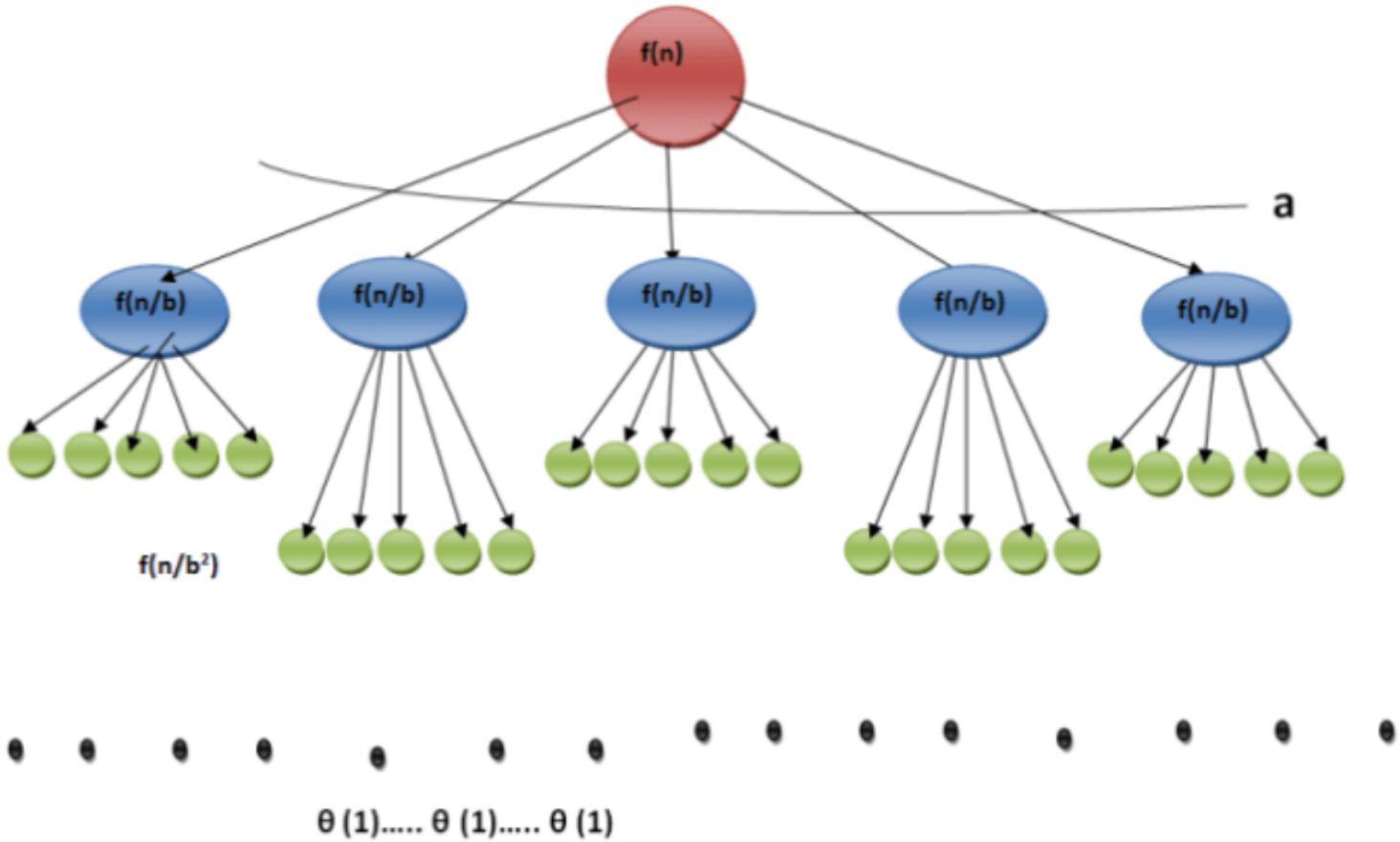
CSCI 3104: Algorithms

Lecture 7: Master Method, Quicksort

Rachel Cox

Department of Computer Science



What will we learn today?

