CH08-320201

Algorithms and Data Structures

ADS

Lecture 9

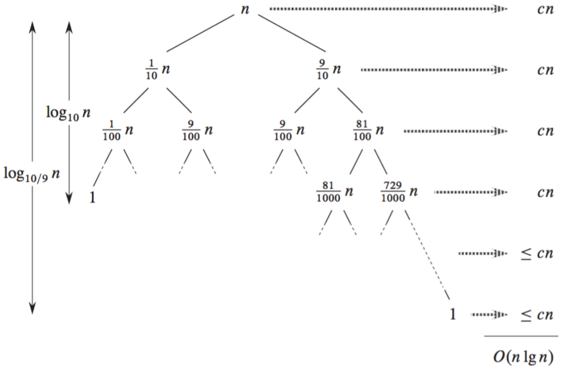
Dr. Kinga Lipskoch

Spring 2019

Runtime Analysis (5)

What if the split is 1/10 : 9/10?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$



Runtime Analysis

- What if we alternate between lucky and unlucky choices
  - $L(n) = 2U(n/2) + \Theta(n)$ 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 that this is usually happening?

Randomized Quicksort (1)

- 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.

Randomized Quicksort (2)

```
RANDOMIZED-PARTITION(A, p, r)
    i = RANDOM(p, r)
    exchange A[p] with A[i]
    return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
    if p < r
        q = RANDOMIZED-PARTITION(A, p, r)
        RANDOMIZED-QUICKSORT(A, p, q - 1)
        RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Randomized Quicksort (3)

- Let $T(n)$ be the random variable for the running time of the randomized quicksort on an input of size n (assuming random numbers are independent).

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

- For $k = 0, 1, ..., n-1$, define indicator random variable
- $E[X_k] = Pr\{X_k = 1\} = 1/n$, since all splits are equally likely (assuming elements are distinct).

Randomized Quicksort (4)

Recurrence:

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

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

Randomized Quicksort (5)

Calculating expectations:

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

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

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

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

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

Randomized Quicksort (6)

- Use substitution method to solve recurrence.
- Guess: $E[T(n)] = \Theta(n \lg n)$.
- Prove: $E[T(n)] \leq an \lg n$ for constant $a > 0$.
- Use:

$$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

(proof by induction)

Randomized Quicksort (7)

Proof:

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

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

if a is chosen large enough.

Quicksort: Conclusion

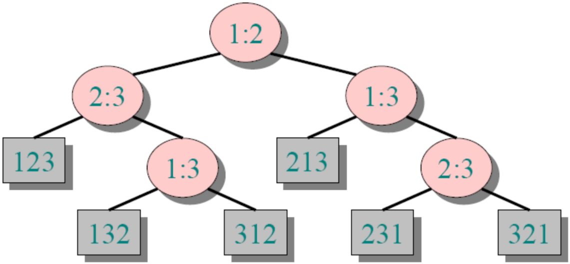
- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is often the best practical choice because its expected runtime is $\Theta(n \lg n)$ and the constant is quite small.
- Quicksort is typically over twice as fast as MergeSort.
- Quicksort is an in-situ sorting algorithm (debatable).
- Quicksort has a worst-case runtime of $\Theta(n^2)$ when the array is already sorted.
- Visualization Randomized Quicksort: http://www.sorting-algorithms.com/quick-sort

Comparison Sorts

- All sorting algorithms we have seen so far are comparison sorts.
- A comparison sort only uses comparisons to determine the relative order of elements.
- The best worst-case running time we encountered for comparison sorting was $O(n \lg n)$.
- Is $O(n \lg n)$ the best we can do?

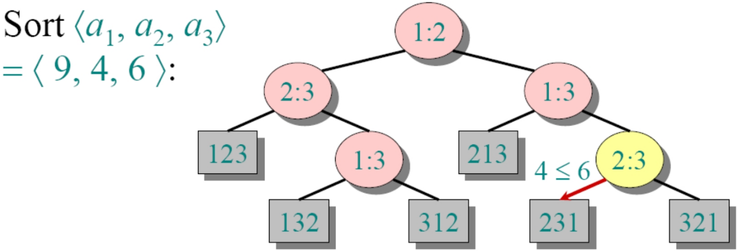
Decision Tree (1)

- Sort $< a_1, a_2, ..., a_n >$
- Each internal node is labeled $i : j$ for $i, j \in \{1, 2, ..., n\}$.
- Left subtree shows subsequent comparisons if $a_i \leq a_j$.
- Right subtree shows subsequent comparisons if $a_i \geq a_j$.



Decision Tree (2)

Example:
Each leaf contains a permutation $< \pi(1), \pi(2), ..., \pi(n) >$ indicating the order $a_{\pi(1)} \leq a_{\pi(2)} \leq ... \leq a_{\pi(n)}$.



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.

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, $n! \leq 2^h$.
Then,

$$\begin{aligned} h &\geq \lg(n!) \\ &\geq \lg((n/e)^n) \\ &= n \lg n - n \lg e \\ &= \Omega(n \lg n). \end{aligned}$$

Used Stirling's formula: $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ when $n \to \infty$.

Lower Bound for Comparison Sorting

- The lower bound for comparison sorting $\Omega(n \lg n)$.
- Heap Sort and Merge Sort are asymptotically optimal comparison sorting algorithms.

Non-Comparison Sorting?

- Is it possible to avoid comparisons between elements?
- Yes, if we can make assumptions on the input data.
- E.g., trivial case:
  - Input: $A[1...n]$, where $A[j] \in \{1, 2, ..., n\}$, and $A[i] \neq A[j]$ for all $i \neq j$
  - Output: $B[1...n]$