Swap($A[i], A[j]$);
    $i \coloneqq i + 1$
Swap($A[i-1], A[0]$)

Paranoid Quicksort

10	15	5	0	11	65	4	2	9
	$i\ j$							

Partition(A, p):
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
  if$(A[j] < A[0])$
    Swap$(A[i], A[j])$;
    $i := i + 1$
Swap$(A[i-1], A[0])$

Paranoid Quicksort

10	15	5	0	11	65	4	2	9
	i j							

Partition(A, p):
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
  if$(A[j] < A[0])$
    Swap$(A[i], A[j])$;
    $i := i + 1$
Swap$(A[i-1], A[0])$

Paranoid Quicksort

10	15	5	0	11	65	4	2	9
	i j							

Partition(A, p):
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
 if$(A[j] < A[0])$
  Swap$(A[i], A[j])$;
  $i := i + 1$
Swap$(A[i-1], A[0])$

Paranoid Quicksort

10	15	5	0	11	65	4	2	9
	i j							

10	15	5	0	11	65	4	2	9
	i	j						

Partition(A, p)**:**
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
 if$(A[j] < A[0])$
  Swap$(A[i], A[j])$;
  $i := i + 1$
Swap$(A[i-1], A[0])$

Paranoid Quicksort

Partition(A, p)**:**
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
  if$(A[j] < A[0])$
    Swap$(A[i], A[j])$;
    $i := i + 1$
Swap$(A[i-1], A[0])$

10	15	5	0	11	65	4	2	9
	i j							

10	15	5	0	11	65	4	2	9
	i	j						

10	5	15	0	11	65	4	2	9
		i j						

Paranoid Quicksort

Partition(A, p):
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
  if$(A[j] < A[0])$
    Swap$(A[i], A[j])$;
    $i := i + 1$
Swap$(A[i-1], A[0])$

10	15	5	0	11	65	4	2	9
	$i\ j$							

10	15	5	0	11	65	4	2	9
	i	j						

10	5	15	0	11	65	4	2	9
		$i\ j$						

10	5	0	15	11	65	4	2	9
			i			j		

Paranoid Quicksort

Partition(A, p):
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
  if$(A[j] < A[0])$
    Swap$(A[i], A[j])$;
    $i := i + 1$
Swap$(A[i-1], A[0])$

