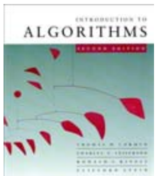
CS 3343 -- Spring 2005



Quicksort

Carola Wenk

Slides courtesy of Charles Leiserson with small changes 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).

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)  ⊳ A[p . . q]
   x ← A[p]         ⊳ pivot = A[p]
   i ← p
   for j ← p + 1 to q
      do if A[j] ≤ x
         then  i ← i + 1
               exchange A[i] ↔ A[j]
   exchange A[p] ↔ A[i]
   return i
```

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

Invariant:

x	$\leq x$	$\geq x$	?

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

Pseudocode for quicksort

```
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)

Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.

Worst-case of quicksort

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

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

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

Worst-case recursion tree

$T(n) = T(0) + T(n–1) + cn$

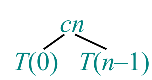
Worst-case recursion tree

$T(n) = T(0) + T(n–1) + cn$

$T(n)$

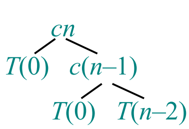
Worst-case recursion tree

$T(n) = T(0) + T(n–1) + cn$



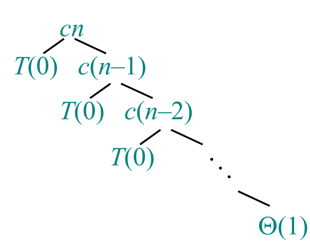
Worst-case recursion tree

$T(n) = T(0) + T(n–1) + cn$



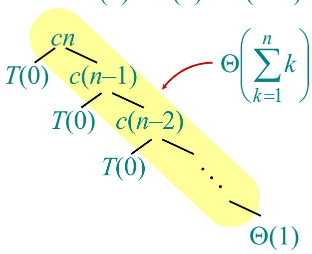
Worst-case recursion tree

$T(n) = T(0) + T(n–1) + cn$



Worst-case recursion tree

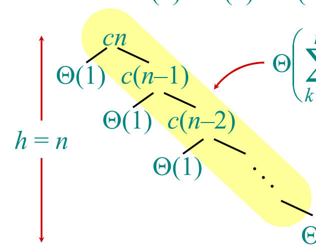
$T(n) = T(0) + T(n-1) + cn$



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

Worst-case recursion tree

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



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

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

Best-case analysis

(For intuition only!)

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

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$ (same as merge sort)

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

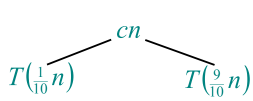
$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

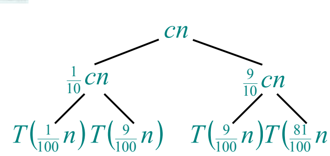
Analysis of "almost-best" case

$T(n)$

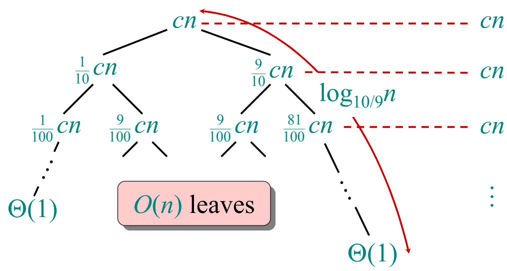
Analysis of "almost-best" case



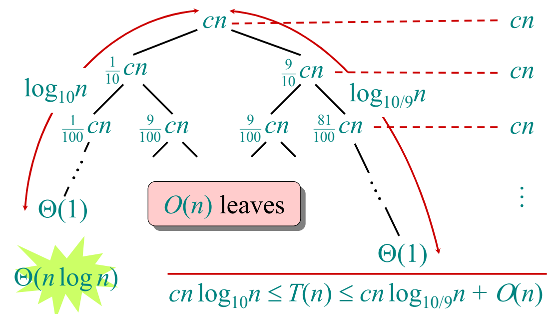
Analysis of "almost-best" case



Analysis of "almost-best" case



Analysis of "almost-best" case



Quicksort Runtimes

- Best case runtime $T_{best}(n) \in O(n \log n)$
- Worst case runtime $T_{worst}(n) \in O(n^2)$
- Worse than mergesort? Why is it called quicksort then?
- Its average runtime $T_{avg}(n) \in O(n \log n)$
- Better even, the expected runtime of **randomized quicksort** is $O(n \log n)$

Average Runtime

The **average runtime** $T_{avg}(n)$ for Quicksort is the average runtime over **all possible inputs** of length n.

- What kind of inputs are there?
- How many inputs are there?

Average Runtime

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

Average Runtime

- Therefore, $T_{avg}(n)$ has to average the runtimes of all $n!$ different input permutations
- Disadvantage of considering average runtime:
  - There are still worst-case inputs that will have a $O(n^2)$ runtime
  - Are all inputs really equally likely ? That depends on the application

$\Rightarrow$ **Better:** Use randomized quicksort

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.

Randomized quicksort analysis

- $T(n)$ = random variable for the running time of randomized quicksort on an input of size n
- $E(T(n))$ = expected value of $T(n)$, the "average runtime" of randomized quicksort

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

Randomized quicksort analysis

Assume that each split is equally likely, with $1/n$ probability.
⇒ The expected runtime (the "average runtime") is

$$E(T(n)) = \frac{1}{n}\sum_{k=0}^{n-1}\left(E(T(k)) + E(T(n-k-1)) + dn\right)$$

Randomized quicksort analysis

Assume that each split is equally likely, with $1/n$ probability.
⇒ The expected runtime (the "average runtime") is

$$E(T(n)) = \frac{1}{n}\sum_{k=0}^{n-1}\left(E(T(k)) + E(T(n-k-1)) + dn\right)$$
$$= \frac{2}{n}\sum_{k=0}^{n-1}\left(E(T(k)) + dn\right)$$

Hairy recurrence

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

(Assume base cases $E(T(0))=E(T(1))=0$.)

Claim: $E[T(n)] \in O(n \log n)$
Prove: $E[T(n)] \le c\,(n \log n - n - 1)$ for some $c > 0$.

Induction (step only)

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1}(ck\log k - ck - c) + dn$$
$$\le \frac{2c\log n}{n}\cdot\frac{(n-1)n}{2} - \frac{4c}{n}\cdot\left(\frac{(n-1)n}{2} - 1\right) - c(n-2) + dn$$
$$\le cn\log n - 2c(n-1) + \frac{4c}{n} - c(n-2) + dn$$
$$\le cn\log n\ ,$$

if c is chosen large enough to dominate dn (e.g., $c=d$ and n large enough).

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.