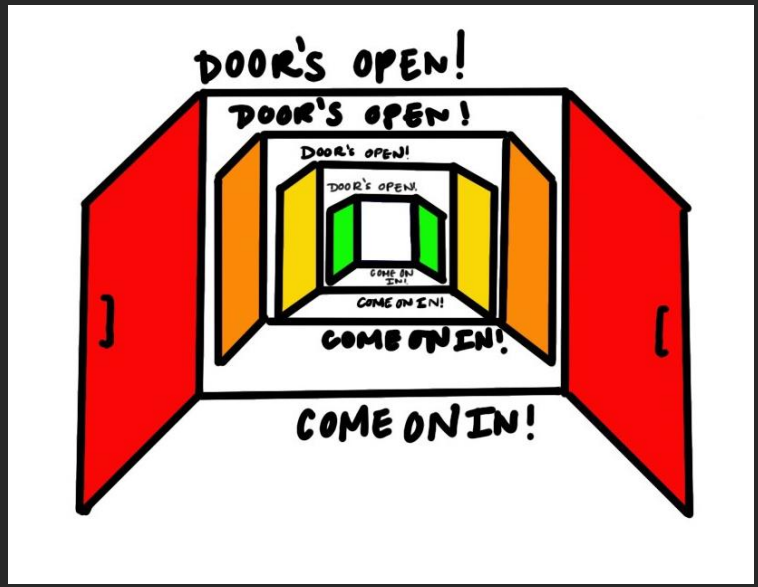
CSCI 3104
Lecture 4: Divide and Conquer Algorithms

