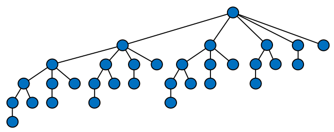
Lecture 6. Randomized Analysis

CpSc 8400: Algorithms and Data Structures
Brian C. Dean



School of Computing
Clemson University
Spring, 2021

1

Recall: Priority Queue Implementations

- There are *many* simple ways to implement the abstract notion of a priority queue as a concrete data structure:

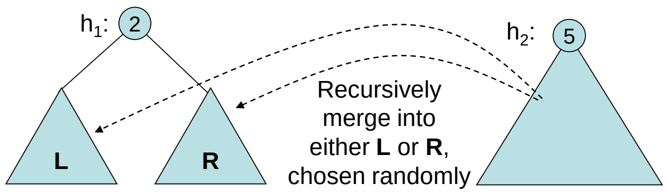
	insert	*remove-min*
Unsorted array or linked list	O(1)	O(n)
Sorted array or linked list	O(n)	O(1)
Binary heap	O(log n)	O(log n)
Balanced binary search tree	O(log n)	O(log n)
Skew heap	O(log n) am.	O(log n) am.
Randomized mergeable binary heap	O(log n) whp.	O(log n) whp.
Binomial heap	O(1) am.	O(log n)
Fibonacci heap	O(1) am.	O(log n) am.

2

2

Recall: The Randomized Mergeable Binary Heap

- Merges any two heaps in O(log n) time with high probability!



- Equivalent outlook: random null path down any tree has length O(log n) with high probability.

3

3

Definition : "With High Probability"

- We say an algorithm with input size n runs in O(log n) time **with high probability** if, for any constant c, we can find another constant k such that

$$\Pr[\text{running time} > k \log n] \le 1 / n^c.$$

- That is, the probability we fail to run in O(log n) time is at most $1 / n^c$, for any constant c of our choosing (as long as we choose a sufficiently large hidden constant in the O(log n) notation).

4

4

An Easy Starting Point

- Suppose we have an algorithm for which:
  - We start with a problem of size n.
  - In every iteration, the effective size of the problem is reduced to a constant fraction of its original size
- Then, our algorithm performs only O(log n) iterations

The Randomized Reduction Lemma

- Suppose we have an algorithm for which:
  - We start with a problem of size n.
  - In every iteration, the effective size of the problem is reduced to a constant fraction of its original size **with some constant probability.**
- Then, our algorithm performs only O(log n) iterations **with high probability**.

Example: Randomized Binary Search

- Suppose we're searching a sorted array A[1…n] for an element of value X.
- In a standard binary search, we compare A[n/2] and X, then recursively search either A[1…n/2] or A[n/2…n].
- Suppose instead of A[n/2], we compare X against a random element and then recursively search left or right as before…

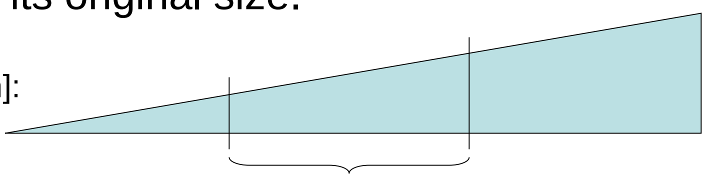
7

7

Example: Randomized Binary Search

- **Claim**: Randomized binary search runs in O(log n) time with high probability.
- **Simple proof**: If we happen to choose a "pivot" element A[j] where n/3 ≤ j ≤ 2n/3 (and this happens with probability ≥ 1/3), then our problem will be reduced to ≤ 2/3 of its original size.

A[1..n]:



8

8

Applying the Randomized Reduction Lemma to Mergeable Heaps

- Suppose we build a null path in an arbitrary heap-ordered binary tree by walking downward at random from the root.
- In each step (say, at element e), one of e's left or right subtrees must contain fewer than ½ of the total number of elements in e's subtree.
  - So with probability ½, each step reduces the number of nodes in our current subtree by a factor of at least 2.
  - Hence, a randomly-built null path has length O(log n) with high probability.
- On a randomized mergeable binary heap, *merge* and all other priority queue operations therefore run in O(log n) time with high probability.

9

9

Randomized Reduction Exercise

- Suppose we build an n-element binary heap by inserting n elements in random order.
- How many times do we change the root?

10

10

Randomized Reduction Exercise

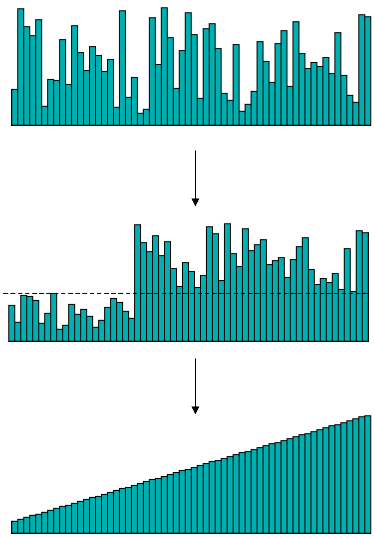
- Suppose we build an n-element binary heap by inserting n elements in random order.
- How many times do we change the root?
- Equivalently, if you scan through a random permutation of n elements while keeping a running minimum, how many times will you see this minimum being reset?

