Lecture 6: Quicksort and Linear-Time Selection

Quicksort: The "dual" of merge sort

Mergesort(A,p,r)**:**
if $p = r$ **then return**
$q \leftarrow \lfloor (p+r)/2 \rfloor$
Mergesort(A,p,q)
Mergesort($A,q+1,r$)
Merge(A,p,q,r)

First call: **Mergesort**($A,1,n$)

5	2	4	7	1	3	2	6

5	2	4	7		1	3	2	6	divide	$\Theta(1)$
2	4	5	7		1	2	3	6	sort	$2T(n/2)$

1	2	2	3	4	5	6	7	merge	$\Theta(n)$

Quicksort: The "dual" of merge sort

```
Mergesort(A,p,r):
if p = r then return
q ← ⌊(p + r)/2⌋
Mergesort(A,p,q)
Mergesort(A,q + 1,r)
Merge(A,p,q,r)

First call: Mergesort(A,1,n)
```

```
Quicksort(A,p,r):
if q ≥ r then return
q = Partition(A,p,r)
Quicksort(A,p,q − 1)
Quicksort(A,q + 1,r)

First call: Quicksort(A,1,n)
```

5	2	4	7	1	3	2	6

2	1	3	2		5	4	7	6

partition $\Theta(n)$

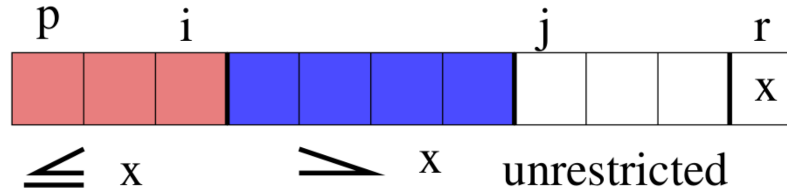
1	2	2	3		4	5	6	7

sort $2T(n/2)$

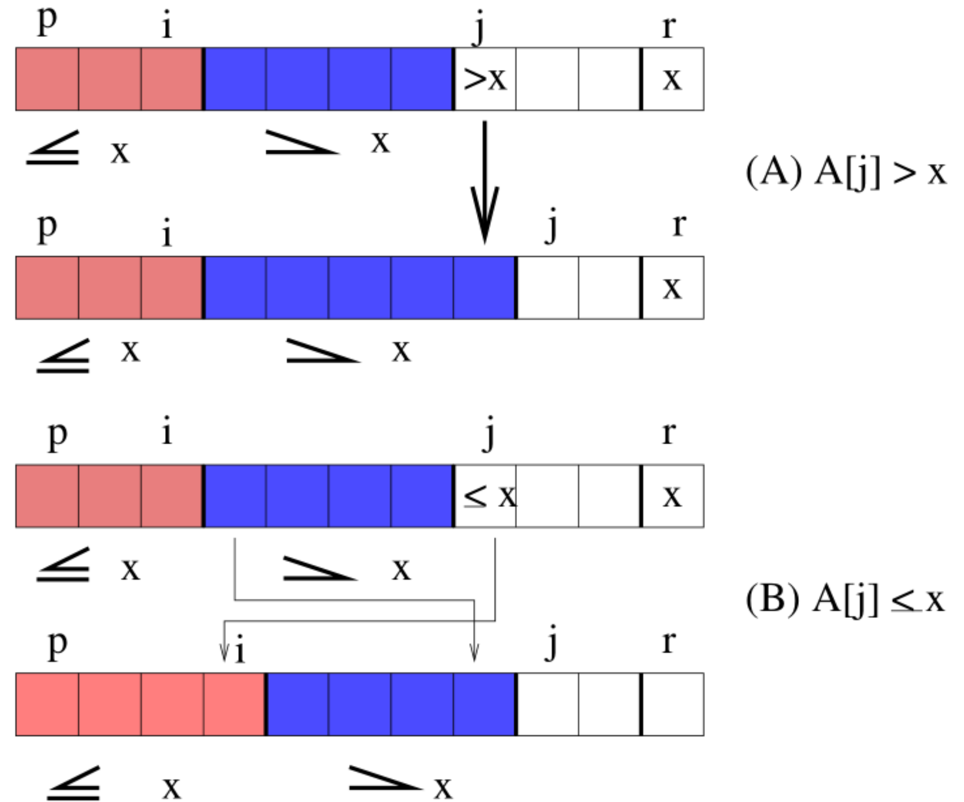
1	2	2	3	4	5	6	7

combine 0

Partition with the last element as the pivot

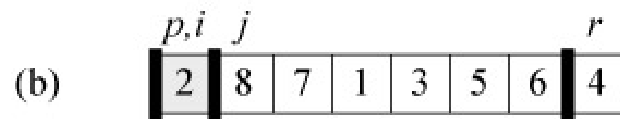


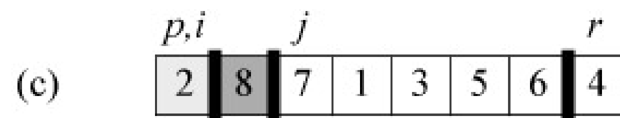

```
Partition(A,p,r):
x ← A[r]
i ← p − 1
for j ← p to r − 1
    if A[j] ≤ x then
        i ← i + 1
        swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

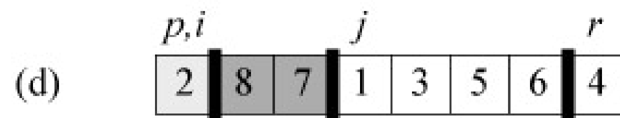


