Lecture: Analysis of Algorithms (CS583 - 004)

Amarda Shehu

Spring 2019

1. Probabilistic Analysis
   - Average Case Analysis of Insertion Sort

Analyzing Average Case Time Complexity

Definition

Let $T(n)$ denote the average case time complexity used by an algorithm to solve a problem on an input size n. Then:

$$T(n) = \sum_{I \in D_n} P(I) \circ t(I)$$

- D_n is the set of all input instances of size n
- I denotes instance I taking values over sample space D_n
- $P(I)$ denotes the probability with which I occurs
- $t(I)$ denotes time it takes to solve problem on input instance I
- $\sum_{I \in D_n} P(I) = 1$ for correct analysis

Light Exercise: Average Case Analysis of Insertion Sort

?

Light Exercise: Average Case Analysis of Insertion Sort

?

Need a bit of a refresher on *expected* values and *random variables*

Light Exercise: Average Case Analysis of Insertion Sort

?

Need a bit of a refresher on *expected* values and *random variables*

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$$

Expected number of H's from one flip of a fair coin is $1/2$.

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$

Expected number of H's from one flip of a fair coin is $1/2$.

Q: What is the expected number of Heads in n tosses of a coin?

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$

Expected number of H's from one flip of a fair coin is $1/2$*.*

Q: What is the expected number of Heads in n tosses of a coin?

Let $X = \sum_{i=1}^{n} X_{H,i}$ be the total number of H's in n tosses.

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$
Expected number of H's from one flip of a fair coin is $1/2$.

Q: What is the expected number of Heads in n tosses of a coin?

Let $X = \sum_{i=1}^{n} X_{H,i}$ be the total number of H's in n tosses.

Then: $$E[X] = E[\sum_{i=1}^{n} X_{H,i}] = \sum_{i=1}^{n} E[X_H] = \sum_{i=1}^{n} 1/2 = n/2$$

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$

Expected number of H's from one flip of a fair coin is $1/2$.

Q: What is the expected number of Heads in n tosses of a coin?

Let $X = \sum_{i=1}^{n} X_{H,i}$ be the total number of H's in n tosses.

$$\text{Then: } E[X] = E[\sum_{i=1}^{n} X_{H,i}] = \sum_{i=1}^{n} E[X_H] = \sum_{i=1}^{n} 1/2 = n/2$$

Expected number of H's from n tosses of a fair coin is $1/2$.

Refresher in Context of Simple Coin Tossing Example

Q: What is the expected number of Heads from one coin toss?

Introduce binary random variable X_H to track this number

$E[X_H] = 1 \cdot P(X_H = 1) + 0 \cdot P(X_H = 0) = 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$

Expected number of H's from one flip of a fair coin is $1/2$.

Q: What is the expected number of Heads in n tosses of a coin?

Let $X = \sum_{i=1}^{n} X_{H,i}$ be the total number of H's in n tosses.

$$\text{Then: } E[X] = E[\sum_{i=1}^{n} X_{H,i}] = \sum_{i=1}^{n} E[X_H] = \sum_{i=1}^{n} 1/2 = n/2$$

Expected number of H's from n *tosses of a fair coin is* $1/2$.

Back to Average Case Analysis of Insertion Sort

InsertionSort(**array** $A[1 \ldots n]$)

```
for j ← 2 to n do
    Temp ← A[j]
    i ← j − 1
    while i > 0 and A[i] > Temp do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← Temp
```

- Loop invariant: At the start of each iteration j, $A[1 \ldots j-1]$ is sorted.

Recall:
$T(n) = \sum_{j=2}^{n} \{A + \sum_{i=0}^{j-1} B + C\}$

Ignoring machine-dependent constants, we can write:
$T(n) = \sum_{j=2}^{n} k_j$, where k_j is a variable that tracks the total number of iterations of the inner while loop in an iteration of the outer for loop

In the worst-case analysis, we assumed that $k_j = j$, arriving at a total quadratic running time for insertion sort.

Here we ask for $E[k_j]$

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$?