Caleb Escobedo

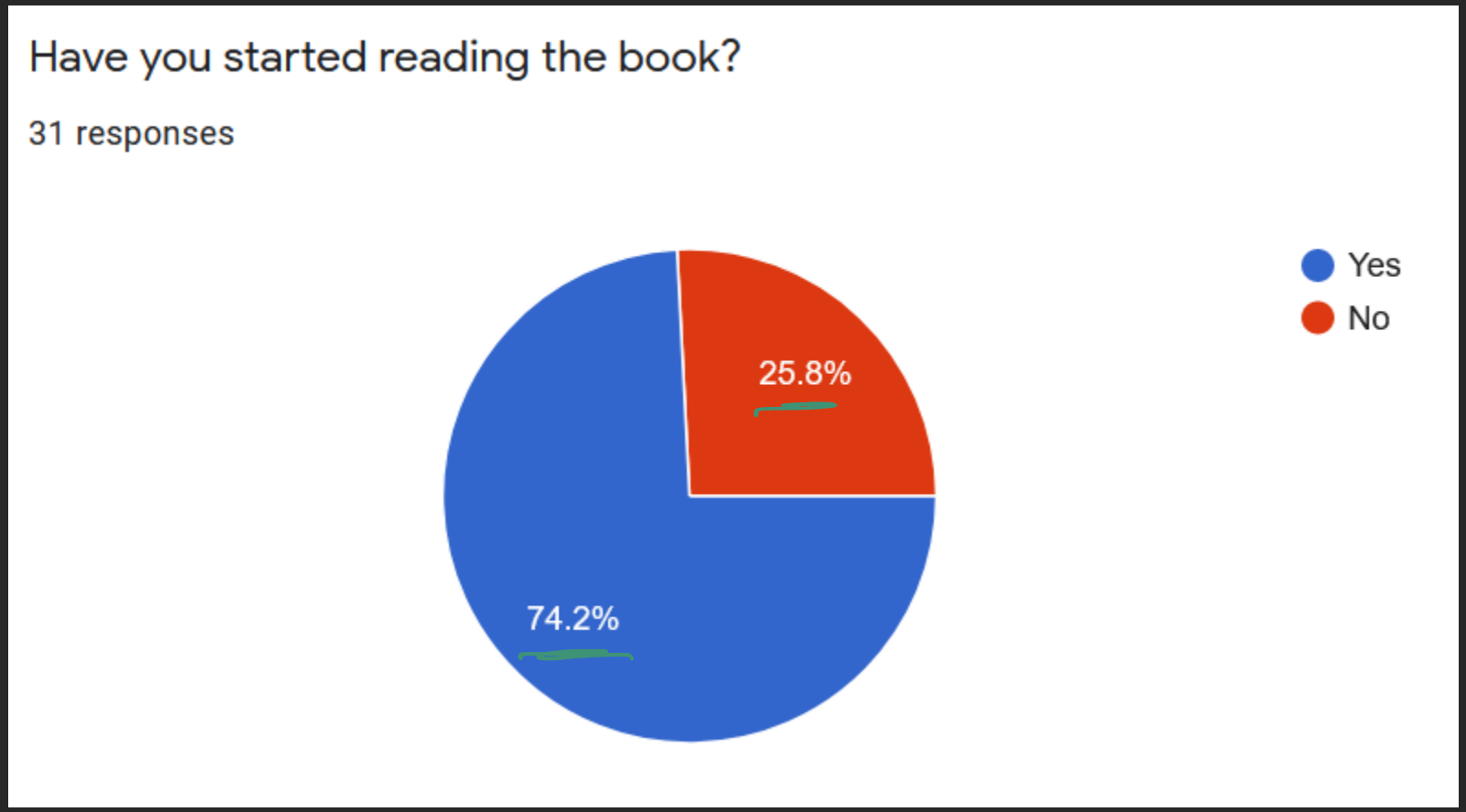
Caleb.Escobedo@colorado.edu

Lecture Outline

- Poll/Admin
- Divide and Conquer Algorithms
- Recurrence Relations
- Quicksort: Introduction
- Correctness
- Runtime



Poll Review

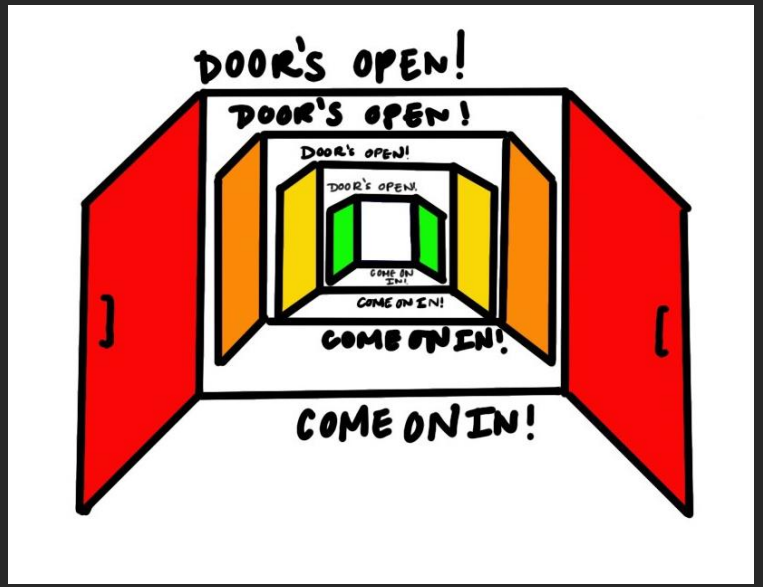

Have you started reading the book?
31 responses
25.8%
74.2%
Yes
No

Administrative

- Reading list is on Canvas!
- Homework 1A is due on Tuesday
- Reading for this this and next lecture
  - Chapter 4 for Divide & Conquer
  - Chapter 7 for Quicksort**
- Daily Practice Problem added to lectures

Lecture Outline

- ~~Poll/Admin~~
- Divide and Conquer Algorithms
- Recurrence Relations
- Quicksort: Introduction
- Correctness
- Runtime



Divide and Conquer Algorithms

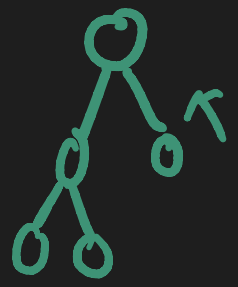
- Three parts:
  - Divide – break problem down
  - Conquer – if small or trivial then solve, else
  - Combine – small solutions into large solution
- Divide and Conquer Algorithm Examples
  - Quicksort - Today! Chapter 7
  - Merge Sort - On your own Chapter 4
  - Binary Search
  - Strassen's Algorithm – Matrix Multiplication

The Structure of Divide and Conquer

Divide and conquer is a recursive approach, and most divide and conquer algorithms have the following structure:

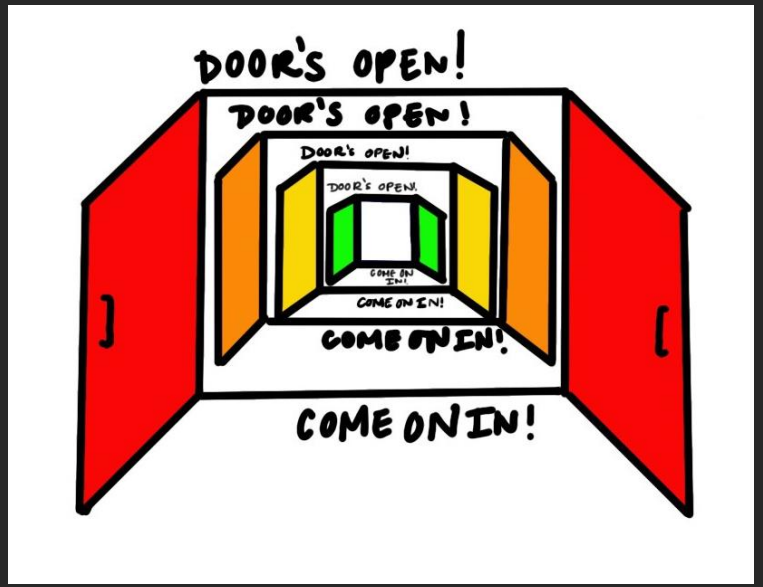
```
def fun(n):
  #base case if n is small or can easily be solved
  if (n==1 or n=='trivial'):
    # solve and return
  else:
    a = fun(n['first half'])
    b = fun(n['second half'])
    ab = combine(a, b)
    return ab
```

base case



Lecture Outline

- ~~Poll/Admin~~
- ~~Divide and Conquer Algorithms~~
- Recurrence Relations
- Quicksort: Introduction
- Correctness
- Runtime



Recurrence Relations

T(n) = runtime

a = # of subproblems

General form for asymptotic analysis:

g[n] = how we split up our input

$$T(n) = a\,T(g[n]) + f(n)$$

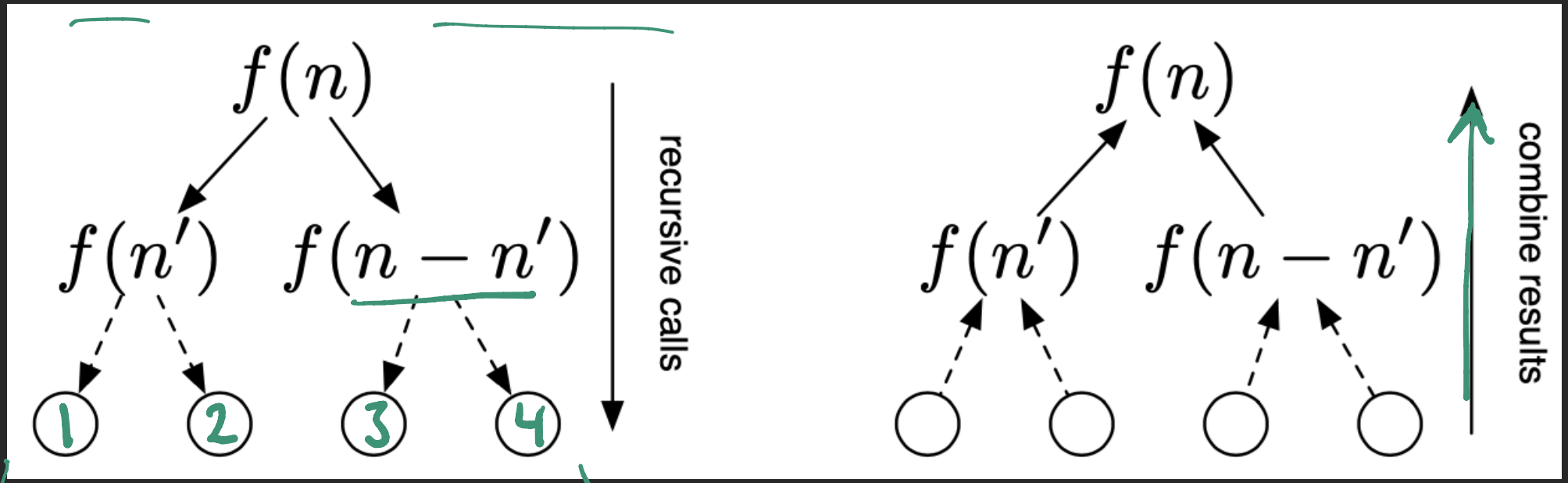
f(n) = cost to split and join

- Unrolling
- Master Method/Theorem
- Recurrence Trees
- See chapter 4.2!