- ❑ Master Method
- ❑ Quicksort
- ❑ Worst-Case Analysis
- ❑ Best-Case Analysis

Intro to Algorithms, CLRS:
Sections 4,5, 7.1, 7.2

Master Method

❖ The **master method** provides a "cookbook" method for solving recurrences of the form:

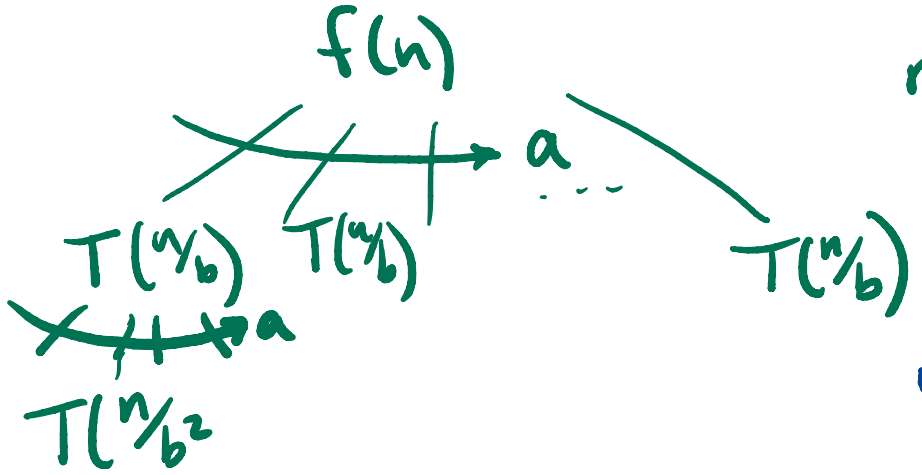
$$T(n) = aT(n/b) + f(n), \text{where } a \geq 1 \text{ and } b > 1.$$

Master Theorem: Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \lg(n)\right)$.
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

Master Method

What does a generic Recursion Tree look like:



number of subproblems = a

subproblem size = n/b

- How many levels?

i^{th} level: subproblem size = $\frac{n}{b^i}$

$\Rightarrow$ last level is when $\frac{n}{b^i} = 1$

$n = b^i$

$i = \log_b n$

$\Rightarrow$ There are $\log_b n$ levels of the tree.

Master Method

number subproblems = a

What does a generic Recursion Tree look like:

Note: $x = a^{\log_a x}$ *

How many leaves?

2^{nd} level $= a^2$

3^{rd} level $= a^3$

$\vdots$

i^{th} level $= a^i$

last level: $\log_b n$

$\Rightarrow$ number of leaves $= a^{\log_b n} = n^{\log_n (a^{\log_b n})}$

$= n^{\log_b n \cdot \log_n a}$

$= n^{\log_b n \cdot \frac{\log_b a}{\log_b n}}$

$= n^{\log_b a}$ } # of leaves

Master Method

What does a generic Recursion Tree look like:

How much work is done at the last level ?

$$\left(n^{\log_b a}\right) * f\left(\frac{n}{b^{\log_b n}}\right) = n^{\log_b a} * f(1)$$

$$\Rightarrow \Theta\left(n^{\log_b a}\right)$$

some constant amount of time to solve the trivial solution

Master Method

1. If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \lg(n)\right)$.
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

1. This means $f(n)$ is polynomially smaller than the number of leaves.
   → Asymptotically, work is done at the leaves
   ⇒ $T(n) = \Theta(n^{\log_b a})$

2. This means that asymptotically $f(n)$ grows at the same rate as the leaves.
   - We know we have $\log_b n$ levels
   - work = num levels * work done per level

   $T(n) = \Theta(\log n * n^{\log_b a})$

Master Method

1. If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \lg(n)\right)$.
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

