

Parallel And Sequential Data Structures and Algorithms

Randomized Algorithms II: Quicksort

Learning Objectives

- Practice using important probability tools by **analyzing parallel randomized quicksort**
- Analyze expected work bounds of randomized algorithms using **indicator random variables** and **linearity of expectation**
- Review **high probability bounds**
- Prove high probability bounds on the span of Quicksort

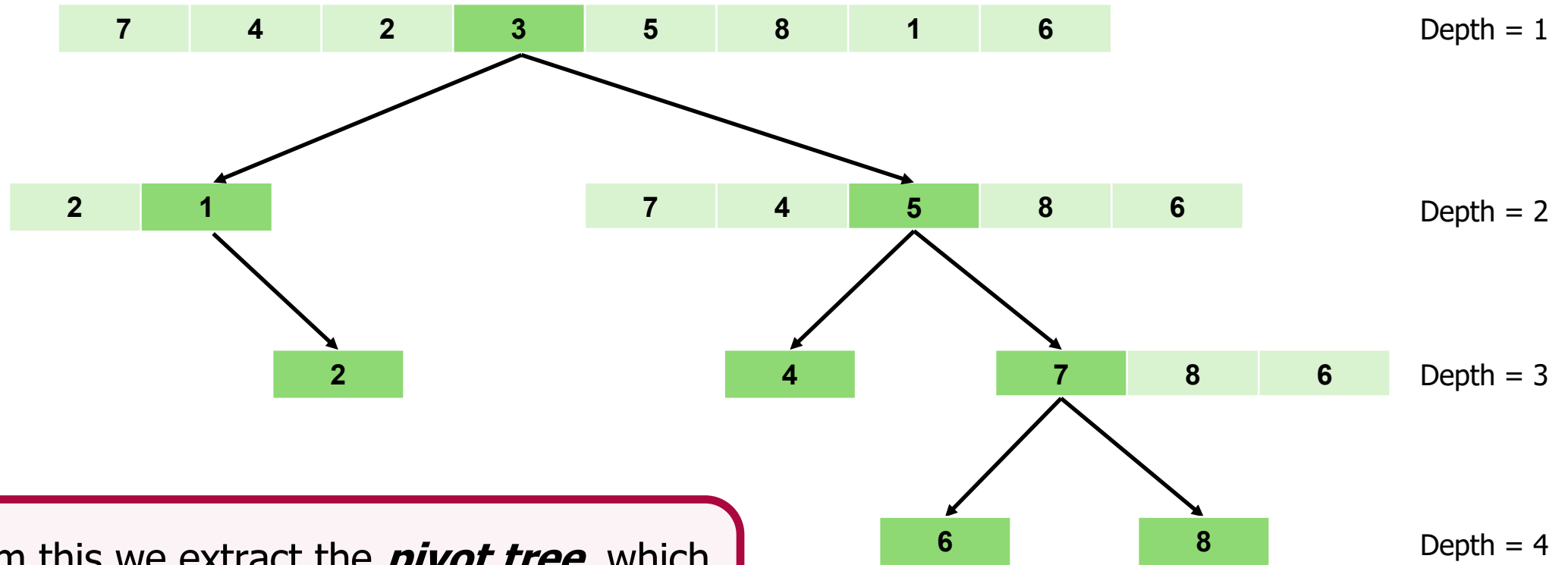
Quicksort

Algorithm (QuickSort):

```
fun quicksort(S : sequence<T>) -> sequence<T>:  
  if length(S) <= 1:  
    return S  
  p = uniform_random_element(S)    // need a source of randomness  
  L = filter(fn x => (x < p), S)  
  R = filter(fn x => (x > p), S)  
  M = filter(fn x => (x == p), S)  
  sortedL, sortedR = parallel (quicksort(L), quicksort(R))  
  return sortedL + M + sortedR
```

Note: This is a functional implementation of Quicksort. The partitioning step is done with three filters rather than in-place (which is harder to parallelize).

Example Run of Quicksort



From this we extract the ***pivot tree***, which is a binary tree in which every element of the input sequence occurs as a node.