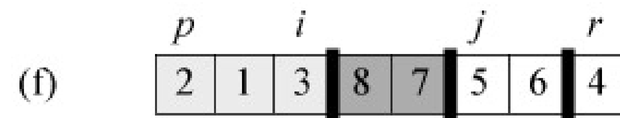
Time: $\Theta(n)$
Working space: $O(1)$ (in-place algorithm)

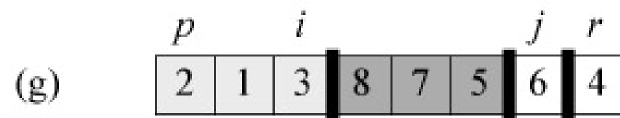
Partition: Example

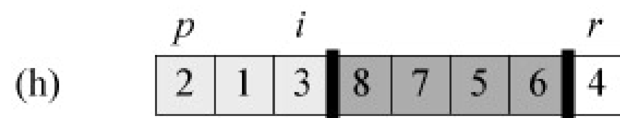


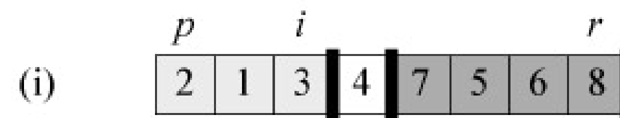












Pivot selection is crucial

Running time.

- [Best case.] Select the median element as the pivot: quicksort runs in $\Theta(n \log n)$ time.
- [Worst case.] Select the smallest (or the largest) element as the pivot: quicksort runs in $\Theta(n^2)$ time.

Q: How to find the median element?

A: Sort?

A: Randomly choose an element as the pivot!

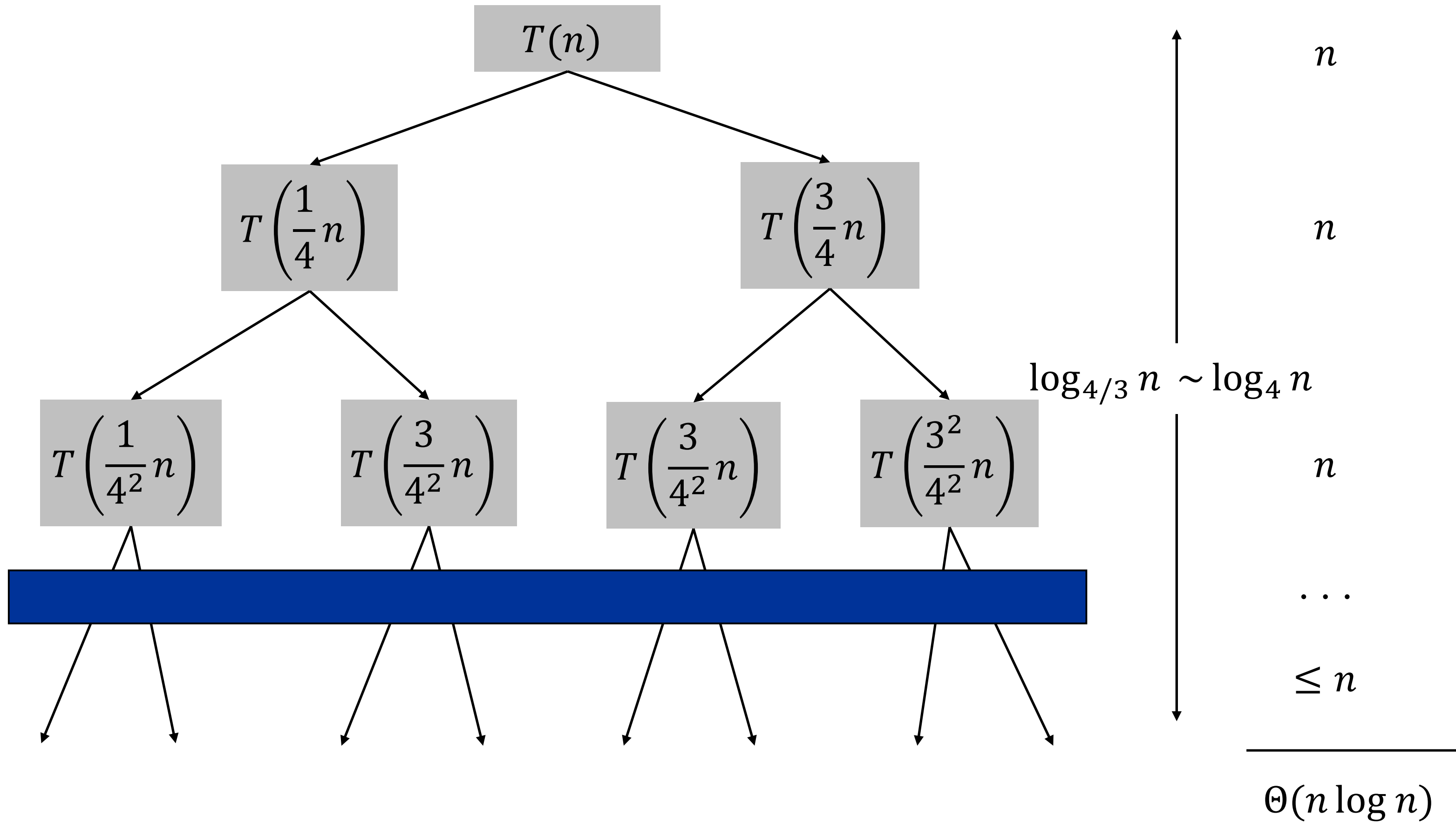
Intuition: A randomly selected pivot "typically" partitions the array as 25% vs 75%, so we have the recurrence

$$T(n) = T\left(\frac{1}{4}n\right) + T\left(\frac{3}{4}n\right) + n$$

which solves to $T(n) = \Theta(n \log n)$. (See next page.)

Solve the recurrence

$$T(n) = T\left(\frac{1}{4}n\right) + T\left(\frac{3}{4}n\right) + n$$



Analysis for Randomized Algorithms

Worst case almost never happens: Every pivot has to be the minimum or maximum, which happens with probability $\Theta(1/n!)$.