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$? $E[k_i] = P(move) * 1 + P(no\ move) * 0$

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$? $E[k_i] = P(move) * 1 + P(no\ move) * 0$

$P(move) = P(A[i] > Key) = 0.5$

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$? $E[k_i] = P(move) * 1 + P(no\ move) * 0$

$P(move) = P(A[i] > Key) = 0.5$

So: $E[k_i] = 0.5 * 1 = 0.5 \implies E[k_j] = \sum_{i=1}^{j-1} 0.5 = \frac{j-1}{2}$

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$? $E[k_i] = P(move) * 1 + P(no\ move) * 0$

$P(move) = P(A[i] > Key) = 0.5$

So: $E[k_i] = 0.5 * 1 = 0.5 \implies E[k_j] = \sum_{i=1}^{j-1} 0.5 = \frac{j-1}{2}$

Finally: $E[T(n)] = \sum_{j=2}^{n} \frac{j-1}{2}$

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$? $E[k_i] = P(move) * 1 + P(no\ move) * 0$

$P(move) = P(A[i] > Key) = 0.5$

So: $E[k_i] = 0.5 * 1 = 0.5 \implies E[k_j] = \sum_{i=1}^{j-1} 0.5 = \frac{j-1}{2}$

Finally: $E[T(n)] = \sum_{j=2}^{n} \frac{j-1}{2}$

You can show that this expected running time is no better than the worst-case running time.

Average Case Analysis of Insertion Sort

k_j: random variable counting total number of moves to the right

So: $E[k_j] = E[\sum_{i=1}^{j-1} k_i]$, where k_i is a random variable tracking the number of moves in one iteration of the while loop

By linearity of expectation: $E[k_j] = \sum_{i=1}^{j-1} E[k_i]$

What is $E[k_i]$? $E[k_i] = P(move) * 1 + P(no\ move) * 0$

$P(move) = P(A[i] > Key) = 0.5$

So: $E[k_i] = 0.5 * 1 = 0.5 \implies E[k_j] = \sum_{i=1}^{j-1} 0.5 = \frac{j-1}{2}$

Finally: $E[T(n)] = \sum_{j=2}^{n} \frac{j-1}{2}$

You can show that this expected running time is no better than the worst-case running time.

Can we do better than $\theta(n^2)$?

You have have already seen an example ...

Can we do better than $\theta(n^2)$?

You have have already seen an example ...

More follow

Can we do better than $\theta(n^2)$?

You have have already seen an example ...

More follow

Lecture: Analysis of Algorithms (CS583 - 004)

Amarda Shehu

Spring 2019

1. Outline of Today's Class
   - Sorting in O(n lg n) Time on Average: Quicksort

Quicksort: Divide and Conquer

- Proposed by C. A. R. Hoare in 1962
- Implements the divide-and-conquer paradigm
- Is a very practical algorithm
- Sorts in place like insertion sort and heapsort
  1. Divide: Partition array into two subarrays around a *pivot* x s.t. values left $\leq$ x $\leq$ values right
  2. Conquer: Recursively sort the two subarrays
  3. Combine: Trivial

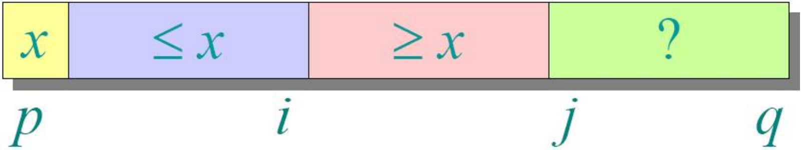
- **Key to speed: linear-time partitioning subroutine**

Quicksort: Partitioning Subroutine

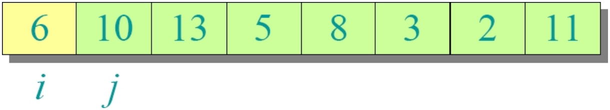
PARTITION(A, p, q)

$x \leftarrow A[p]$
$i \leftarrow p$
for $j \leftarrow p + 1$ to q **do**
    if $A[j] \leq x$ **then**
        $i \leftarrow i + 1$
        swap$(A[i], A[j])$
return i

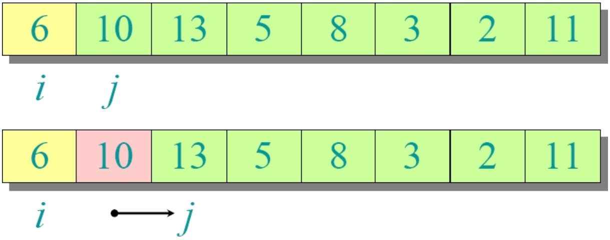
Running time $= O(n)$ for n elements.



