MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 8

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Manal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55
Class Room : 2101

More intuition

- Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ….
- $L(n) = 2U(n/2) + \Theta(n)$ ***lucky***
- $U(n) = L(n-1) + \Theta(n)$ ***unlucky***

Solving:

$$L(n) = 2(L(n/2-1) + \Theta(n/2)) + \Theta(n)$$
$$= 2L(n/2-1) + \Theta(n)$$
$$= \Theta(n \lg n)$$

How can we make sure we are usually lucky ?

Randomized quicksort

- **IDEA:** Partition around a ***random*** *element.*
- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Pseudocode for Randomized quicksort

```
RANDOMIZED-PARTITION(A, p, r)
1   i ← RANDOM(p, r)
2   exchange A[r] ↔ A[i]
3   return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
1   if p < r
2      then q ← RANDOMIZED-PARTITION(A, p, r)
3           RANDOMIZED-QUICKSORT(A, p, q - 1)
4           RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Randomized quicksort analysis

- Let T(n) =the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.
- For k= 0, 1, …, n–1, define the **indicator random variable:**

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

- $E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.

Analysis (continued)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n) \right)$$

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectations of both sides.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Linearity of expectation.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

Independence of X_k from other random choices.

Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)
\end{aligned}
$$

Linearity of expectation; $E[X_k] = 1/n$.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

$$= \frac{2}{n}\sum_{k=1}^{n-1} E[T(k)] + \Theta(n)$$

Summations have identical terms.

Exercise

$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0, 1$ *terms can be absorbed in the* $\Theta(n)$.)

Exercise:

Prove: $E[T(n)] \le an \lg n$ for constant $a > 0$.

– Choose a large enough so that $an \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$

Substitution method

$$E[T(n)] \le \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

Substitute inductive hypothesis.

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

Use fact.

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\
&\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an \lg n - \left(\frac{an}{4} - \Theta(n)\right)
\end{aligned}
$$

Express as ***desired*** – ***residual***.

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\
&\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an \lg n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\le an \lg n ,
\end{aligned}
$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from ***code tuning.***
- Quicksort behaves well even with caching and virtual memory.

How fast can we sort?

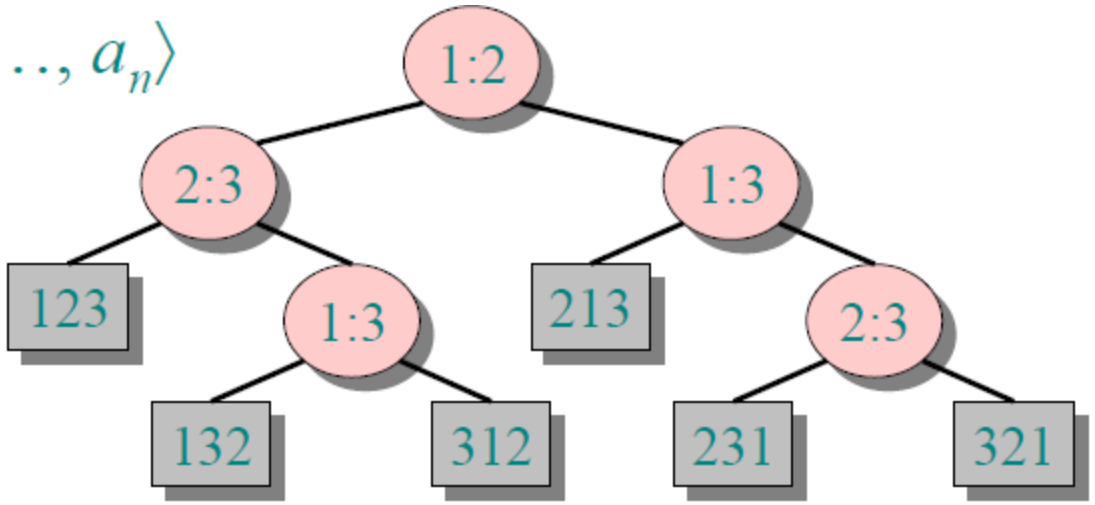
- All the sorting algorithms we have seen so far are **comparison sorts:** only use comparisons to determine the relative order of elements.
- *E.g.,* insertion sort, merge sort, quicksort, heapsort.
- The best worst-case running time that we've seen for comparison sorting is $O(n \lg n)$.

Is $O(n \lg n)$ ***the best we can do ?***

- Decision tree scan help us answer this question.

Decision-tree example

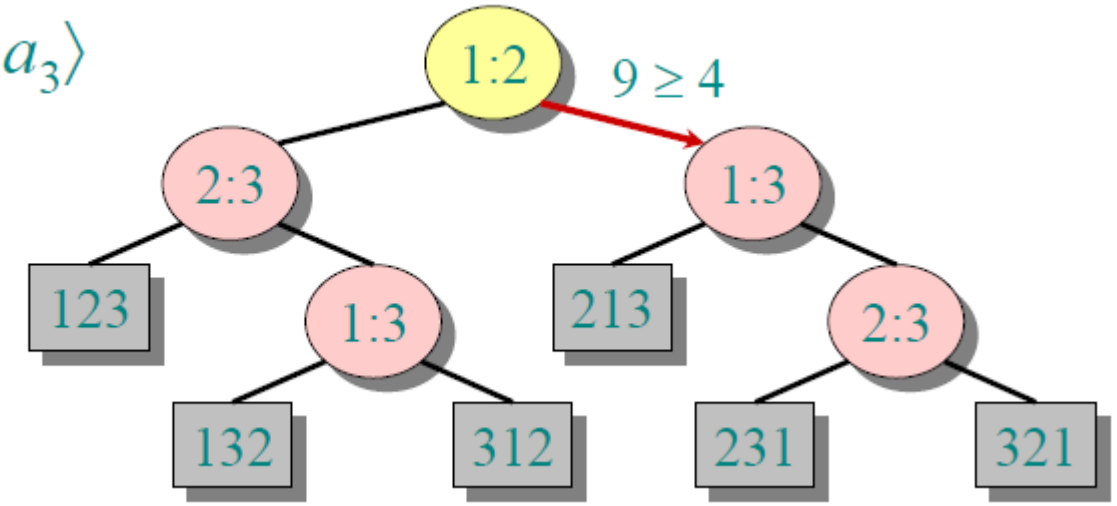
Sort $\langle a_1, a_2, \ldots, a_n \rangle$