n = [1, 2, 3, 4]

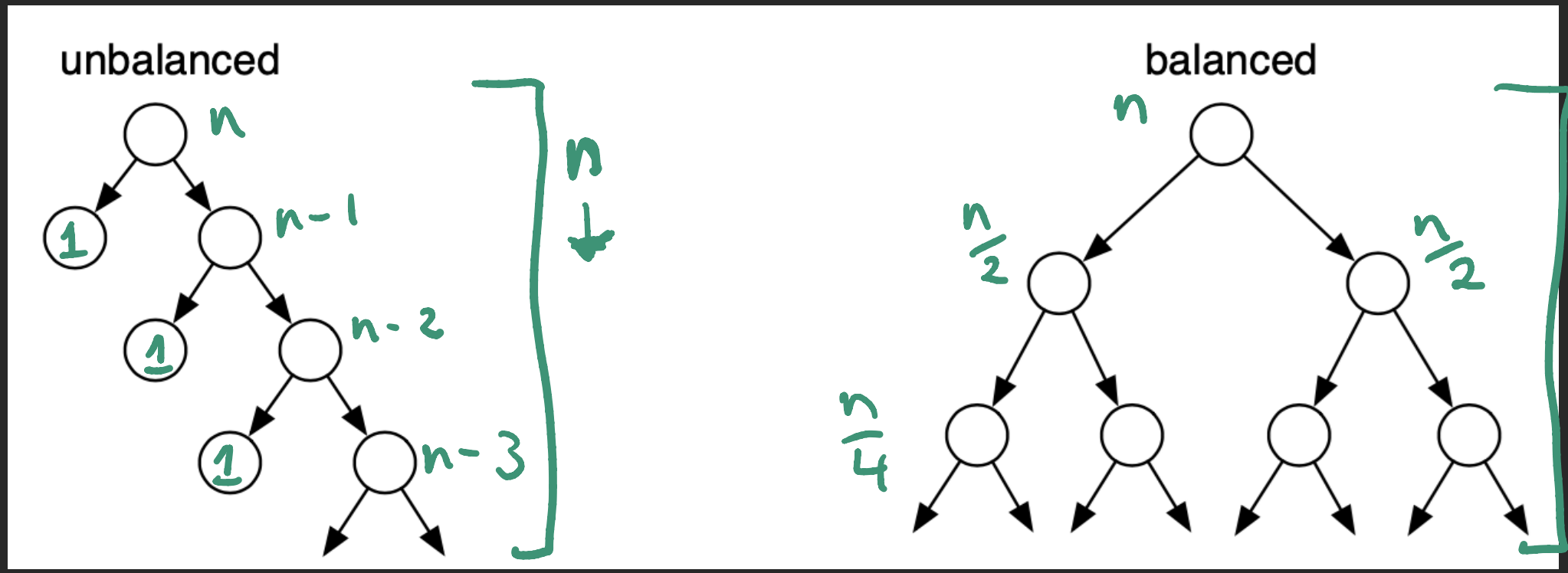
Divide and Conquer

n' = [1] n - n' = [2, 3, 4]



