CMPS 6610 – Fall 2018

Quicksort

Carola Wenk

Slides courtesy of Charles Leiserson with additions by Carola Wenk

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).
- We are going to perform an expected runtime analysis on randomized quicksort

Quicksort: Divide and conquer

Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

$\leq x$	x	$\geq x$

2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

Partitioning subroutine

PARTITION(A, p, q) $\rhd$ $A[p\,.\,.\,q]$

 $x \leftarrow A[p]$ $\rhd$ pivot $= A[p]$

 $i \leftarrow p$

 for $j \leftarrow p + 1$ **to** q

  do if $A[j] \le x$

   then $i \leftarrow i + 1$

    exchange $A[i] \leftrightarrow A[j]$

 exchange $A[p] \leftrightarrow A[i]$

 return i

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

Invariant:

x	$\le x$	$\ge x$	?
p	i	j	q

Example of partitioning

6	10	13	5	8	3	2	11
i	j						

Example of partitioning

6	10	13	5	8	3	2	11
i		j					

Example of partitioning

6	10	13	5	8	3	2	11
i			j				

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i		j				

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

i $\longrightarrow j$

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11

i $\longrightarrow j$

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

i j

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
		i				j	

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11

$\longrightarrow i \qquad\qquad\qquad j$

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

6	5	3	2	8	13	10	11
			i				$\longrightarrow j$

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11

i $\quad\quad\quad\quad\quad\quad$ $\longrightarrow j$

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11
			i				

Pseudocode for quicksort

QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT$(A, p, q-1)$
            QUICKSORT$(A, q+1, r)$

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

Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.

Deterministic Algorithms

Runtime for deterministic algorithms with input size n:

- Worst-case runtime
  - ➔ Attained by one input of size n
- Best-case runtime
  - ➔ Attained by one input of size n
- Average runtime
  - ➔ Averaged **over all possible inputs** of size n

Worst-case of quicksort

QUICKSORT(A, p, r)
  if $p < r$
    then $q \leftarrow$ PARTITION(A, p, r)
      QUICKSORT(A, p, $q-1$)
      QUICKSORT(A, $q+1$, r)

- Let $T(n)$ = worst-case running time on an array of n elements.
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.
- $T(n) = T(0) + T(n-1) + \Theta(n)$

Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

$T(n)$

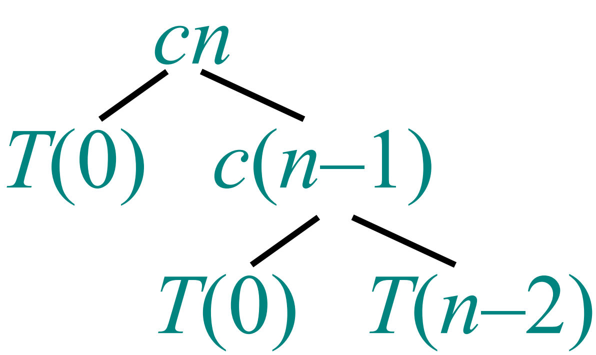
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

$$\begin{array}{ccc} & cn & \\ \swarrow & & \searrow \\ T(0) & & T(n–1) \end{array}$$

Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

cn

$T(0)$ $\quad$ $c(n–1)$

$T(0)$ $\quad$ $c(n–2)$

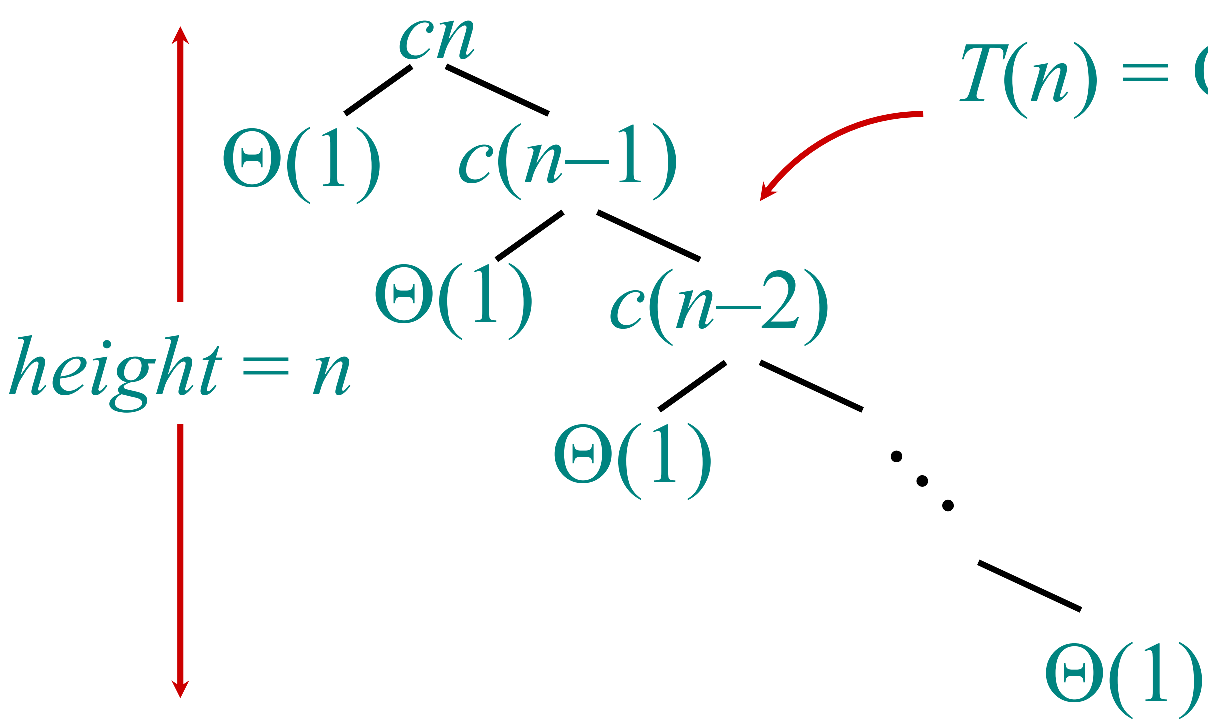
$T(0)$ $\quad$ $\ddots$

$\Theta(1)$

Worst-case recursion tree

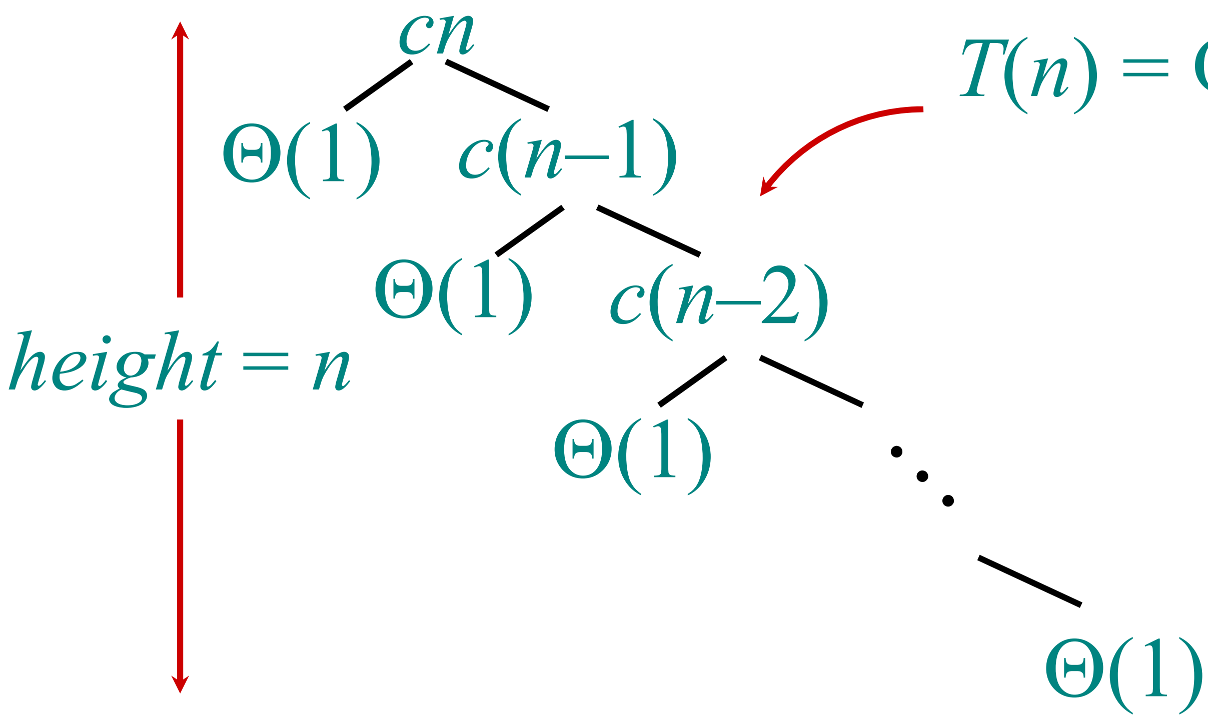
$$T(n) = T(0) + T(n-1) + cn$$

$$T(n) = \Theta\left(\sum_{k=1}^{\text{height}} k\right)$$



Worst-case recursion tree

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

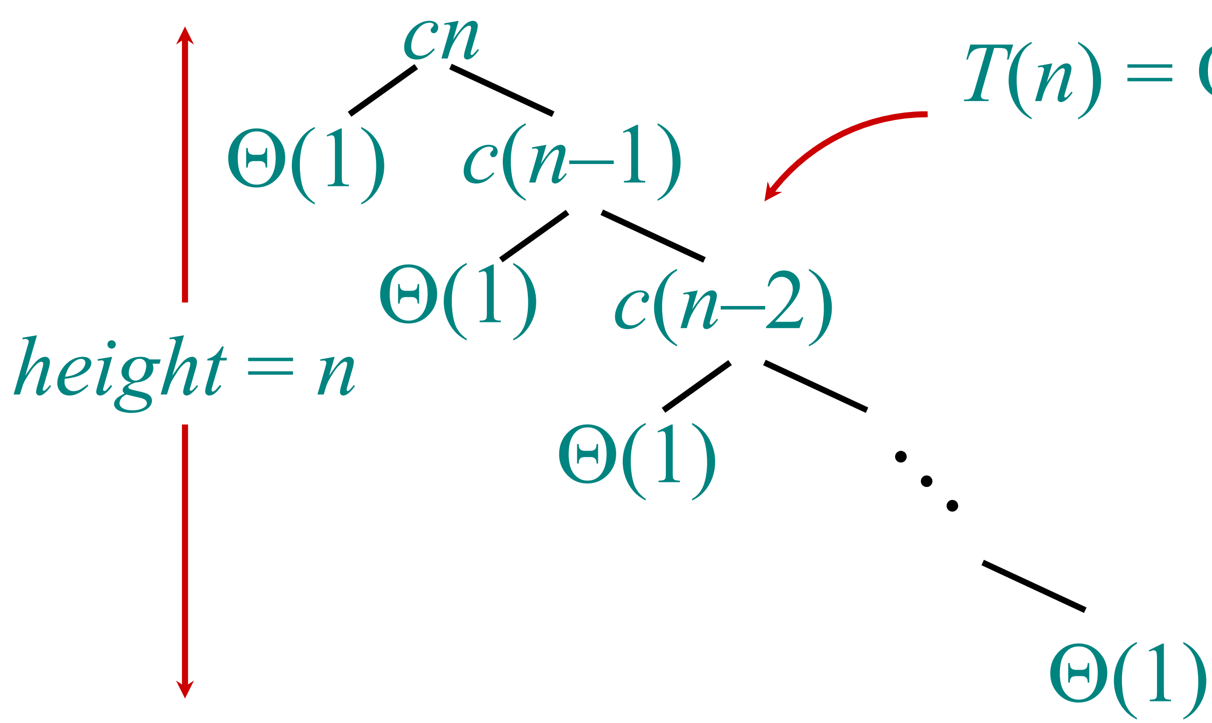


$$T(n) = \Theta\left(\sum_{k=1}^{n} k\right)$$

(arithmetic series)

Worst-case recursion tree

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



$$T(n) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

(arithmetic series)

Deterministic Algorithms

Runtime for deterministic algorithms with input size n:

- Worst-case runtime: $O(n^2)$
  - ➔ Attained by input: [1,2,3,…,n] or [n, n-1,…,2,1]
- Best-case runtime
  - ➔ Attained by one input of size n
- Average runtime
  - ➔ Averaged **over all possible inputs** of size n

Best-case analysis

(For intuition only!)

If we're lucky, PARTITION splits the array evenly:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

(same as merge sort)

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

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

What is the solution to this recurrence?