Expected running time: $\max_i E_r[T(i,r)]$, where i is any input of size n, r is the random numbers used internally, i.e., this is expected running time (expectation over the random numbers) on the worst possible input.

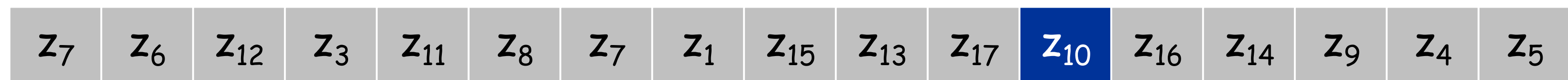
Average case analysis	Expected case analysis
Used for deterministic algorithms	Used for randomized algorithms
Assume the input is chosen randomly from some distribution	Need to work for any input
Depends on assumptions on the input, weaker	Randomization is inherent within the algorithm, stronger

Analysis of quicksort: The binary tree representation

Assumption: All elements are distinct

Note: Running time = Θ(# comparisons)

Relabel the elements from small to large as $z_1, z_2, \ldots, z_n$



first pivot, chosen randomly

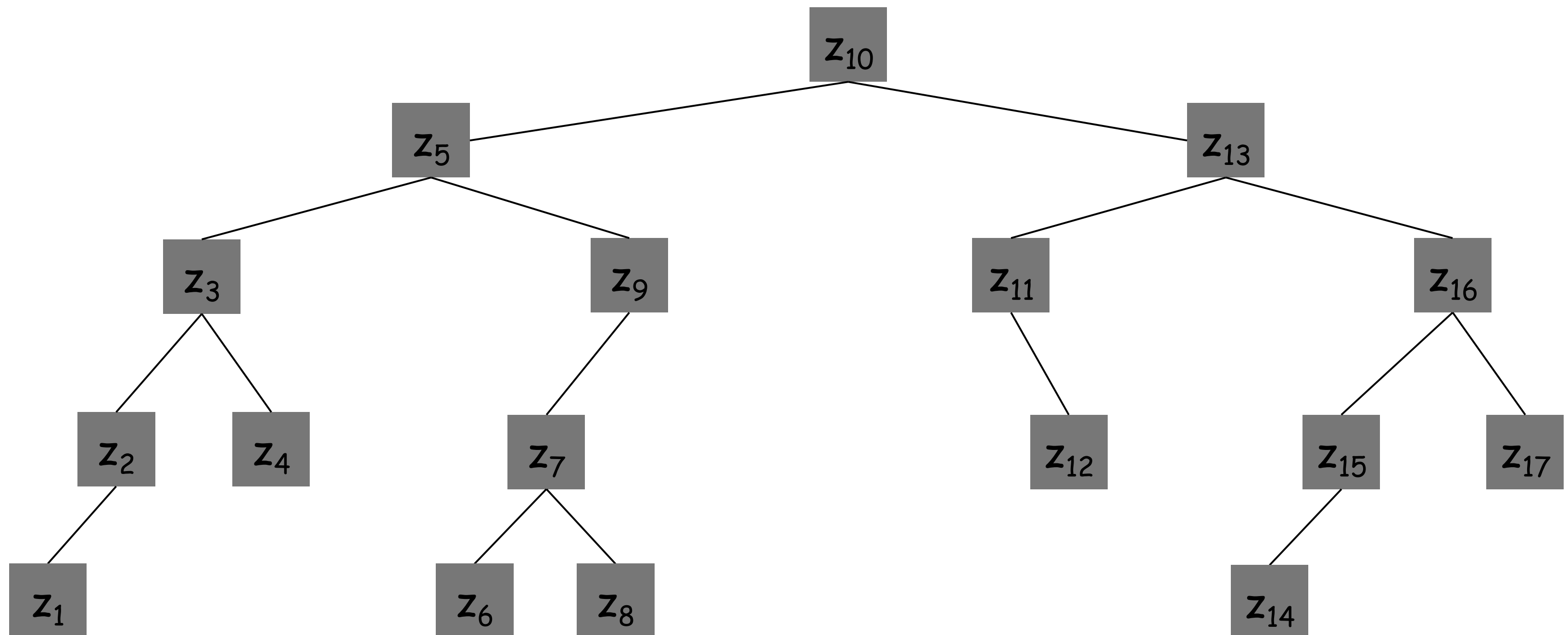
Analysis of quicksort

Observation 1: Element only compared with its ancestors and descendants.

- z_2 and z_7 are compared if their lowest common ancestor (lca) is z_2 or z_7.
- z_2 and z_7 are not compared if their lca is z_3, z_4, z_5, or z_6.
- Other elements cannot be the lca of z_2 and z_7

Observation 2: Every element in $\{z_i, \ldots, z_j\}$ is equally likely to be the lca of z_i and z_j

So, Pr[z_i and z_j are compared] = $2 / (j - i + 1)$.



Analysis of quicksort (continued)

Theorem. Expected # of comparisons is $\Theta(n \log n)$.