Expected Work Analysis

Two Important Tools

Definition (Indicator Random Variable): A random variable that is 1 if an event E happens and 0 otherwise.

$$X_E = \begin{cases} 1, & \text{if } E \text{ occurs} \\ 0, & \text{otherwise} \end{cases}$$

- **Example:** Flip a fair coin n times. Let $X_i = 1$ if flip i is heads, else 0
- Let X be a random variable denoting the number of heads

$$X = \sum_{i \in [n]} X_i$$

Key Idea: Indicator variables let us count complicated events by summing simpler 0/1 random variables.

Two Important Tools

Theorem (Linearity of Expectation): For random variables X and Y ,

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

Extremely important: Linearity holds even when X and Y are not independent!

- Extraordinarily powerful in combination with indicator random variables; even if they are not independent, linearity gives us their expected sum!

$$X = \sum_{i \in [n]} X_i \quad \Rightarrow \quad \mathbb{E}[X] = \sum_{i \in [n]} \mathbb{E}[X_i] = \sum_{i \in [n]} \Pr[X_i = 1] = \sum_{i \in [n]} \frac{1}{2} = \frac{n}{2}$$

Quicksort (Reminder)

Algorithm (QuickSort):

```
fun quicksort(S : sequence<T>) -> sequence<T>:  
  if length(S) <= 1:  
    return S  
  p = uniform_random_element(S)    // need a source of randomness  
  L = filter(fn x => (x < p), S)  
  R = filter(fn x => (x > p), S)  
  M = filter(fn x => (x == p), S)  
  sortedL, sortedR = parallel(quicksort(L), quicksort(R))  
  return sortedL + M + sortedR
```

Quicksort: Cost Setup

Setup: Assume all elements are distinct and write them in sorted order:

$$k_0 < k_1 < \cdots < k_{n-1}$$



Why is it valid to assume that the elements are distinct?

Duplicate elements only make the algorithm more efficient, since it never recurses on the pivot group

Fact: No duplicates is the **worst case!**

In each call to Quicksort, every element in the current subproblem is compared to the pivot a constant number of times.

Key Idea: We can analyze Quicksort's work by **counting comparisons**.

Quicksort: Cost Setup

Key Idea: We can analyze Quicksort's work by **counting comparisons**.

- **New goal:** Count the **expected number of comparisons** of Quicksort
- For each pair $i < j$, define:

$$X_{i,j} = \begin{cases} 1, & \text{if } k_i \text{ and } k_j \text{ are ever compared,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\Rightarrow \text{Expected cost of Quicksort} = \sum_{0 \leq i < j < n} \mathbf{E}[X_{i,j}] = \sum_{0 \leq i < j < n} \Pr[X_{i,j} = 1]$$

$\Rightarrow$ We need the **probability that each pair gets compared**.

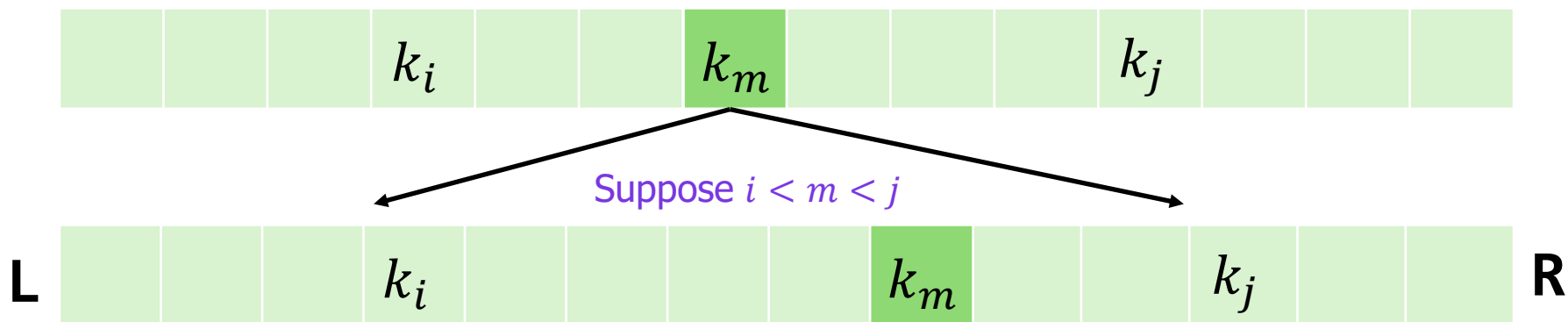
Quicksort: Comparisons

Setup: Assume all elements are distinct and write them in sorted order:

$$k_0 < k_1 < \dots < k_{n-1}$$

?