10	15	5	0	11	65	4	2	9
	i j							

10	15	5	0	11	65	4	2	9
	i	j						

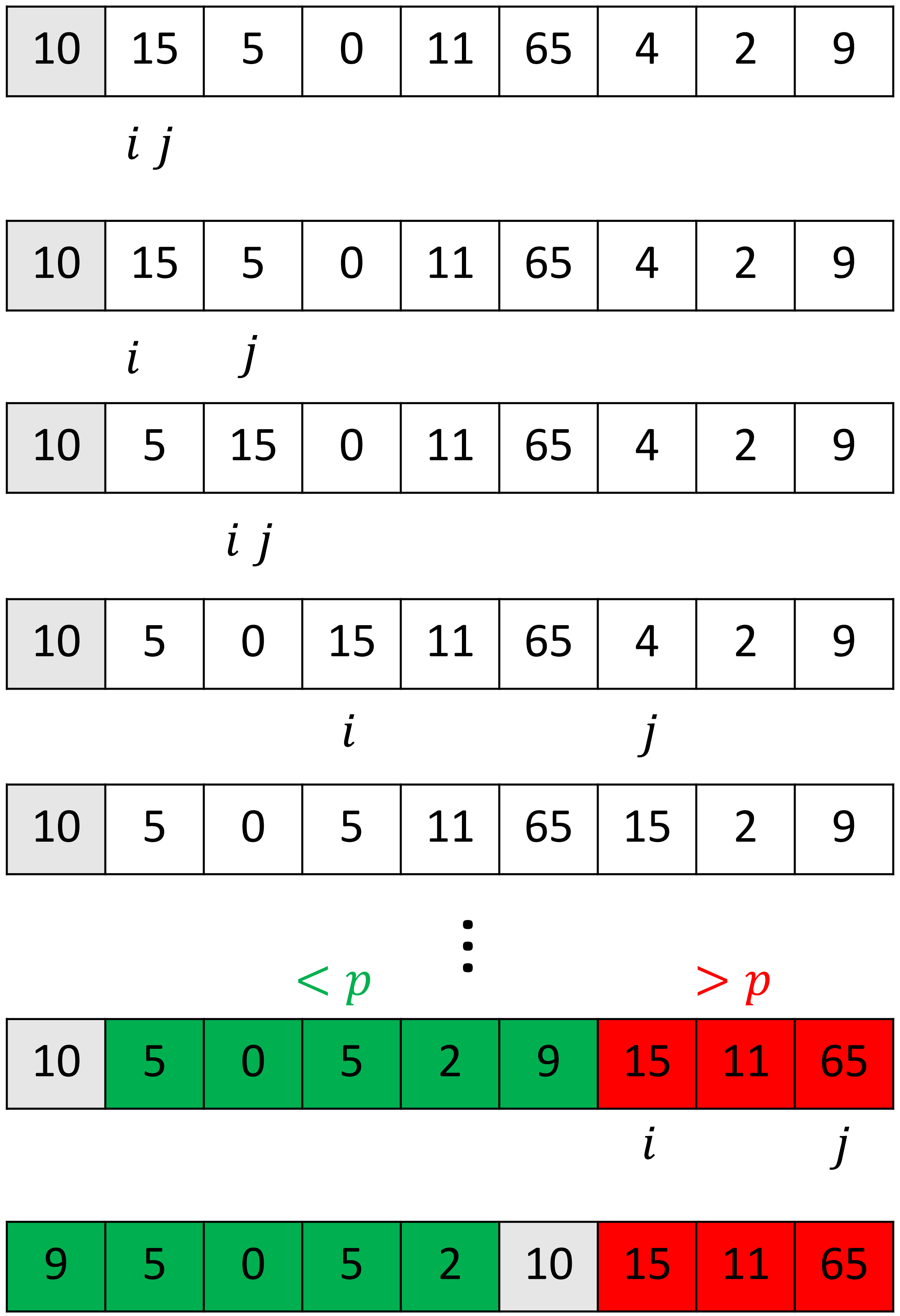
10	5	15	0	11	65	4	2	9
		i j						

10	5	0	15	11	65	4	2	9
			i			j		

10	5	0	5	11	65	15	2	9
				i		j		

Paranoid Quicksort

Partition(A, p):
Swap$(A[0], A[p])$
$i := 1$
for$(j = i, \ldots)$
  if$(A[j] < A[0])$
    Swap$(A[i], A[j])$;
    $i := i + 1$
Swap$(A[i-1], A[0])$



Paranoid Quicksort

Partition(A, p)**:**
Swap($A[0], A[p]$)
$i \coloneqq 1$
for($j = i, \ldots$)
  if($A[j] < A[0]$)
    Swap($A[i], A[j]$);
    $i \coloneqq i + 1$
Swap($A[i-1], A[0]$)

Observation:
Let $|A| = n$. Partition takes $O(n)$ time

Paranoid Quicksort

Correctness:
Almost 1-to-1 the same as with Mergesort

Outline

- Quicksort
  - Pivot
  - Paranoid

- Runtime
  - Expectation
  - With high probability in the tutorial exercises

Paranoid Quicksort - Runtime

Recurrence:
$T(n) = T(n-i) + T(i) + R(n)$

Cost of partitioning

Random variables

Quicksort(A, ℓ, r)**:**
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \ldots, r$

Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p-1)$
Quicksort$(A, p+1, r)$

Paranoid Quicksort - Runtime

Recurrence:

$T(n) = T(n-i) + T(i) + R(n)$

Random variables

Cost of partitioning

Quicksort(A, ℓ, r)**:**
If$(\ell \geq r)$
  Return

Until(p is *good*)
  Choose random $p \in \ell, \ldots, r$

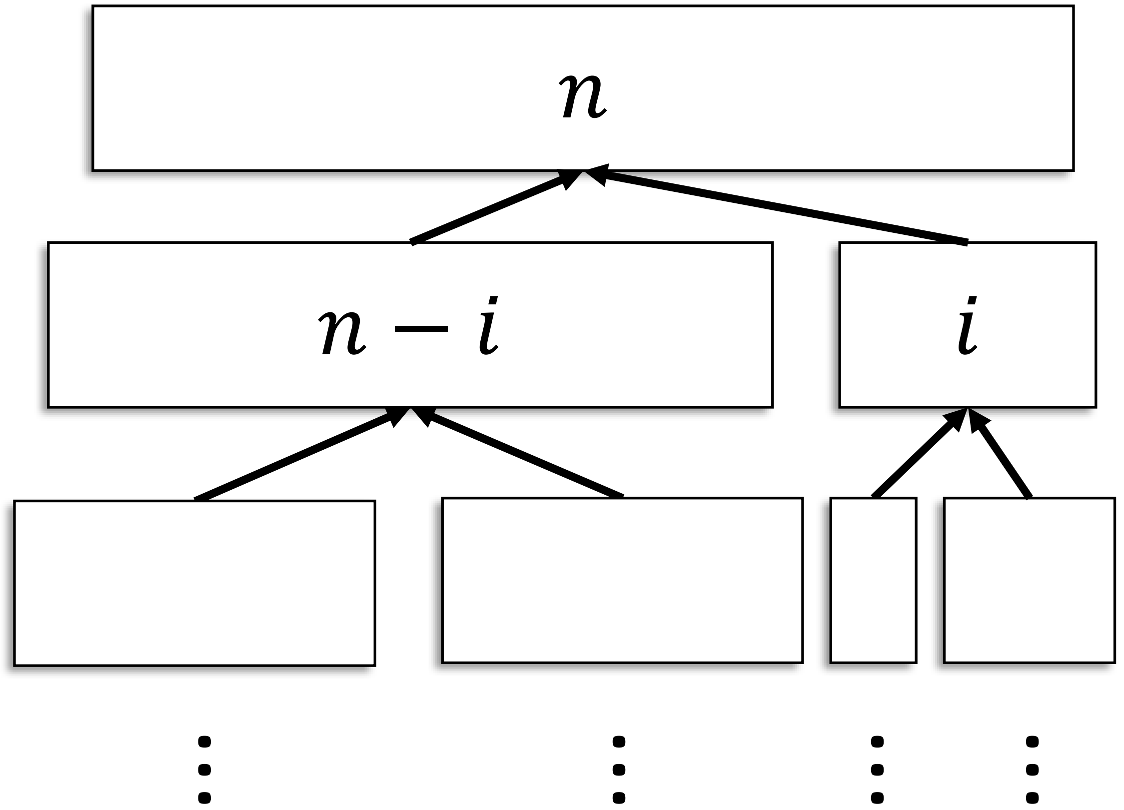
Partition$(A[\ell, r], p)$
Quicksort$(A, \ell, p-1)$
Quicksort$(A, p+1, r)$