Analysis of "almost-best" case

$T(n)$

Analysis of “almost-best” case

$$cn$$

$$T\left(\tfrac{1}{10}n\right) \qquad\qquad T\left(\tfrac{9}{10}n\right)$$

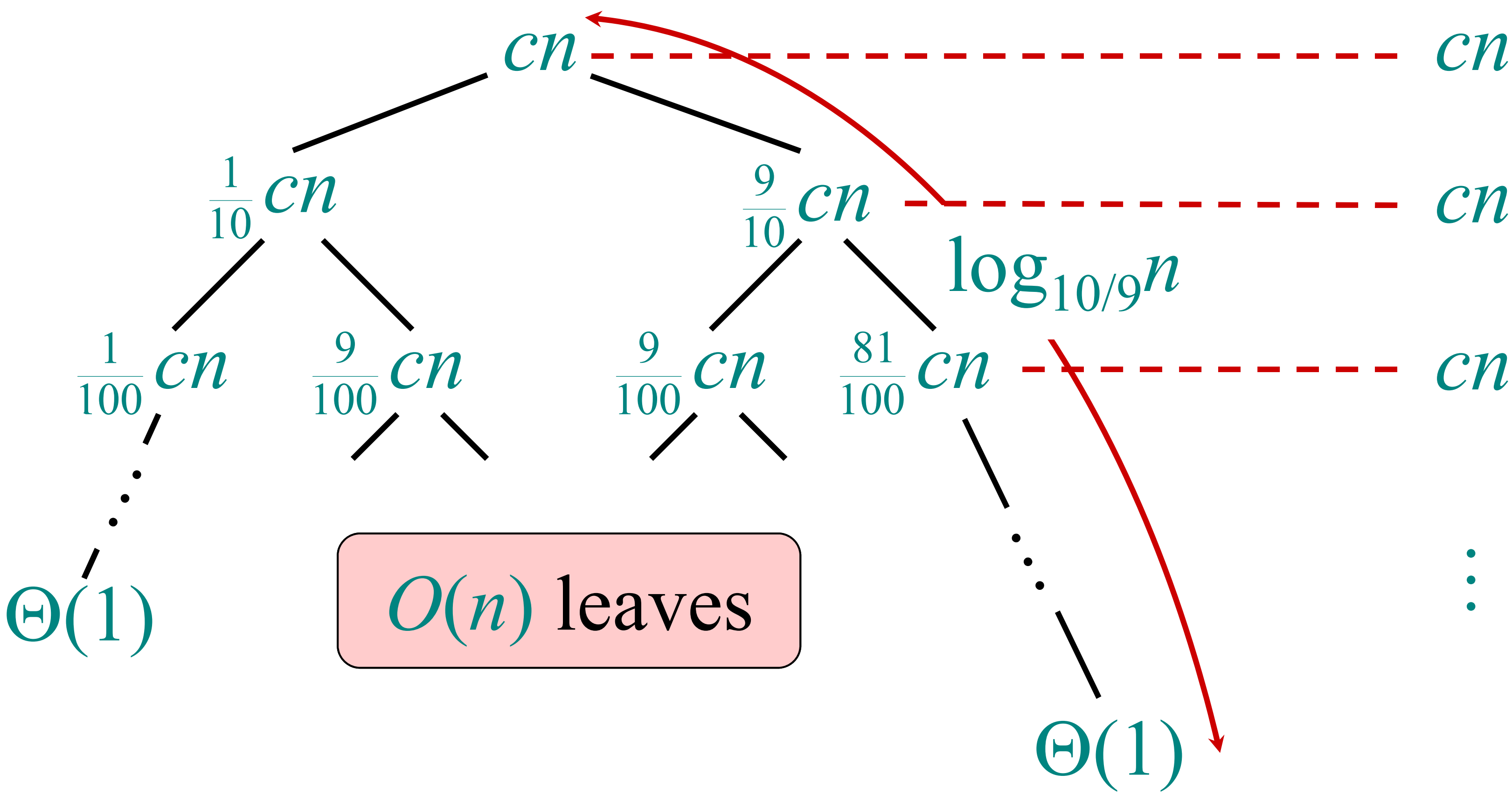
Analysis of “almost-best” case

$$cn$$

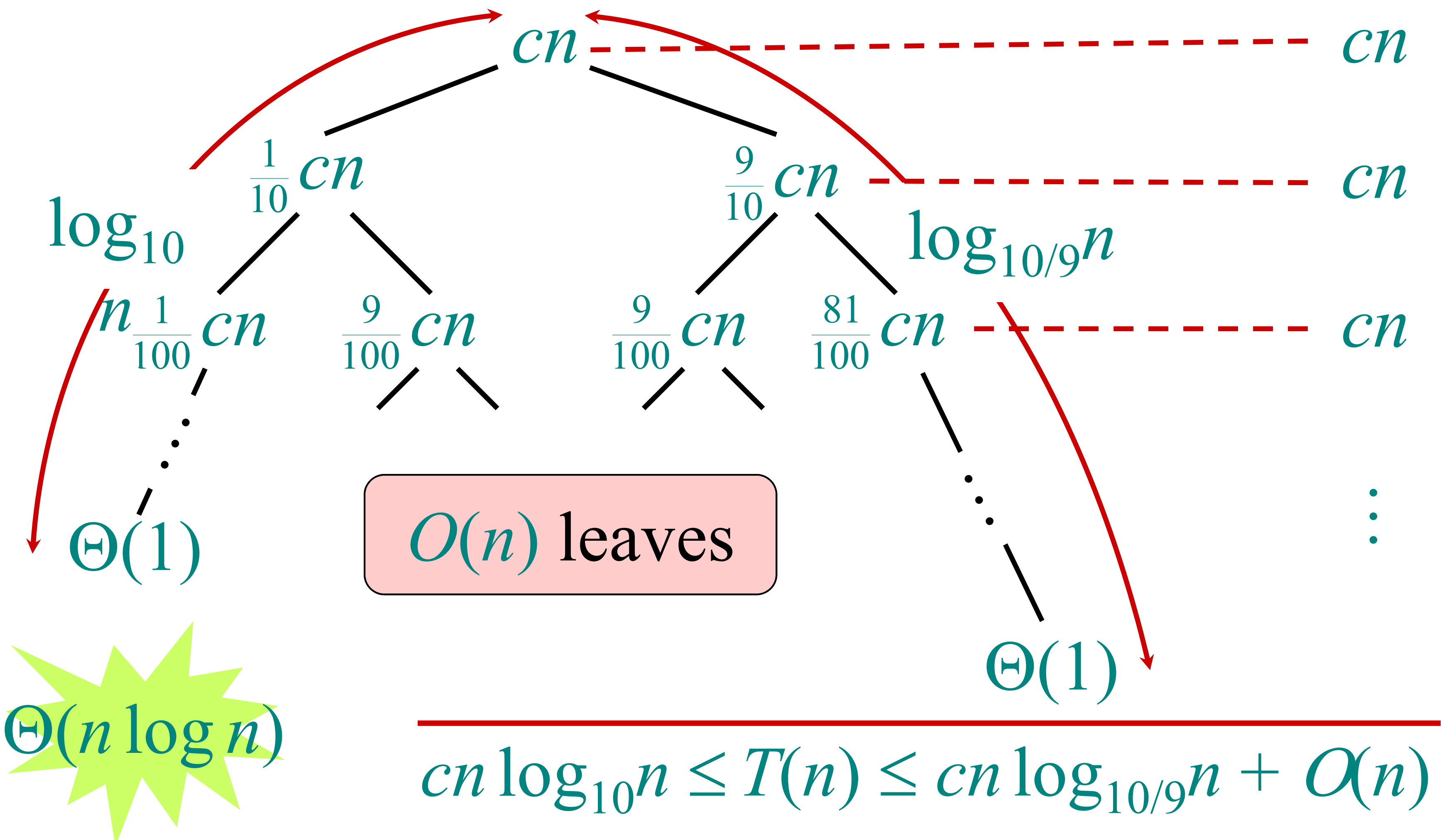
$$\tfrac{1}{10}cn \qquad \tfrac{9}{10}cn$$

$$T\left(\tfrac{1}{100}n\right) \; T\left(\tfrac{9}{100}n\right) \qquad T\left(\tfrac{9}{100}n\right) \; T\left(\tfrac{81}{100}n\right)$$

Analysis of "almost-best" case



Analysis of "almost-best" case



$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$

Deterministic Algorithms

Runtime for deterministic algorithms with input size n:

- Worst-case runtime: $O(n^2)$
  - ➔ Attained by input: $[1,2,3,\ldots,n]$ or $[n, n\text{-}1,\ldots,2,1]$
- Best-case runtime: $O(n \log n)$
  - ➔ Attained by input of size n that splits evenly or $\frac{1}{10} : \frac{9}{10}$ at every recursive level
- Average runtime
  - ➔ Averaged **over all possible inputs** of size n

Average Runtime

- What kind of inputs are there?
  - Do $[1,2,\ldots,n]$ and $[5,6,\ldots,n+5]$ cause different behavior of Quicksort?
  - No. Therefore it suffices to only consider all permutations of $[1,2,\ldots,n]$.
- How many inputs are there?
  - There are $n!$ different permutations of $[1,2,\ldots,n]$

$\Rightarrow$ Average over all $n!$ input permutations.