Consider a pair k_i and k_j with $i < j$. In what circumstance are k_i and k_j compared?



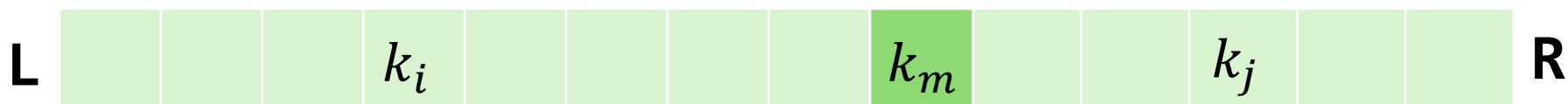
If a pivot k_m with rank strictly **between** i and j is chosen before k_i or k_j are chosen as a pivot, then k_i and k_j are **never compared**!

Quicksort: Comparisons

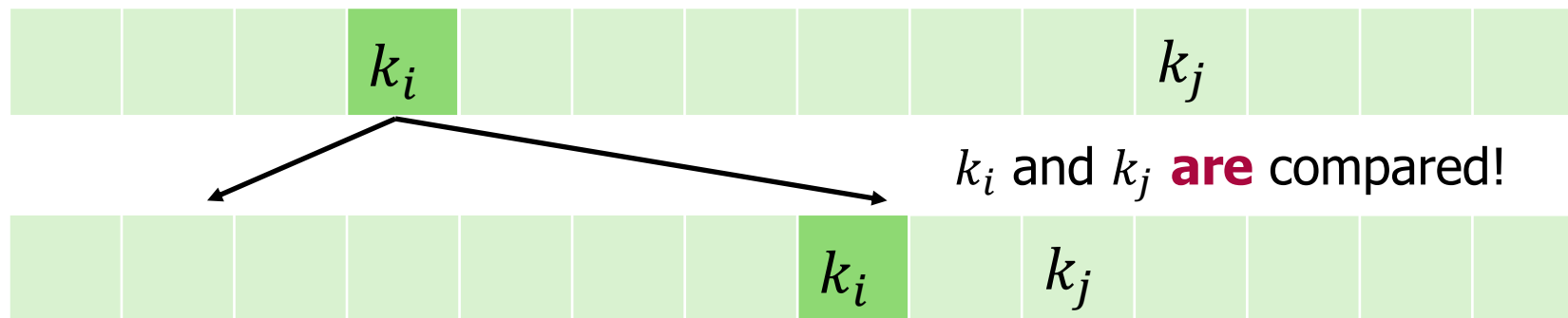
?

Consider a pair k_i and k_j with $i < j$. In what circumstance are k_i and k_j compared?

- If a pivot k_m with rank strictly **between** i and j is chosen before k_i or k_j are chosen as a pivot, then k_i and k_j are **never compared**!



- What if k_i is chosen **before** k_j and anything in between them?



Quicksort: Comparisons



Consider a pair k_i and k_j with $i < j$. In what circumstance are k_i and k_j compared?

- If a pivot k_m with rank strictly **between** i and j is chosen before k_i or k_j are chosen as a pivot, then k_i and k_j are **never compared**!
- If k_i is chosen **before** k_j and anything in between them, they **are** compared
- Symmetrically if k_j is chosen before k_i and anything in between them

Lemma: k_i and k_j are compared if and only if the first pivot chosen from the interval $k_i, \dots, k_j$ is either k_i or k_j

i.e., k_i and k_j compared $\Leftrightarrow$ no pivot chosen earlier than them splits them apart

Quicksort: Comparison Probability

Lemma: k_i and k_j are compared if and only if the first pivot chosen from the interval $k_i, \dots, k_j$ is either k_i or k_j

? In the interval $k_i, \dots, k_j$, how many possible first pivots are there?

$j - i + 1$ possible pivots $k_i, k_{i+1}, \dots, k_{j-1}, k_j$ 2 favourable pivots

? How many of these pivots result in k_i and k_j getting compared?