Paranoid Quicksort - Runtime

Recurrence:
$T(n) = T(n-i) + T(i) + R(n)$

Cost of partitioning

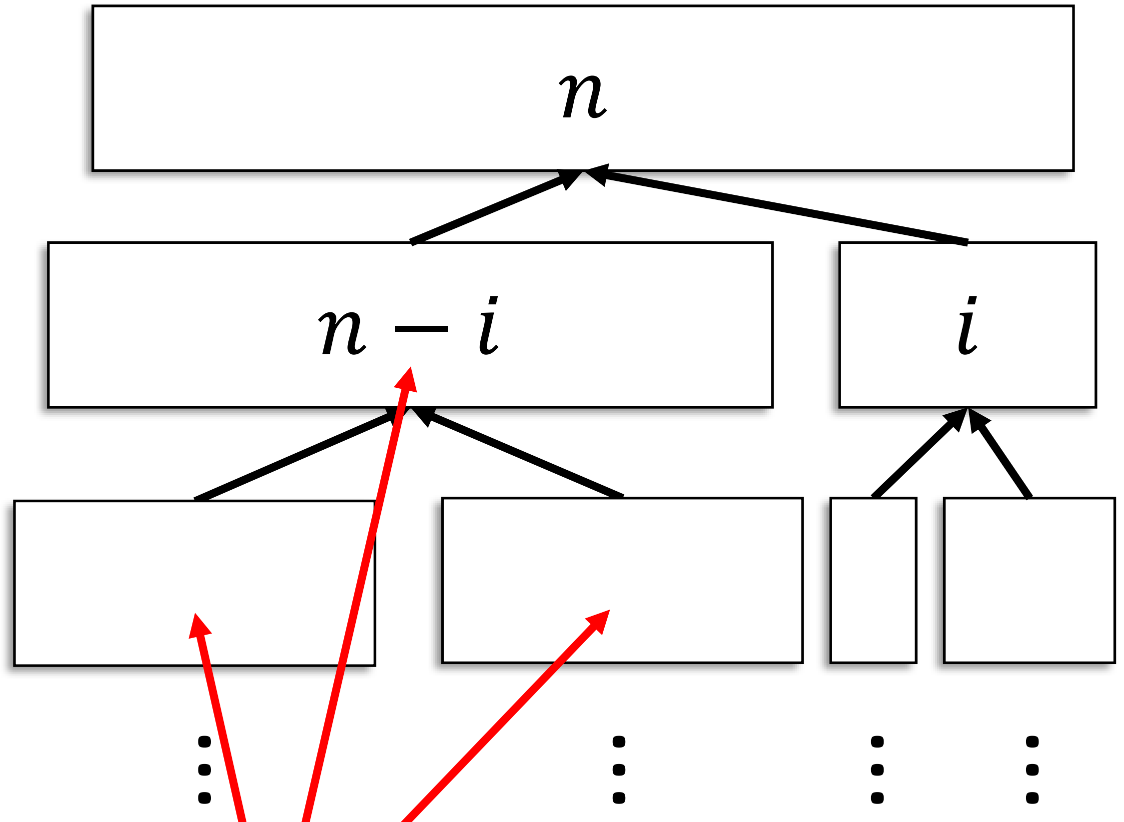
Random variables



Paranoid Quicksort - Runtime

Recurrence:
$T(n) = T(n-i) + T(i) + R(n)$

Random variables
Depend on each other



Dependencies:
The cost of j:th level depends on $j - 1$

Runtime

Paranoia:
The array $A[n]$ is always split at least $\frac{1}{10} : \frac{9}{10}$

$\longrightarrow$

Recursion tree depth is $O(\log n)$

Runtime

Paranoia:
The array $A[n]$ is always split at least $\frac{1}{10} : \frac{9}{10}$

Recursion tree depth is $O(\log n)$

Linearity of Expectation:

$$E[T(n)] = \max_i \{T(n-i) + T(i)\} + E[\#\text{partitions}] \cdot cn$$

Runtime

Paranoia:
The array $A[n]$ is always split at least $\frac{1}{10}:\frac{9}{10}$

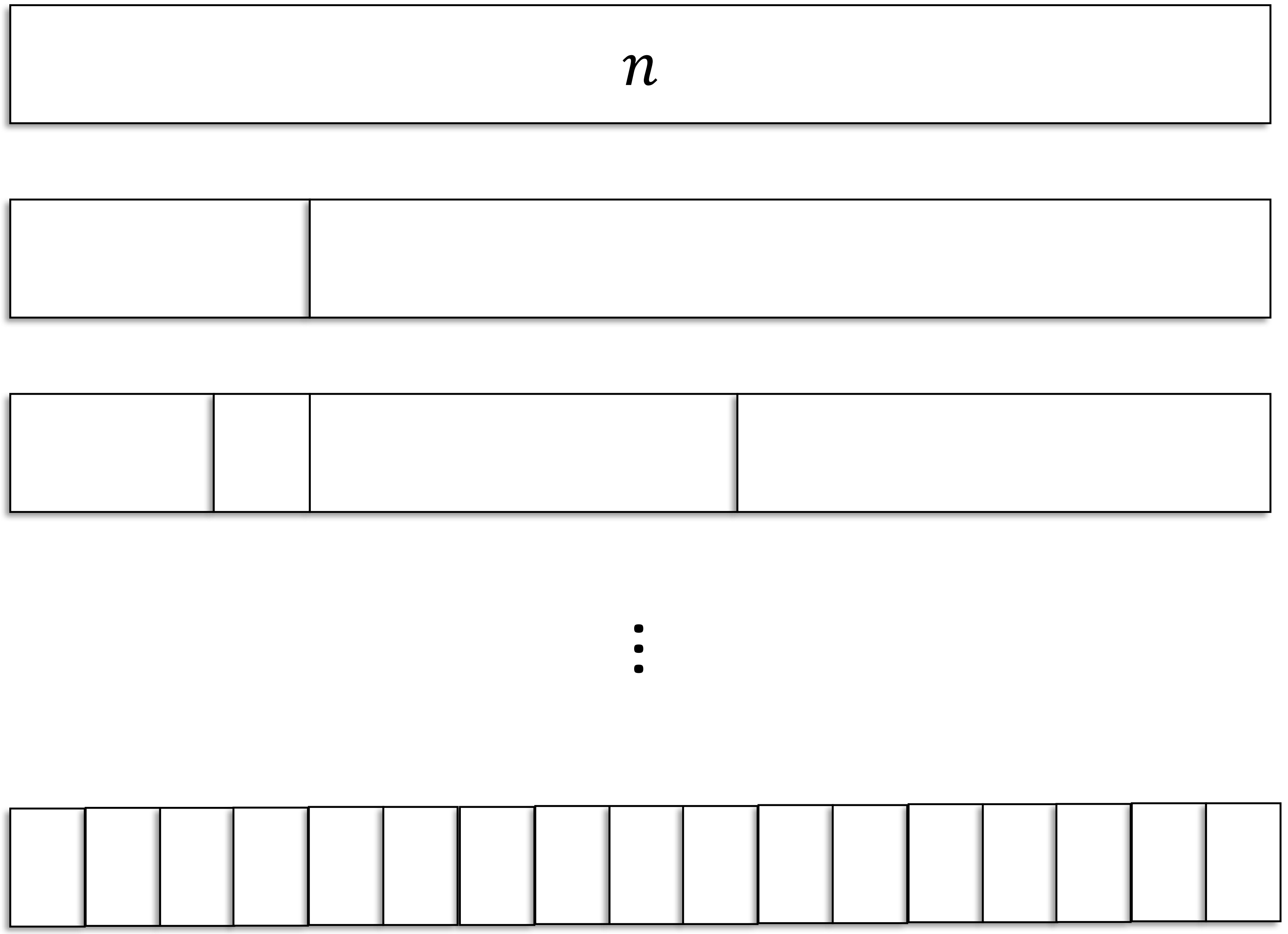
Recursion tree depth is $O(\log n)$

Linearity of Expectation:
$E[T(n)] = \max_i \{T(n-i) + T(i)\} +$
$E[\text{\#partitions}] \cdot cn$

Figure out the worst case (expected) cost per level.