11

11

Quicksort

- In linear time, **partition** array based on the value of some **pivot** element.
- Then recursively sort left side (all elements ≤ pivot) and right side (≥ pivot).
- How to partition?
  - Ideally "in place"
  - Careful if equal elements…



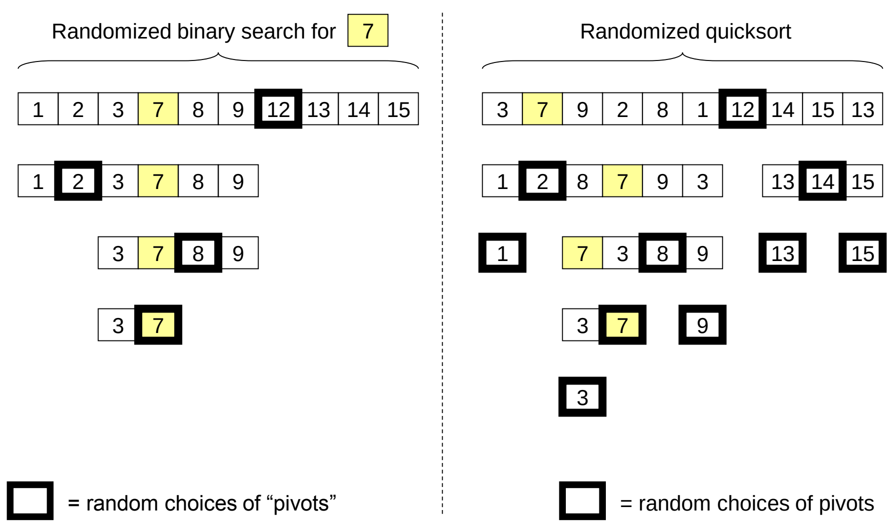
12

12

Quicksort Variants

- **Simple quicksort.** Choose pivot using a simple deterministic rule; e.g., first element, last element, median(A[1], A[n], A[n/2]).
  - Θ(n log n) time if "lucky", but $\Theta(n^2)$ worst-case.
- **Deterministic quicksort.** Pivot on median (we'll see shortly how to find the median in linear time).
  - Θ(n log n) time, but not the best in practice.
- **Randomized quicksort.** Choose pivot uniformly at random.
  - Θ(n log n) time with high probability, and fast / popular in practice

Randomized Reduction and Randomized Quicksort



The Union Bound

- Recall that **Pr**[A U B] = **Pr**[A] + **Pr**[B] – **Pr**[A ∩ B].
- It typically suffices just to use the rough upper bound **Pr**[A U B] ≤ **Pr**[A] + **Pr**[B].
- For multiple events $E_1 \ldots E_k$, this gives us what is known as the **union bound** or **Boole's inequality**:
  Pr[E_1 U E_2 U … U E_k] ≤ **Pr**[E_1] + … + **Pr**[E_k].
- For example:
  - Suppose each of 50 parts in a complex machine fails with probability ≤ 1/100
  - Then **Pr**[entire machine fails] ≤ 50(1/100) = ½.

15

15

The Union Bound

- If there are n bad events that can happen, and each happens with probability ≤ p, then the probability **any** bad event happens is at most np.
  I.e, for events $E_1 \ldots E_n$, **Pr**[$\cup_i E_i$] ≤ Σ_i **Pr**[E_i]
- This meshes particularly well with our definition of "with high probability": If some property holds for a generic input element w.h.p., then it also holds for each of our n input elements w.h.p.
- **Example:** If randomized quicksort spends only O(log n) on a generic input element w.h.p, then its total running time is O(n log n) w.h.p.

16

16

A Prototypical "High Probability" Analysis…

- **Step 1:** Consider some generic input element e. Show that randomized quicksort spends O(log n) work on e w.h.p.
  - Usually easy with the randomized reduction lemma.
- **Step 2:** The union bound automatically allows us to extend this result to show that randomized quicksort spends O(log n) work on **each** of its input elements w.h.p.
- So total running time is O(n log n) w.h.p.

More Randomized Reduction Practice

- Throw N balls into N bins, each one independently at random. How full is the fullest bin?

More Randomized Reduction Practice

- Throw N balls into N bins, each one independently at random. How full is the fullest bin? O(log N) balls whp!

19

19

More Randomized Reduction Practice

- Throw N balls into N bins, each one independently at random. How full is the fullest bin? O(log N) balls whp!
- Coupon collecting: Each time you open a box of cereal, you receive one of N coupon types (independently at random). How many boxes of cereal must you open to collect at least one of each coupon type?

20

20

More Randomized Reduction Practice

- Throw N balls into N bins, each one independently at random. How full is the fullest bin? O(log N) balls whp!
- Coupon collecting: Each time you open a box of cereal, you receive one of N coupon types (independently at random). How many boxes of cereal must you open to collect at least one of each coupon type? O(N log N) whp!
  - Many related questions; e.g., cover time of a complete graph.