Corollary:

$$\Pr[X_{i,j} = 1] = \frac{\text{pivot choices in which } k_i, k_j \text{ compared}}{\text{possible pivot choices}} = \frac{2}{j - i + 1}$$

Expected Number of Comparisons

Recall (Key Idea): We can analyze Quicksort's work by **counting comparisons**.

$$\text{Expected cost of Quicksort} = \sum_{0 \leq i < j < n} \mathbf{E}[X_{i,j}] = \sum_{0 \leq i < j < n} \Pr[X_{i,j} = 1]$$

Corollary:

$$\Pr[X_{i,j} = 1] = \frac{2}{j - i + 1}$$

$$\Rightarrow \text{Expected cost of Quicksort} = \sum_{0 \leq i < j < n} \frac{2}{j - i + 1}$$

Calculating the Cost

$$\text{Expected cost} = \sum_{0 \leq i < j < n} \frac{2}{j - i + 1}$$

Technically I only proved $O(n \log n)$.
Exercise: Show that its tight (Θ)

$$= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{2}{j - i + 1}$$

Let $k = j - i + 1$

$$= \sum_{i=0}^{n-1} \sum_{k=2}^{n-i} \frac{2}{k}$$

Recall $H_n = \sum_{k=1}^n \frac{1}{k}$

$$\leq 2 \sum_{i=0}^{n-1} H_n = 2nH_n = \Theta(n \log n)$$

Expected Work: Summary

1. Count comparisons with **indicators**: $X_{i,j} = 1 \Leftrightarrow k_i$ and k_j are compared
2. Use **linearity of expectation**:
$$\mathbf{E}[X] = \sum_{0 \leq i < j < n} \mathbf{E}[X_{i,j}] = \sum_{0 \leq i < j < n} \Pr[X_{i,j} = 1]$$
3. Compute one pair's **probability**:
$$\Pr[X_{i,j} = 1] = \frac{2}{j - i + 1}$$
4. **Sum** over all pairs:
$$\mathbf{E}[X] = \sum_{0 \leq i < j < n} \frac{2}{j - i + 1} = \Theta(n \log n)$$

Theorem: Randomized Quicksort costs $\Theta(n \log n)$ **expected work**.

Expectation is Not Enough

Theorem: Randomized Quicksort has $\Theta(n \log n)$ **expected work**

Claim: The span is $O(D \log n)$ where D is the **recursion depth**

The $O(\log(n))$ comes from the three **filter calls that do the partitioning*

Suppose we analyze the algorithm and managed to prove that **every leaf** in the recursion tree has **expected depth** $O(\log n)$