3. This means that f(n) is polynomially larger than the number of leaves
   $\Rightarrow$ work at root dominates
   $\Rightarrow T(n) = \Theta(f(n))$

Master Method

Example: Use the Master Method to solve the recurrence: $T(n) = 4T\left(\frac{n}{2}\right) + n$

$a = 4$
$b = 2$
$f(n) = n$

$n^{\log_b a} = n^{\log_2 4} = n^2$

$\Rightarrow f(n) = \vartheta(n^{2-\varepsilon})$ where $\varepsilon = 1$

$\Rightarrow$ CASE 1 of M.T. applies $\Rightarrow$ $T(n)$ is $\theta(n^2)$

Example: Use the Master Method to solve the recurrence: $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

$a = 4$
$b = 2$
$f(n) = n^2$

$n^{\log_b a} = n^{\log_2 4} = n^2$

$f(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

$\Rightarrow$ CASE 2 applies

$\Rightarrow$ $T(n)$ is $\Theta(n^2 \log n)$

Master Method

Example: Use the Master Method to solve the recurrence: $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

$a = 4$

$b = 2$

$f(n) = n^3$

$n^{\log_b a} = n^2$

$f(n) = n^{2+1}$

$\Rightarrow f(n) = \Omega(n^{2+\varepsilon})$ where $\varepsilon = 1$

CASE 3 applies.

Represents work at level below the root

Regularity condition:

$af(n/b) \leq cf(n)$ $c<1$

$4\left(\frac{n}{2}\right)^3 = \frac{4}{8}n^3 = \frac{1}{2}n^3$

$\Rightarrow 4(n/2)^3 \leq cn^3$

for $c = 1/2 < 1$

$\Rightarrow T(n) = \Theta(n^3)$

Master Method

Practice on your own.

Example: Use the Master Method to solve the recurrence: $T(n) = 3T\left(\frac{n}{4}\right) + nlg(n)$

$a = 3$
$b = 4$
$f(n) = n\lg(n)$
$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
$\Rightarrow f(n) = \Omega(n^{\log_4 3 + \varepsilon})$
where $\varepsilon \approx 0.207$

Therefore, CASE 3 applies.
We must check the Regularity condition:
$af(n/b) = 3f(n/4)$
$= 3\frac{n}{4}\lg(n/4)$
$\leq \frac{3}{4} n \lg n = cf(n)$; $c = 3/4 < 1$

Since the regularity condition is met, we may conclude:

$T(n) = \Theta(n \lg n)$

Example: Use the Master Method to solve the recurrence: $T(n) = 2T\left(\frac{n}{2}\right) + nlg(n)$

$a = 2$
$b = 2$
$f(n) = n\lg(n)$
$n^{\log_b a} = n^{\log_2 2} = n = O(n)$

$f(n)$ is asymptotically larger, however it is not polynomially larger.

$\Rightarrow$ The recurrence falls somewhere between CASE 2 and CASE 3 so we cannot apply the Master Theorem here.

Quicksort

Divide + Conquer

- the work is done in divide step.

Divide: Partition the array $A[p \ldots r]$ into two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ such that each element of $A[p \ldots q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1 \ldots r]$. Compute the index q as part of this partitioning procedure.

Conquer: Sort the two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ by recursive calls to quicksort.

Combine: Because the subarrays are already sorted, no work is needed to combine them: the entire array $A[p \ldots r]$ is now sorted.

* In practice, quicksort often outperforms Mergesort.