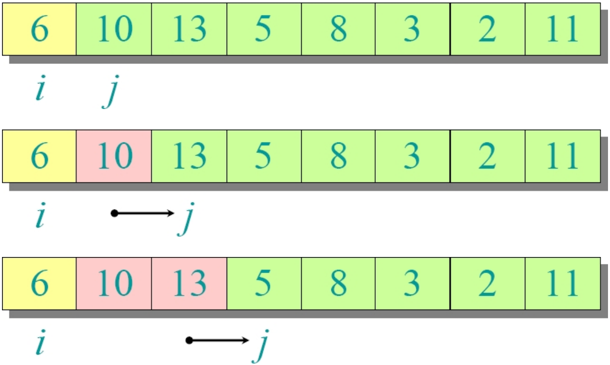
Partitioning: Trace



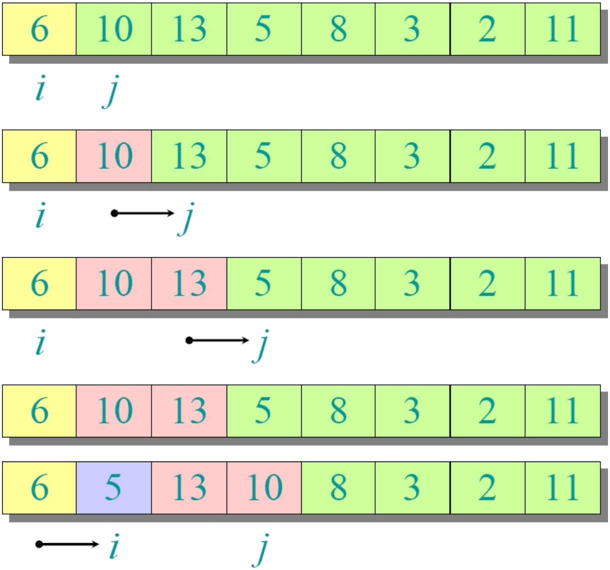
Partitioning: Trace



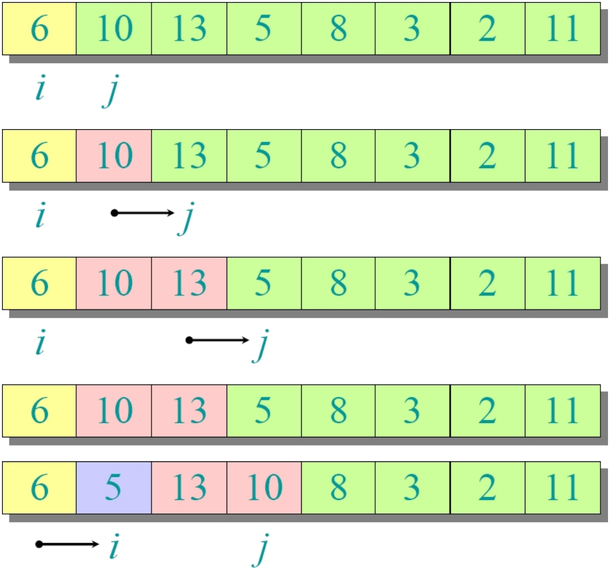
Partitioning: Trace



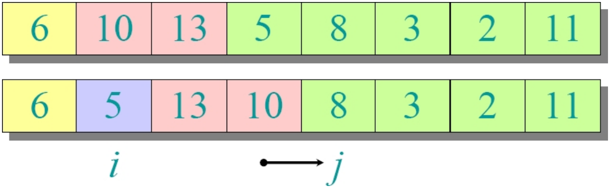
Partitioning: Trace



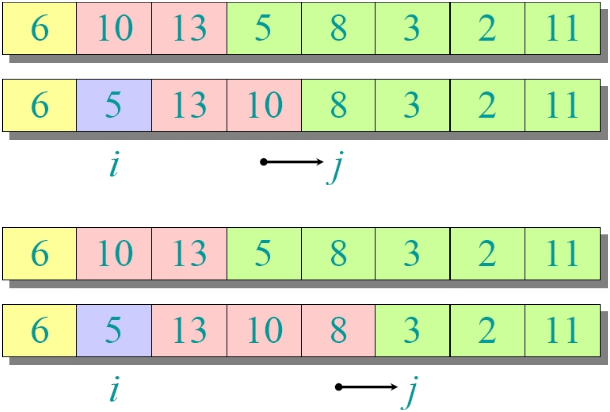
Partitioning: Trace



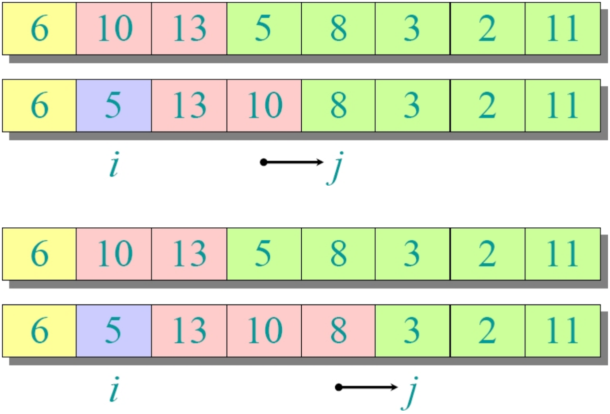
Partitioning: Trace



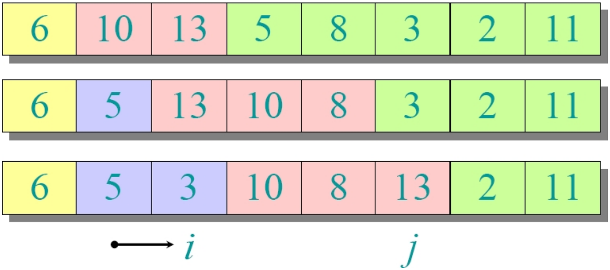
Partitioning: Trace



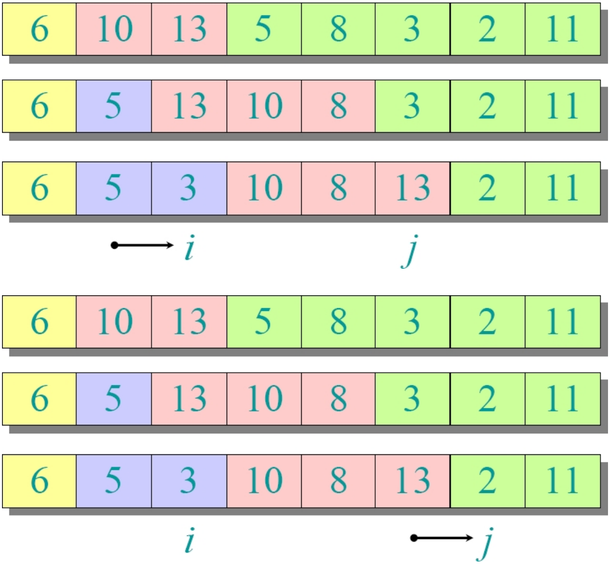
Partitioning: Trace



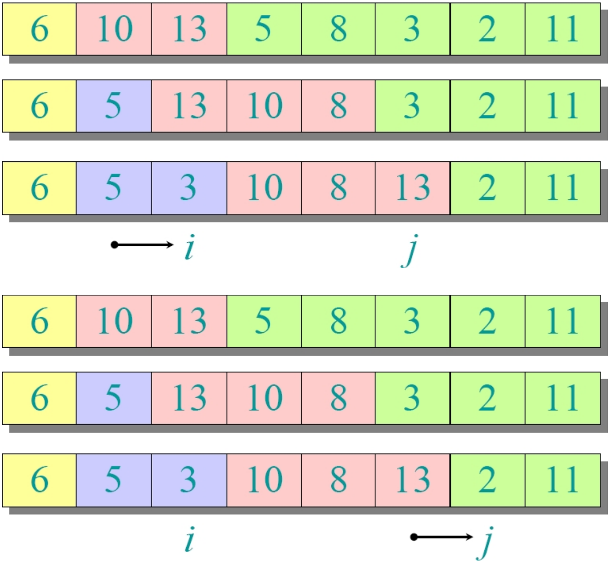
Partitioning: Trace



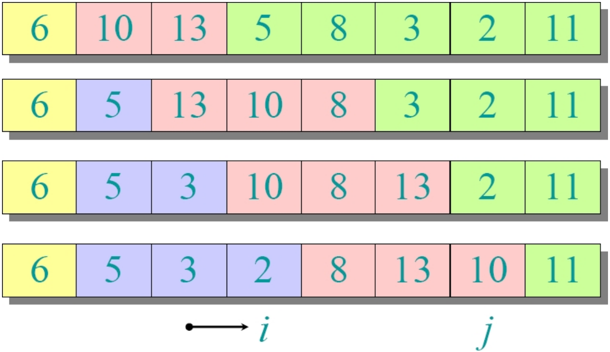
Partitioning: Trace



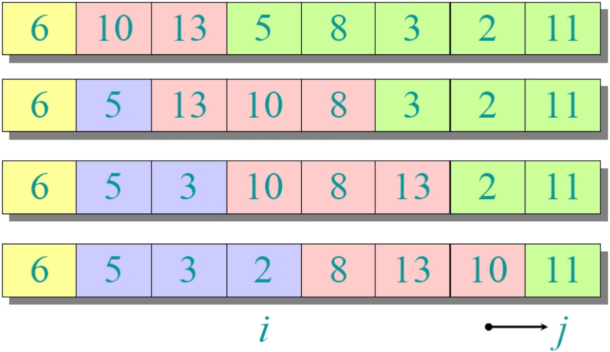
Partitioning: Trace



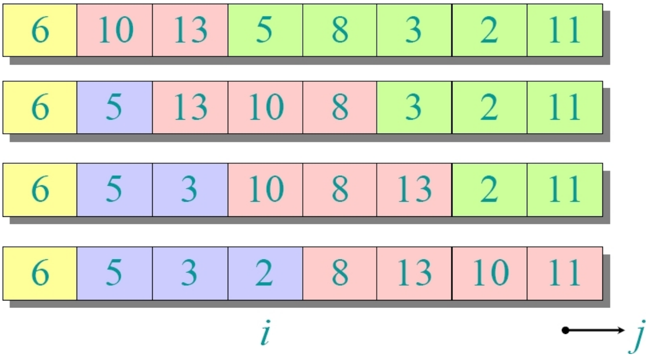
Partitioning: Trace



Partitioning: Trace



Partitioning: Trace



Quicksort: Pseudocode And Analysis

QUICKSORT(A, p, r)

```
if p < r then
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

Initial call: QUICKSORT$(A, 1, n)$

Worst-case Time Analysis:

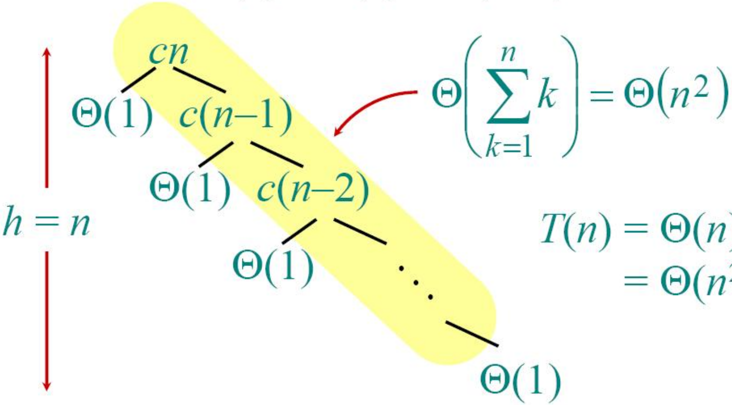
- Assume elements are distinct
- There are better algorithms for duplicate elements

- Let $T(n)$ be worst-case running time on n elements
- A is sorted/reverse sorted; partition around min/max element