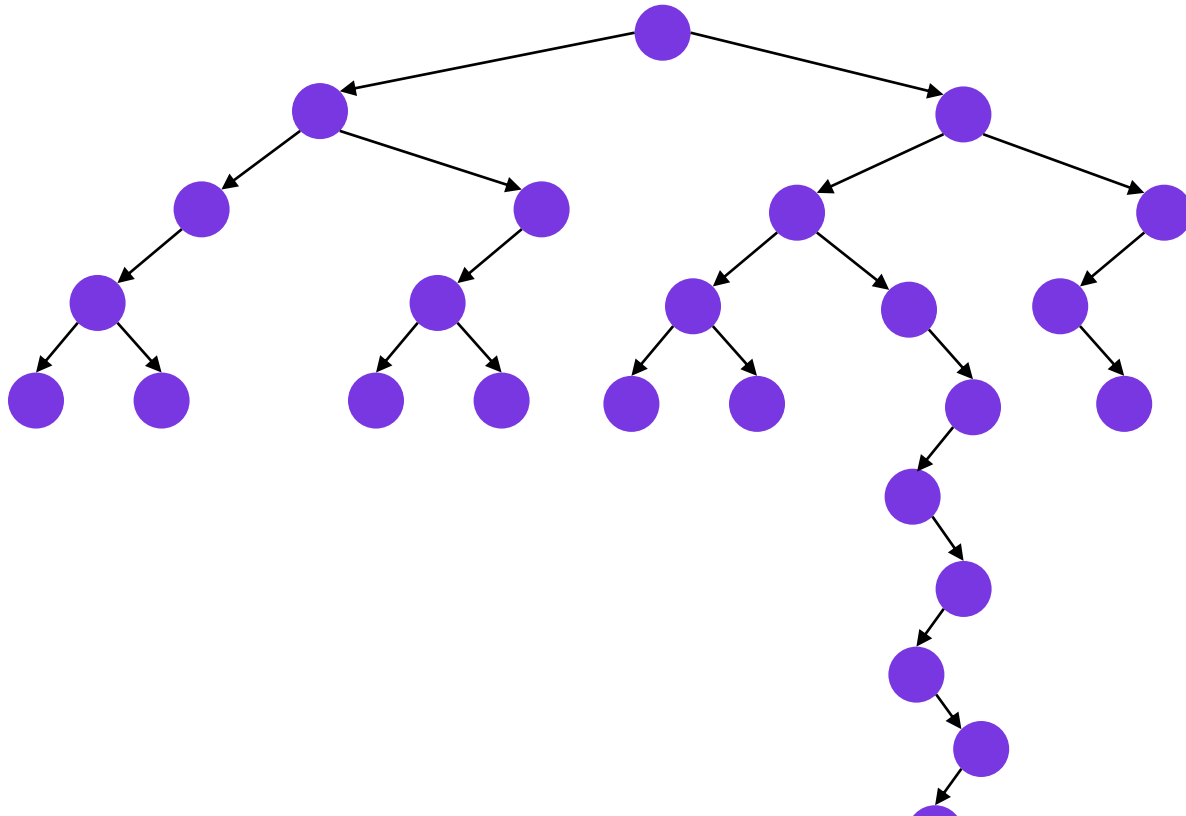
?

What would this tell us about the **expected depth** of the whole recursion tree?

NOTHING!!

Maximum Expectation

$$E \left[\max_{0 \leq i < n} D_i \right] \neq \max_{0 \leq i < n} E[D_i]$$



- Expectation tells us that the **average** depth is low
- But a single unlucky leaf can make the maximum depth much worse...

High Probability Bounds

With High Probability

- We want a guarantee that says **every individual path** is unlikely to be much deeper than the bound. E.g.,

$$\Pr[D_i > c \log(n)] \leq ??$$

- More specifically, we **want** it to be true that

$$\Pr[D_i > c \log(n)] \leq ?? \implies \Pr\left[\max_{0 \leq i < n} D_i > c \log(n)\right] \leq ??$$

- **High probability bounds** have this property!

With High Probability

Definition: We say that $D(n) \leq f(n)$ **with high probability (w.h.p.)** if there exists a constant n_0 such that for **any** integer value $k \geq 1$, and any $n \geq n_0$:

$$\Pr[D(n) \leq k \cdot f(n)] \geq 1 - \frac{1}{n^k}$$

We write $D(n) \in O(f(n))$ w.h.p. if $D(n) \leq c \cdot f(n)$ for some constant c .

The probability that the depth is low is high

The bound holds **for all k** . So, "high probability" = **as high as we want**

With High Probability

Alternate: We say that $D(n) \leq f(n)$ **with high probability (w.h.p.)** if there exists a constant n_0 such that for **any** integer value $k \geq 1$, and any $n \geq n_0$:

$$\Pr[D(n) > k \cdot f(n)] \leq \frac{1}{n^k}$$

We write $D(n) \in O(f(n))$ w.h.p. if $D(n) \leq c \cdot f(n)$ for some constant c .

The probability that the depth is high is low

The bound holds **for all k** . So, "high probability" = **as high as we want**

The Union Bound

Theorem (Union Bound): For any events A and B ,

$$\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$$

Important: The union bound works even when A and B are not independent!