$$T(n) = aT(g[n]) + f(n)$$

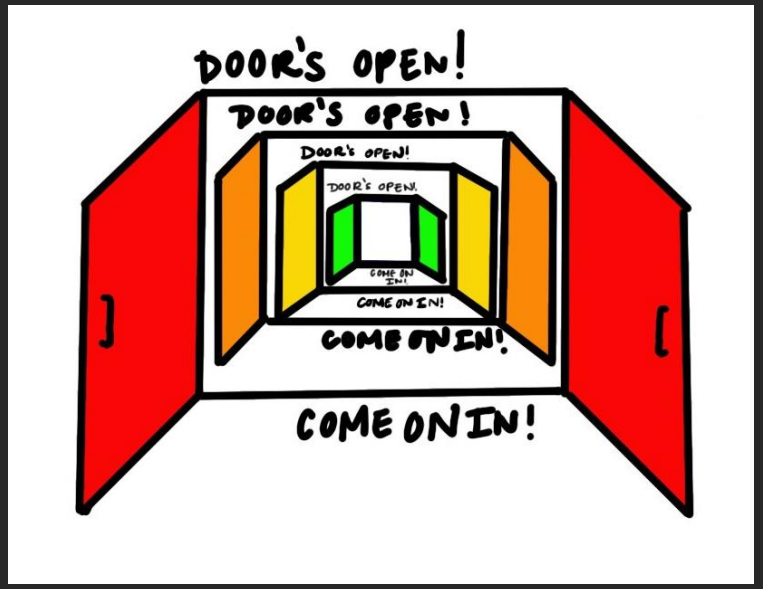
Divide and Conquer: Tree Depth



$$T(n) = aT(g[n]) + f(n)$$

Lecture Outline

- Poll/Admin
- Divide and Conquer Algorithms
- ~~Recurrence Relations~~
- Quicksort: Introduction
- Correctness
- Runtime



Quicksort: Recursive Sorting

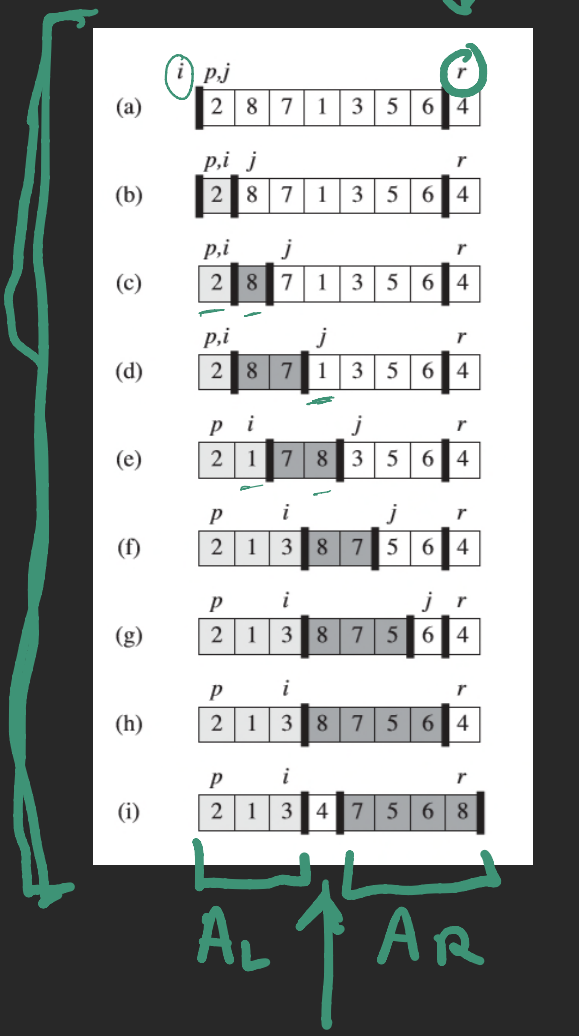
- Quicksort and Mergesort are sorting algorithms that employ recursion and are less time complex than other well known sorting algorithms (Insertion, Bubble)
  - The average case for Quicksort is O(n log n)
  - Randomized version of Quicksort is as fast as Mergesort
  - Worst case runtime Big-theta(n^2)

$\Theta(n^2)$

$\Theta(n \log n)$

The Steps of Quicksort

- **Divide:** pick a pivot r. $A_L \leq r$, $A_R > r$
- **Conquer:** Call quick sort on A_L, A_R or if small don't do anything
- **Combine:** Inplace Algo. don't do anything.



Looking Closer: Pseudocode

A = [- ...]
p = 1
r = A.len

```
// Precondition:  A is the array to be sorted, p>=1;
//                r is <= the size of A
// Postcondition: A[p..r] is in sorted order

Quicksort(A,p,r) {
   if (p<r){
      q = Partition(A,p,r)
      Quicksort(A,p,q-1)
      Quicksort(A,q+1,r)
}}
```

left
right

```
//Precondition:  A[p..r] is the array to be partitioned, p>=1 and r<= size of A,
//               A[r] is the pivot element
//Postcondition: Let A' be the array A after the function is run. Then A'[p..r]
//               contains the same elements as A[p..r]. Further, all elements in
//               A'[p..res-1] are <= A[r], A'[res] = A[r], and all elements in
//               A'[res+1..r] are > A[r]

Partition(A,p,r) {
   x = A[r]
   i = p-1
   for (j=p; j<=r-1;j++) {
      if A[j]<=x {
         i++
         exchange(A[i],A[j])
   }}
   exchange(A[i+1],A[r])
   return i+1
}
```