Quicksort

- $QUICKSORT(A, p, r)$

  if $p < r$

  $$\begin{cases} q = PARTITION(A, p, r) \\ QUICKSORT(A, p, q-1) \\ QUICKSORT(A, q+1, r) \end{cases}$$

$PARTITION(A, p, r)$

- $x = A[r]$ "pivot"
- $i = p - 1$
- **for** $j = p$ **to** $r - 1$
  - **if** $A[j] \leq x$
    - $i = i + 1$
    - exchange $A[i]$ with $A[j]$
- exchange $A[i + 1]$ with $A[r]$
- **return** $i + 1$

• choosing the pivot to be the last element

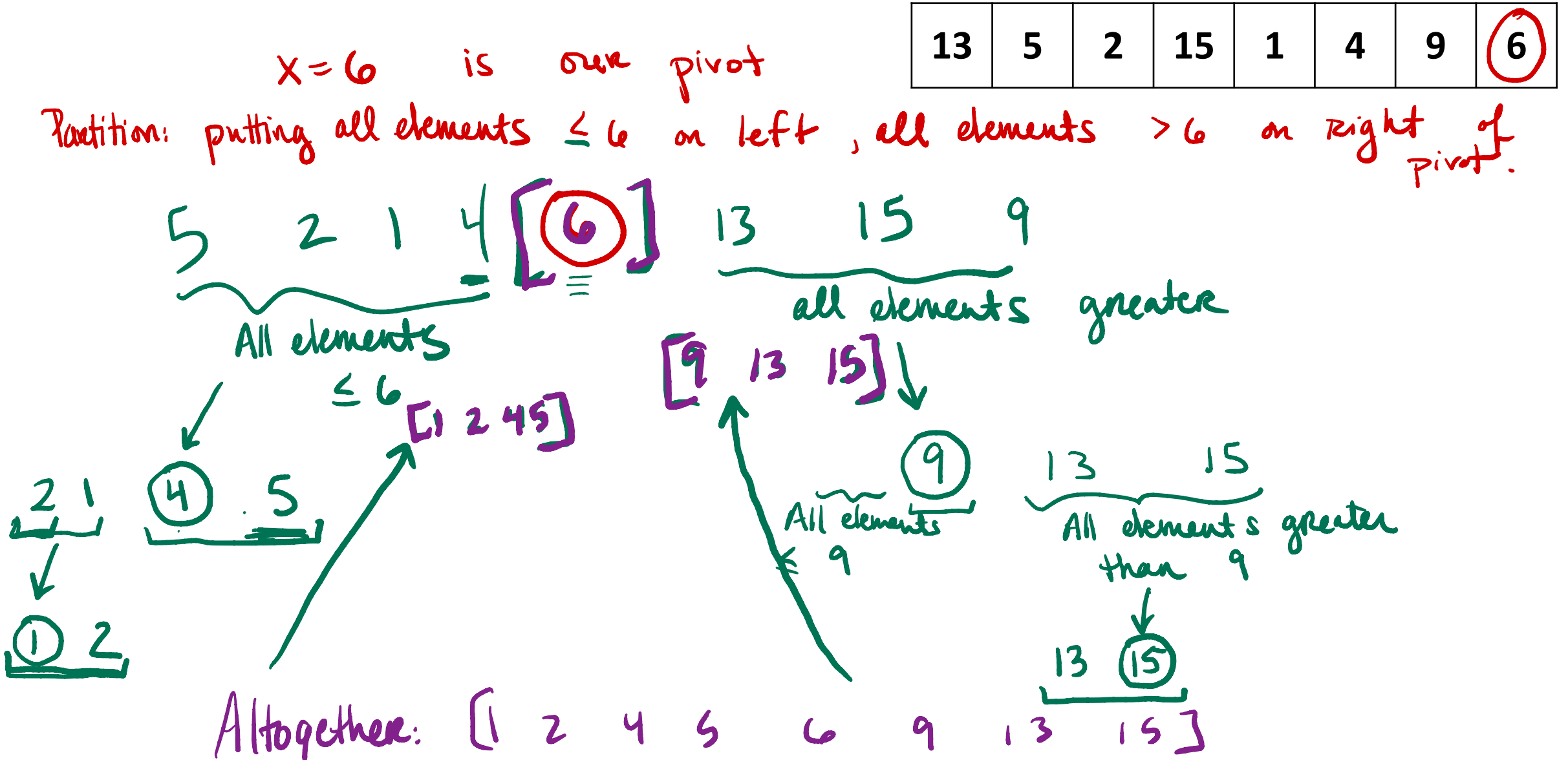
placing pivot in correct location

} location of pivot

Quicksort

Example: Use the QUICKSORT algorithm idea on the following list:

13	5	2	15	1	4	9	6



Quicksort

Example: Completely sort the following list using QUICKSORT:

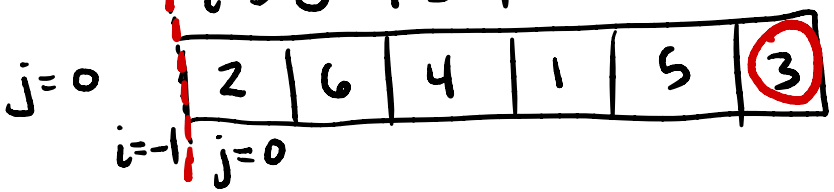
2	6	4	1	5	3

$Q(A, 0, 5)$ (p, r)

call partition$(A, 0, 5)$

$x = 3$

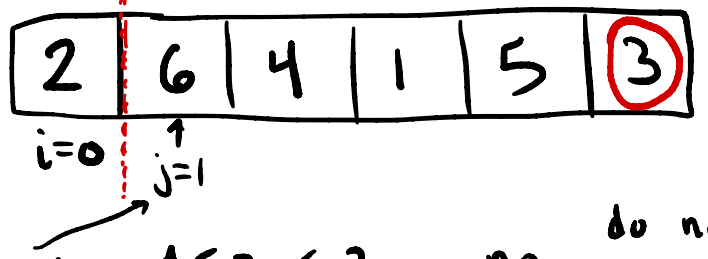
$i = 0 - 1 = -1$



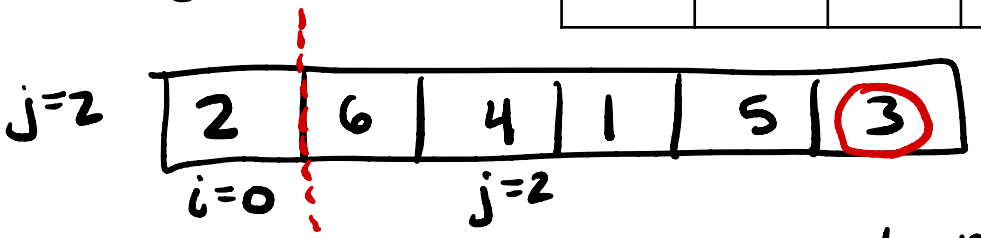
- is $A[0] \le 3$ yes

$i = -1 + 1 = 0$

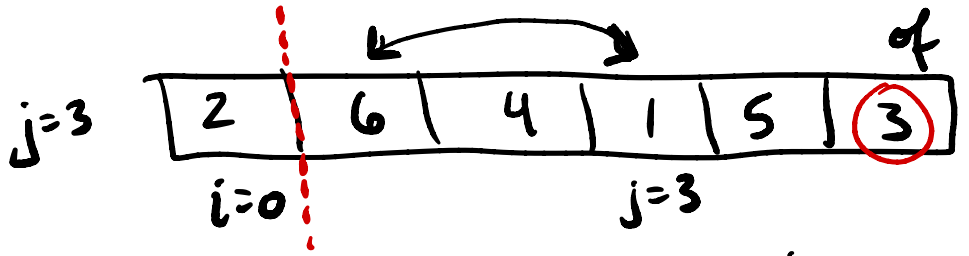
exchange $A[0]$ with $A[0]$



$j = 1$ — is $A[1] \le 3$ no. do nothing 6 is on the correct side of the boundary.