- Each internal node is labeled *i:j for* $i, j \in \{1, 2, \ldots, n\}$.
- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Decision-tree example

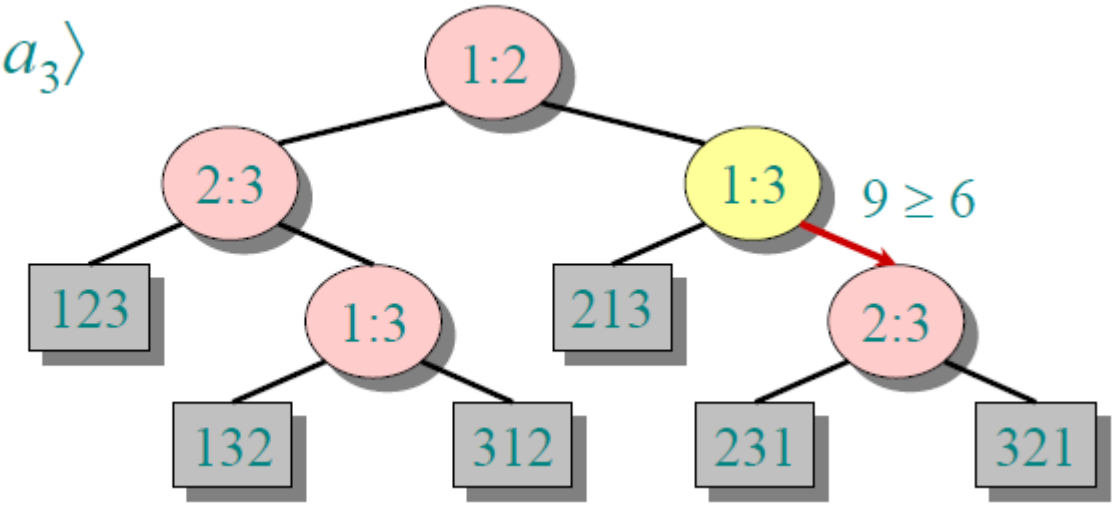
Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



- Each internal node is labeled *i:j for* $i, j \in \{1, 2, \ldots, n\}$.
- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree example

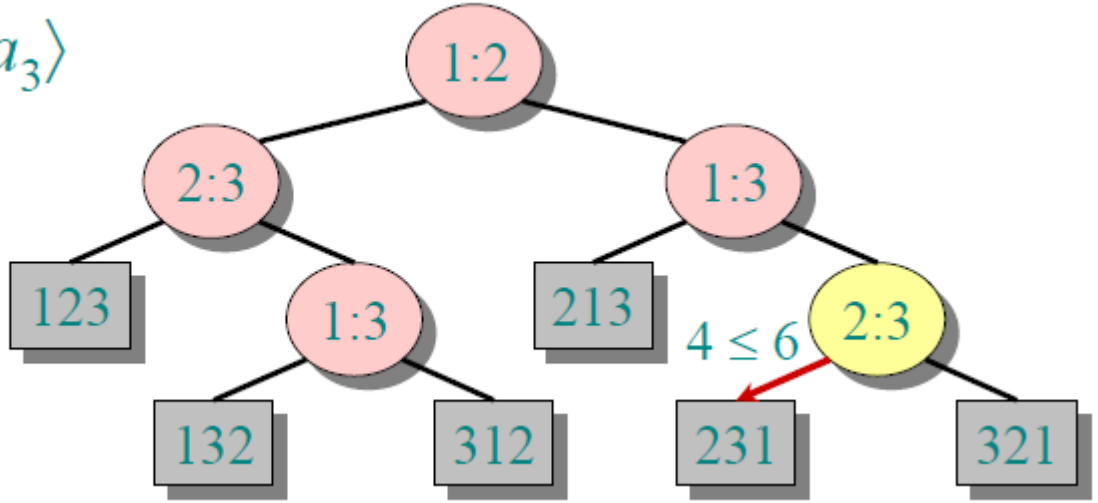
Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



- Each internal node is labeled *i:j for* $i, j \in \{1, 2, \ldots, n\}$.
- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Decision-tree example

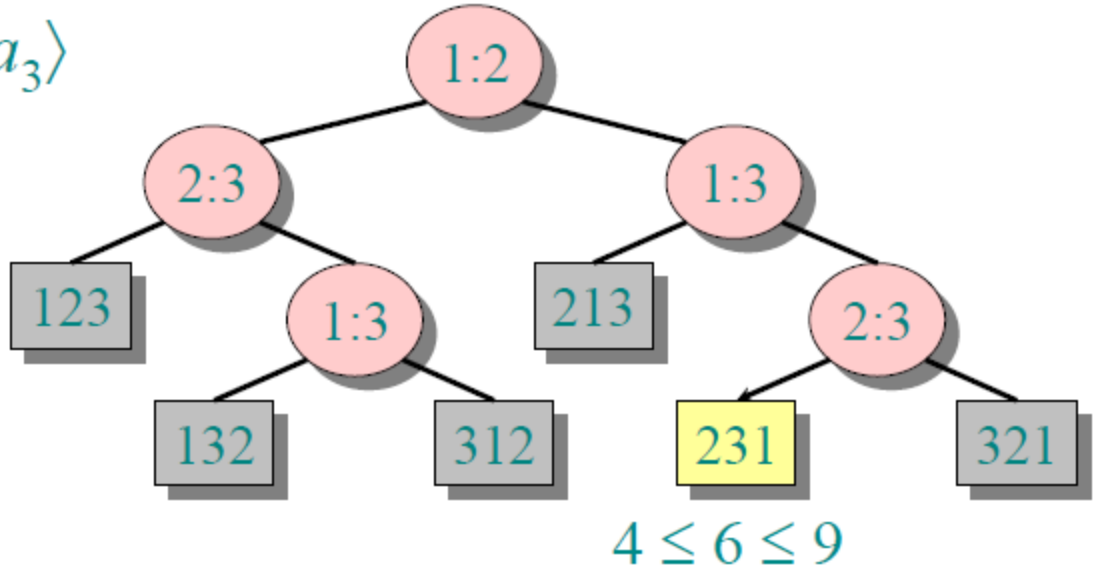
Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



- Each internal node is labeled *i:j for* $i, j \in \{1, 2, \ldots, n\}$.
- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



- Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$ has been established.

Decision-tree model

A decision tree can model the execution of any comparison sort:

- One tree for each input size n.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = height of tree.

Lower bound for decision-tree sorting

Theorem: Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof: The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations. A height-h binary tree has $\leq 2^h$ leaves. Thus, $n! \leq 2^h$.

$$
\begin{aligned}
\therefore\ h &\geq \lg (n!) && (\lg \text{ is mono. increasing}) \\
&\geq \lg ((n/e)^n) && (\text{Stirling's formula}) \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n).
\end{aligned}
$$

Lower bound for comparison sorting

Corollary: Heapsort and merge sort are asymptotically optimal comparison sorting algorithms.