- One side of partition always has no elements

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \theta(n) \\ &= \theta(1) + T(n-1) + \theta(n) \\ &= T(n-1) + \theta(n) \text{ - arithmetic series} \\ &= \theta(n^2) \end{aligned}$$

Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$



$$T(n) = \Theta(n) + \Theta(n^2)$$
$$= \Theta(n^2)$$

Best-case (Lucky) Analysis for Intuition

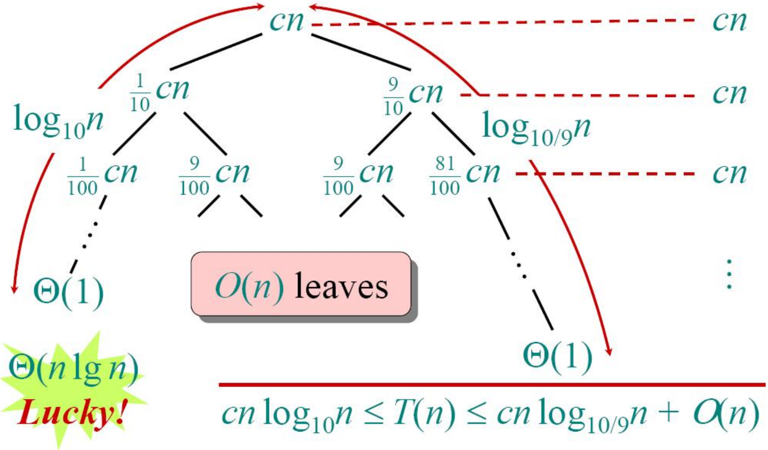
Best Case:

- If we are lucky, PARTITION splits the array evenly
- $T(n) = 2T(n/2) + \theta(n) = \theta(nlgn)$
- Let $L(n)$ denote the running time when we are lucky
- Versus $U(n)$ - the worst-case running time of $\theta(n^2)$

Almost Best Case:

- What if the split is not even?
- Say, it is $\frac{1}{10}:\frac{9}{10}$
- $T(n) = T(\frac{1}{10}n) + T(\frac{9}{10}n) + \theta(n)$
- What is the solution to this recurrence?

Analysis of "almost best"



More Intuition

- Suppose that QUICKSORT is alternately lucky, unlucky, lucky, unlucky, lucky, ...
- $$\begin{aligned} L(n) &= 2U(n/2) + \theta(n) \\ U(n) &= L(n-1) + \theta(n) \end{aligned}$$
- Solving further:
- $$\begin{aligned} L(n) &= 2(L(n/2 - 1/2) + \theta(n/2)) + \theta(n) \\ &= 2L(n/2 - 1/2) + \theta(n) \\ &= \theta(nlgn) \text{ - Lucky!!!} \end{aligned}$$