High Probability + Union Bound

- Suppose every individual path satisfies $D_i \leq f(n)$ w.h.p., then:

$$\Pr[D_i > k \cdot f(n)] \leq \frac{1}{n^k}$$

Remember that w.h.p. says that this bound holds **for all** k

- So, it holds for $k' = k + 1$:

$$\Pr[D_i > (k + 1) \cdot f(n)] \leq \frac{1}{n^{k+1}}$$

High Probability + Union Bound

Remember that w.h.p. says that this bound holds for all k

- So, it holds for $k' = k + 1$:

$$\Pr[D_i > (k + 1) \cdot f(n)] \leq \frac{1}{n^{k+1}}$$

- There are at most n leaves in the recursion, so by the **union bound**

$$\Pr\left[\max_{0 \leq i < n} D_i > (k + 1)f(n)\right] \leq \sum_{i=0}^{n-1} \Pr[D_i > (k + 1)f(n)]$$

Max recursion depth

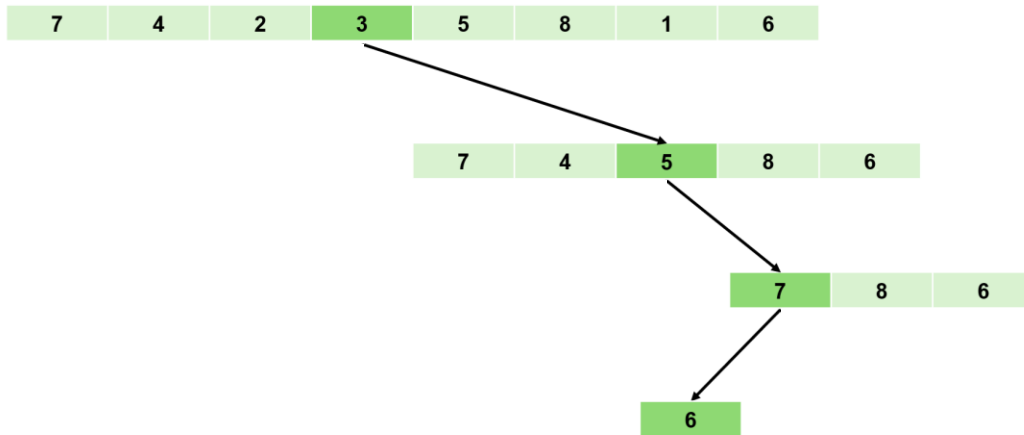
$$\leq n \frac{1}{n^{k+1}} = \frac{1}{n^k}$$

Quicksort Recursion Depth

We just proved: if we can bound the recursion depth of **each fixed leaf** of Quicksort w.h.p. $\Rightarrow$ we bound the maximum recursion depth w.h.p.

?

What does **one leaf** of Quicksort look like in the recursion tree?



It looks identical to **QuickSelect** from last lecture!!

A leaf of Quicksort corresponds to **a single run of QuickSelect!**

Recall: QuickSelect (Last Lecture)

- Recall: The QuickSelect algorithm is the same as Quicksort, except it only takes **one recursive branch** at each level
- Therefore, the leaf of Quicksort's recursion tree corresponding to k_i is the leaf that QuickSelect would reach* when searching for k_i .

Theorem (QuickSelect Recursion Depth): On a sequence of length n , quickselect terminates in $5 \log_2 n$ recursive calls with high probability

Corollary (Quicksort Leaf Depth): On a sequence of length n , for a fixed key k_i , the leaf of Quicksort's recursion tree corresponding to k_i is at most $5 \log_2 n$ deep with high probability.

**assuming the same random pivot choices*

Quicksort Recursion Depth

Corollary (Quicksort Leaf Depth): On a sequence of length n , for a fixed key k_i , the leaf of Quicksort's recursion tree corresponding to k_i is at most $5 \log_2 n$ deep with high probability.

We proved earlier: if we can bound the recursion depth of **one leaf** of Quicksort w.h.p. $\Rightarrow$ we bound the maximum recursion depth w.h.p.

Corollary (Quicksort Recursion Depth): On a sequence of length n , Quicksort's recursion depth is $O(\log n)$ with high probability.

Theorem: Randomized Quicksort costs $O(\log^2 n)$ **span w.h.p.**

Quicksort Analysis: Summary

Theorem: Randomized Quicksort costs $\Theta(n \log n)$ **expected work**.

Proof via counting comparisons using **indicator variables** and **linearity of expectation**

Theorem: Randomized Quicksort costs $O(\log^2 n)$ **span w.h.p.**

Proof via QuickSelect's **high probability** depth and **union bound** over leaves

Key Takeaway: Expected work and w.h.p. span require different probabilistic tools: **indicators + linearity of expectation** for work, **high-probability bounds + union bound** for recursion depth / span.