Average Runtime: Quicksort

- The average runtime averages runtimes over all $n!$ different input permutations
- One can show that the average runtime for Quicksort is $O(n \log n)$
- Disadvantage of considering average runtime:
  - There are still worst-case inputs that will have the worst-case runtime of $O(n^2)$
  - Are all inputs really equally likely? That depends on the application

$\Rightarrow$ **Better:** Use a randomized algorithm

Randomized quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order. It depends on a probabilistic experiment (sequence *s* of numbers obtained from random number generator)

  ⇒ Runtime is a random variable (maps sequence of random numbers to runtimes)
- **Expected runtime** = expected value of runtime random variable
- 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 sequence *s* of random numbers.

Quicksort Runtimes

- Best case runtime $T_{best}(n) \in O(n \log n)$
- Worst case runtime $T_{worst}(n) \in O(n^2)$

- Average runtime $T_{avg}(n) \in O(n \log n)$
- Better even, the expected runtime of **randomized quicksort** is $O(n \log n)$

Probability

- Let S be a **sample space** of possible outcomes.
- $E \subseteq S$ is an **event**
- The (Laplacian) **probability of** E is defined as $P(E)=|E|/|S|$
  $\Rightarrow P(s)=1/|S|$ for all $s \in S$

Note: This is a special case of a probability distribution. In general $P(s)$ can be quite arbitrary. For a loaded die the probabilities could be for example $P(6)=1/2$ and $P(1)=P(2)=P(3)=P(4)=P(5)=1/10$.

Example: Rolling a (six-sided) die

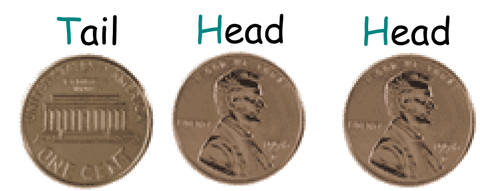
- $S = \{1,2,3,4,5,6\}$
- $P(2) = P(\{2\}) = 1/|S| = 1/6$
- Let $E = \{2,6\} \Rightarrow P(E) = 2/6 = 1/3 = P($rolling a 2 or a $6)$

In general: For any $s \in S$ and any $E \subseteq S$

- $0 \leq P(s) \leq 1$
- $\sum_{s \in S} P(s) = 1$
- $P(E) = \sum_{s \in E} P(s)$

Random Variable

- A random variable X on S is a function from S to $\mathbb{R}$,
  $X: S \rightarrow \mathbb{R}$



Example 1: Flip coin three times.

- S = {HHH, HHT, HTH, HTT, THH, THT, TTH, TTT}
- Let $X(s)$ = # heads in s
  $\Rightarrow X(\text{HHH}) = 3$
  $X(\text{HHT})=X(\text{HTH})=X(\text{THH}) = 2$
  $X(\text{TTH})=X(\text{THT})=X(\text{HTH}) = 1$
  $X(\text{TTT}) = 0$

Example 2: Play game: Win \$5 when getting HHH, pay \$1 otherwise

- Let $Y(s)$ be the win/loss for the outcome s
  $\Rightarrow Y(\text{HHH}) = 5$
  $Y(\text{HHT}) = Y(\text{HTH}) = \ldots = -1$

What is the *average* win/loss?

Expected Value

- The **expected value** of a random variable $X\colon S \to \mathbb{R}$ is defined as

$$E(X) = \sum_{s\in S} P(s) \cdot X(s) = \sum_{x\in\mathbb{R}} P(\{X=x\}) \cdot x$$

Notice the similarity to the **arithmetic mean** (or **average**).