Pf.

- Let $X_{ij} = 1$ if z_i is compared with z_j
- # of comparisons is $X = \sum_{i<j} X_{ij}$
- E[# of comparisons] $= \sum_{i<j} E[X_{ij}] = \sum_{i<j} \Pr[z_i$ and z$_j$ are compared]

	$j = 2$	3	4	…	n	
$i = 1$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	…	$\frac{1}{n}$	$O(\log n)$
2		$\frac{1}{2}$	$\frac{1}{3}$	…	$\frac{1}{n-1}$	$O(\log n)$
3			$\frac{1}{2}$	…	$\frac{1}{n-2}$	$O(\log n)$
…	…	…	…	…	…	…
$n-1$					$\frac{1}{2}$	$O(\log n)$
						$O(n \log n)$

Q: Can you show this is $\Theta(n \log n)$?

Quicksort in practice

Why does quicksort work very well in practice?

- $\Theta(n \log n)$ time in expectation on any input
  - Actually, it's $\Theta(n \log n)$ time with very high probability
- Small hidden constants
- Cache-efficient

In practice

- Start with quicksort
- When recursion is too deep (say, $> 10 \log n$), switch to insertion sort or heap sort (discussed later)
- Benefit of quicksort but also with $\Theta(n \log n)$ worst-case guarantee
- Implemented in C++ Standard Template Library (STL)

Randomized Selection

Selection. Given an array A of n distinct elements and an integer i, return the i-th smallest element of A.

Goal: Want to do better than sorting, i.e., linear time.