- How can we make sure QUICKSORT is *usually* lucky?

Randomized Quicksort

Basic Idea: Partition around a *random* element

- Running time is independent of 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 now only by the output of a random-number generator

Randomized Quicksort Analysis

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

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

- Each of these $k : n-k-1$ partitions ($k \in \{0, 1, \dots, n-1\}$ is equally likely, assuming distinct elements)
- So: $E[T(n)] = \frac{1}{n}\sum_{k=0}^{n-1}\{E[T(k)] + E[T(n-k-1)] + \theta(n)\}$

Randomized Quicksort Analysis Continued

Continuing:

$$
\begin{aligned}
E[T(n)] &= \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(k)] + E[T(n-k-1)] + \theta(n)\}\\
&= \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(k)] + E[T(n-k-1)]\} + \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\theta(n)\}\\
&= \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(k)] + E[T(n-k-1)]\} + \tfrac{1}{n}\cdot n\cdot\theta(n)\}\\
&= \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(k)] + E[T(n-k-1)]\} + \theta(n)\}\\
&= \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(k)]\} + \tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(n-k-1)]\} + \theta(n)\\
&\quad \textit{summations have identical terms}\\
&= \tfrac{2}{n}\textstyle\sum_{k=0}^{n-1}\{E[T(k)]\} + \theta(n)
\end{aligned}
$$

What do we do now?

Randomized Quicksort Analysis Continued

- The $k = 0, 1$ terms can be absorbed in the $\theta(n)$
- So: $E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1}\{E[T(k)]\} + \theta(n)$

- Guess: $E[T(n)] \in O(nlgn)$
- By induction, need to find $a > 0$ s.t. $E[T(n)] \leq a \cdot n \cdot lgn$
- Use the fact that $\sum_{k=2}^{n-1} k \cdot lgk \leq \frac{1}{2}n^2 \cdot lgn - \frac{1}{4}n^2$ (integration technique bounds this summation)

- Then, using the substitution/induction technique:

$$\begin{aligned} E[T(n)] &\leq \frac{2}{n}\sum_{k=2}^{n-1} a \cdot k \cdot lgk + \theta(n) \\ &= \frac{2a}{n}(\frac{1}{2}n^2 \cdot lgn - \frac{1}{4}n^2) + \theta(n) \\ &= a \cdot n \cdot lgn - (\frac{an}{2} - \theta(n)) \\ &\leq a \cdot n \cdot lgn \end{aligned}$$

- Note: a needs to be large enough so that $\frac{an}{2}$ dominates $\theta(n)$

Final Word on Quicksort

- Useful general-purpose algorithm
- Typically over twice as fast as mergesort
- Can benefit substantially from code tuning
- Behaves well even with caching and virtual memory