Example 2 (continued):

$$E(Y) = \sum_{s\in S} P(s) \cdot Y(s) = P(HHH) \cdot 5 + P(HHT) \cdot (-1) + P(HTH) \cdot (-1) + P(HTT) \cdot (-1)$$
$$+ P(THH) \cdot (-1) + P(THT) \cdot (-1) + P(TTH) \cdot (-1) + P(TTT) \cdot (-1)$$
$$= P(HHH) \cdot 5 + \sum_{s\in S\setminus\{HHH\}} P(s) \cdot (-1) = 1/2^3 \cdot 5 + 7 \cdot 1/2^3 \cdot (-1)$$
$$= (5-7)/2^3 = -2/8 = -1/4$$

$$= \sum_{y\in\mathbb{R}} P(\{Y=y\}) \cdot y = P(HHH) \cdot 5 + P(\{Y=-1\}) \cdot (-1) = 1/2^3 \cdot 5 + 7/2^3 \cdot (-1) = -1/4$$

$\Rightarrow$ The average win/loss is $E(Y) = -1/4$

Theorem (Linearity of Expectation):

Let X, Y be two random variables on S. Then the following holds:

$$E(X+Y) = E(X) + E(Y)$$

Proof: $E(X+Y) = \sum_{s\in S} P(s) \cdot (X(s)+Y(s)) = \sum_{s\in S} P(s) \cdot X(s) + \sum_{s\in S} P(s) \cdot Y(s) = E(X) + E(Y)$ $\square$

Randomized algorithms

- Allow random choices during the algorithm
- Sample space S = {all sequences of random choices}
- The runtime $T: S \rightarrow \mathbf{R}$ is a random variable. The runtime $T(s)$ depends on the particular sequence s of random choices.

$\Rightarrow$ Consider the **expected runtime** $\mathrm{E}(T)$

Expected Runtime Analysis for Quicksort

- Assume all elements in the input array are distinct
- Runtime is proportional to $\Theta(n + X)$, where X= #comparisons made in PARTITION routine
- Comparisons are made between a pivot (in some recursive call) and another array element

Expected Runtime Analysis for Quicksort

- Let $z_1, \dots, z_n$ be the elements of the input array in sorted (non-decreasing) order
- Let $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$
- Each pair of elements z_i and z_j is compared at most once:
  - One of them has to be the pivot
  - After the PARTITION routine, this pivot has its final position in sorted order and won't be compared in subsequent recursive calls

Expected Runtime Analysis for Quicksort

- Let $X_{ij} = \begin{cases} 1, if\ z_i\ is\ compared\ to\ z_j \\ 0, otherwise \end{cases}$
- X_{ij} is an indicator random variable
- Total # comparisons $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$
- $E(X) = E(\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij})$

  linearity of expectation
- It remains to compute $E(X_{ij})$

Expected Runtime Analysis for Quicksort

- $E(X_{ij}) = 1 \cdot P(z_i \text{ is compared to } z_j)$
  $+ 0 \cdot P(z_i \text{ is not compared to } z_j)$
- It remains to compute: $P(z_i \text{ is compared to } z_j)$
  - If pivot x is chosen such that $z_i < x < z_j$ then z_i and z_j are on different sides of the pivot and won't be compared subsequently
  - If z_i is chosen as a pivot before any other element in Z_{ij} then z_i will be compared to every element in $Z_{ij} \setminus \{z_i\}$

Expected Runtime Analysis for Quicksort

- The argument is symmetric for z_j
- Therefore, z_i and z_j are compared if and only if the first element of Z_{ij} to be chosen as a pivot is z_i or z_j
- $P(z_i \textit{ is compared to } z_j) =$
$P(z_i \textit{ is first pivot from } Z_{ij})$
$+ P(z_j \textit{ is first pivot from } Z_{ij})$
$= \frac{1}{|Z_{ij}|} + \frac{1}{|Z_{ij}|} = \frac{2}{|Z_{ij}|} = \frac{2}{j-i+1}$

Expected Runtime Analysis for Quicksort

- $E(X) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$
  $= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$
  $$< 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k} \in O(2 \sum_{i=1}^{n-1} \log(n-i))$$
  Harmonic number
- Therefore, $E(X) \in O(n \log n)$

Average Runtime vs. Expected Runtime

- Average runtime is averaged over all inputs of a deterministic algorithm.
- Expected runtime is the expected value of the runtime random variable of a randomized algorithm. It effectively "averages" over all sequences of random numbers.
- De facto both analyses are very similar. However in practice the randomized algorithm ensures that not one single input elicits worst case behavior.

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.