```
Select(A,p,r,i):
if p = r then return A[p]
q ← Partition(A,p,r)
k ← q − p + 1
if i = k then return A[q]
else if i < k then return Select(A,p,q − 1,i)
else return Select(A,q + 1,r,i − k)

First call: Select(A,1,n,i)
```

Analysis: Textbook method too complicated (involving a lot of math)

- In tutorial: Use indicator random variables, similar to quicksort
- Here: A simple and clever method

Analysis of randomized selection

Theorem: The expected running time of randomized selection is $\Theta(n)$.

Pf: Call a pivot "good" if it is between the 25%- and 75%-percentile of A, otherwise "bad".

- Each good pivot reduces n by at least $1/4$.
- The probability of a random pivot being good is $1/2$.

Let X_i be the running time between the i-th good pivot (not including) and the $(i+1)$-th good pivot (including), $i = 0, 1, 2, \ldots$

- The cost to process each pivot in this stage $\leq \left(\frac{3}{4}\right)^i n$
- E[# pivots in each stage] $= 2$ (waiting time)
- $E[X_i] \leq 2 \cdot \left(\frac{3}{4}\right)^i n$

Expected total running time $\leq E[\sum_i X_i] = \sum_i E[X_i] = O(n)$.

Remark: There is also a deterministic linear-time selection algorithm (Sec 9.3 in textbook)

Space analysis

Observation: The selection algorithm is recursive, but it is a tail-recursion. Can rewrite it without using recursion, so that it uses $O(1)$ working memory.

Note: Good compilers often do this automatically!

Working space of quicksort:

- Quicksort is not a tail-recursion.
- Each level of recursion needs $O(1)$ working space (on system stack).
- There can be $\Theta(n)$ levels of recursion in the worst case.
- But can show that there are only $\Theta(\log n)$ levels of recursion in expectation (analysis is complicated)
- Conclusion: Quicksort uses expected $\Theta(\log n)$ working space.