- is $A[2] \le 3$ no. do nothing 4 on correct side of boundary



- is $A[3] \le 3$ yes!

$i = 1$

exchange $A[1]$ with $A[3]$

$j = 4$ — is $A[4] \le 3$ no do nothing

Last step, put pivot in correct location.

⇒ Exchange $A[i+1]$ with $A[r]$ [2 1 3 6 5 4]

From previous page $q = 2$ (location of original pivot $x = 3$)

$q = \text{Partition}(A, 0, 5) = 2$

LEFT

$Q(A, p, r) = Q(A, 0, 1)$

$0 < 1$

$q = \text{Partition}(A, 0, 1) = 0$

$x = A[1]$

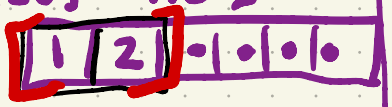
$i = -1$

for $j = 0$ to 0

$A[0] \le 1$ no

exit for loop

exchange $A[0]$ and $A[1]$



return 0

$Q(A, p, q-1) = Q(A, 0, -1)$

Base case

$p < r$

$Q(A, q+1, r) = Q(A, 1, 1)$

Base case

$p < r$

$Q(A, 3, 5)$

$3 < 5$

$q = \text{Partition}(A, 3, 5) = 3$

$x = A[5]$

$i = 2$

for $j = 3$ to 4

$A[3] \le 4$ no

$A[4] \le 4$ no

exit for loop

exchange $A[3]$ and $A[5]$

return 3

$Q(A, 3, 2)$

Base case

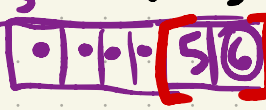
$3 < 2$

$Q(A, 4, 5)$

$4 < 5$

$q = \text{Partition}(A, 4, 5) = 5$

$x = A[5]$



$i = 3$

for $j = 4$ to 4

$A[4] \le 6$ yes

$i = 4$

exchange $A[5]$ and $A[5]$

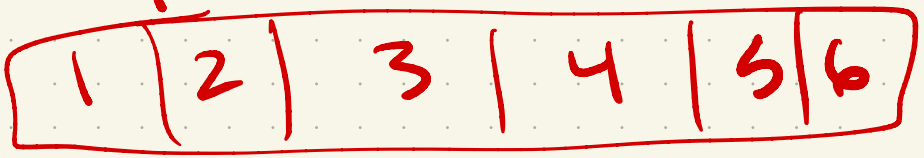
Return 5

$Q(A, 4, 4)$

Base case

$Q(A, 6, 5)$ Base case.

We end up with



Quicksort

Worst-Case Partitioning

- Occurs when the partitioning routine produces one subproblem with $n - 1$ elements and one with 0 elements.

The recursion tree for this scenario looks like:

We will wrap this in Lecture 8.

Quicksort

Worst-Case Partitioning

Example: Find the solution to the recurrence $T(n) = T(n-1) + \Theta(n)$

Quicksort

Best-Case Partitioning

- Most even possible split; two subproblems, each of size no more than $n/2$

The recursion tree for this scenario
looks like:

Quicksort

Best-Case Partitioning

Example: Find the solution to the recurrence $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

Quicksort

Question: What happens when you always have a partition with 4 elements on the left and $n - 5$ elements on the right?

Quicksort

Question: What happens if we have a strategy that gives us a 1:9 split every time?

Quicksort

“Good splits” - splits of the type $\left(\frac{n-1}{b}, (n-1)\left(1-\frac{1}{b}\right)\right)$

“Bad splits” - splits of the type $(n-k, k-1)$ for some constant k.

Quicksort

Question: What if we have a mixture of good and bad splits?

- It turns out that getting a good split a constant fraction of the time is enough to obtain $\Theta\big(nlg(n)\big)$ running time.

Next Time

- Randomized Quicksort
- Average Case Analysis