Runtime

Cost of **Partition** on k elements is $O(k)$



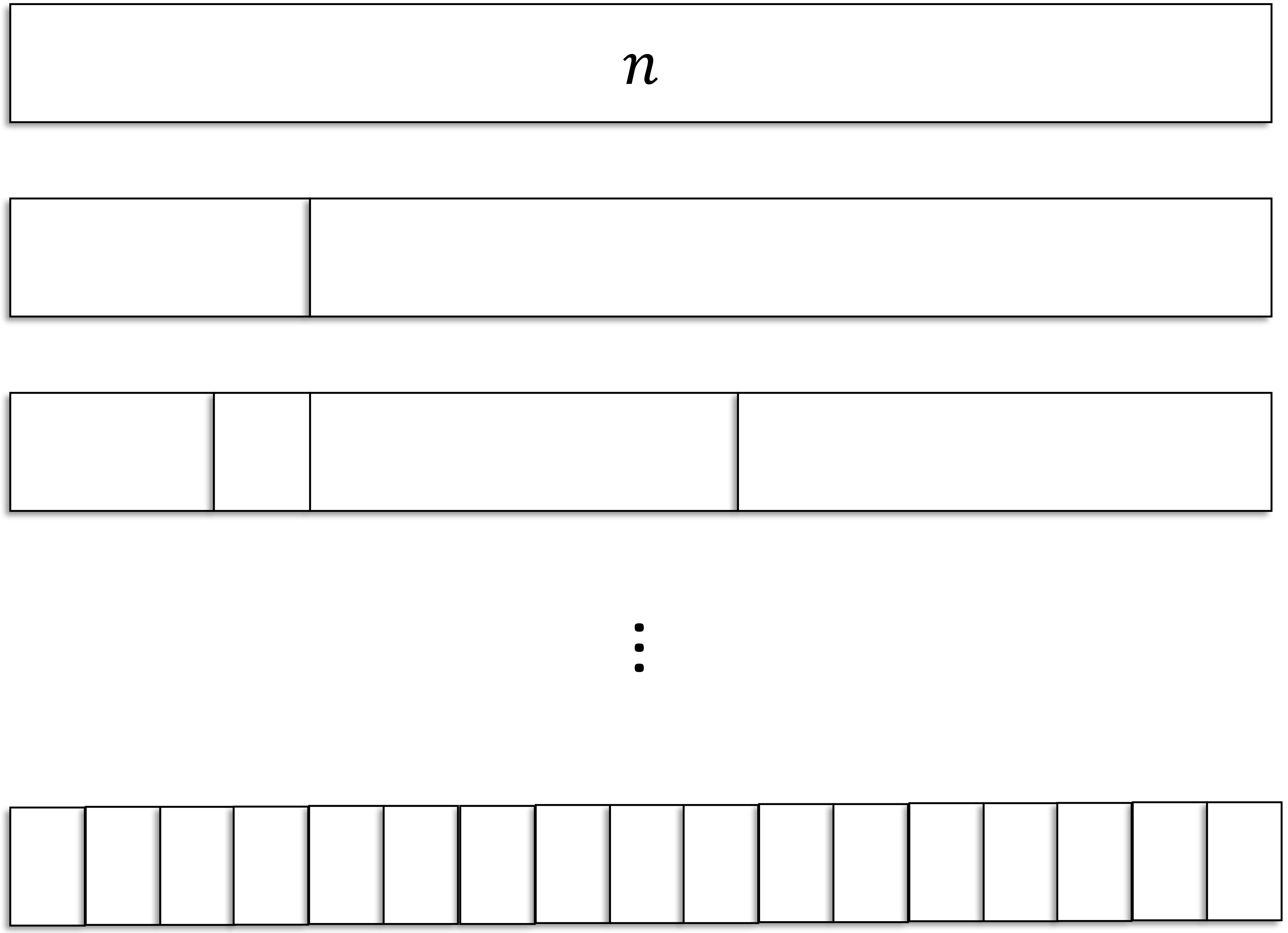
Runtime

$P(\text{good partition}) \geq 4/5$

$\downarrow$

$E[\#\text{partitions}] \leq 2$

Cost of **Partition** on k elements is $O(k)$



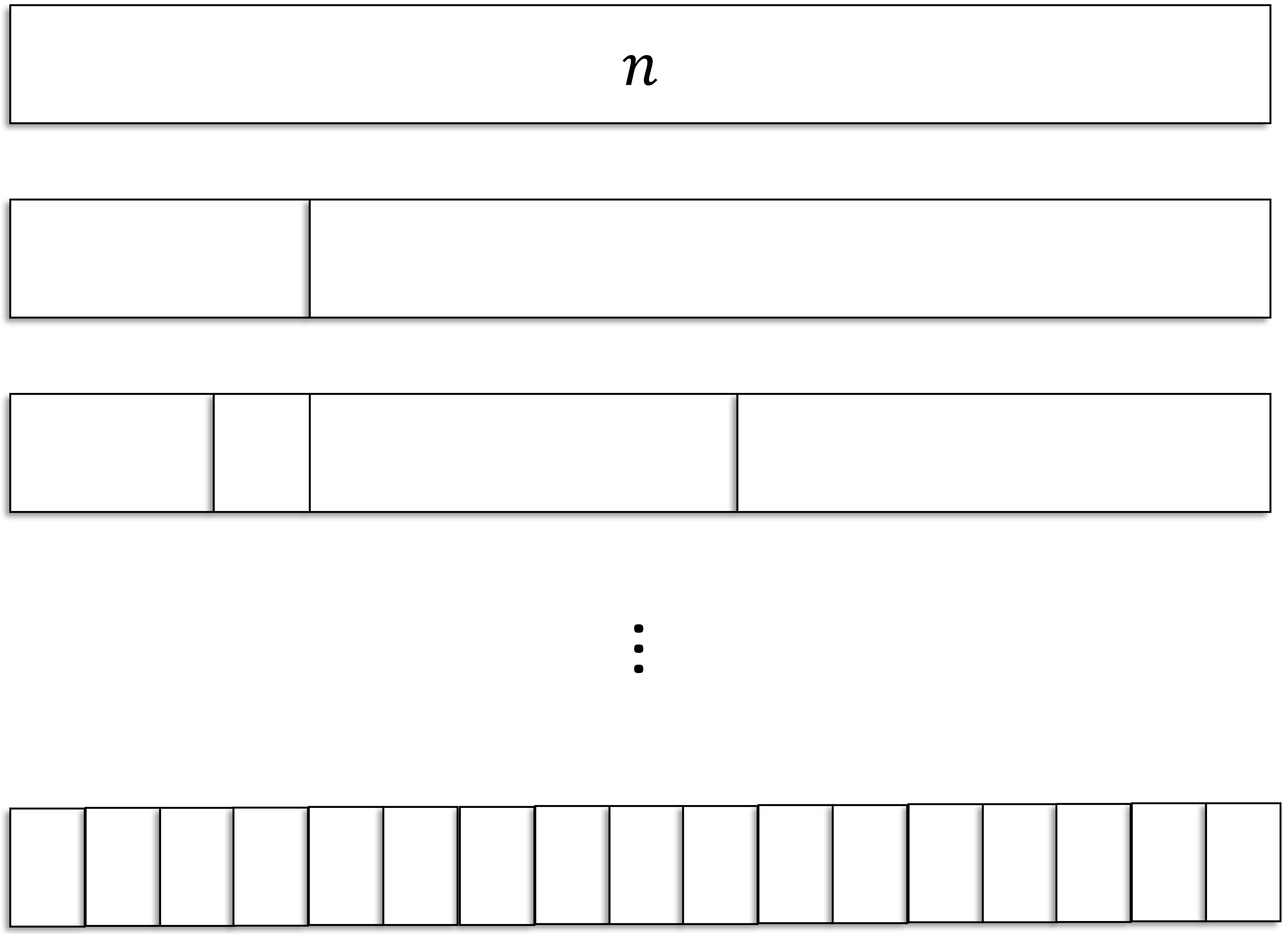
Runtime

$P(\text{good partition}) \geq 4/5$

$E[\#\text{partitions}] \leq 2$

Cost of **Partition** on k elements is $O(k)$

Cost per level:
$E[\#\text{partitions}] \cdot c \cdot n = O(n)$



Runtime

Expected cost per level:
$E[\#\text{partitions}] \cdot c \cdot n = O(n)$

Recursion tree depth is $O(\log n)$

Linearity of Expectation:
Total expected cost is the sum of costs per level:

$$O\big(\log n \cdot O(n)\big) = O(n \log n)$$

Wrap up

Paranoid Quicksort:
Split is always good

Linearity of Expectation

Recursion tree has
bounded depth

Expected runtime:
$O(n \log n)$