pivot

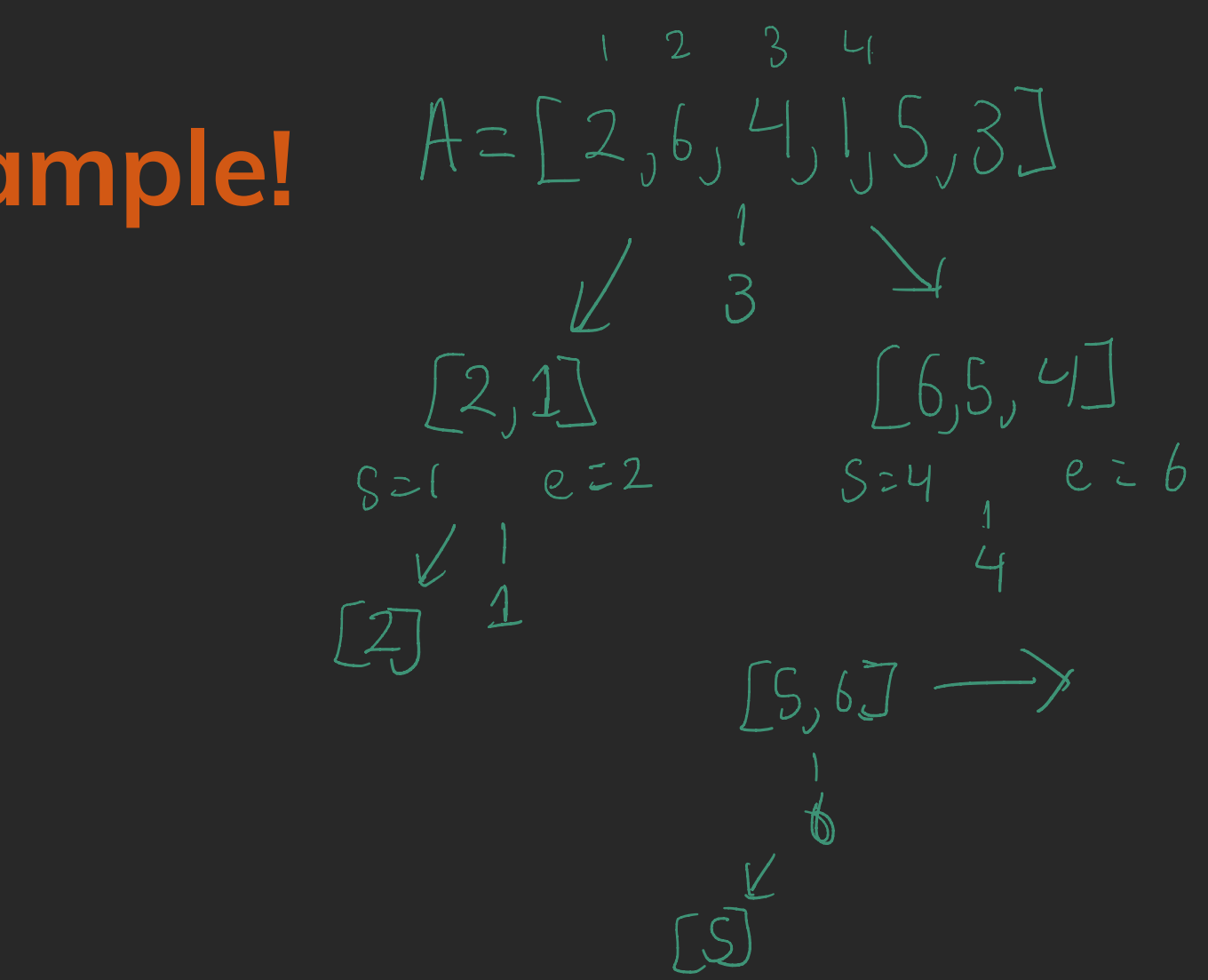
f(n) = O(n)

move to left

pivot in middle

$T(n) = aT(g[n]) + f(n)$

Example!



p = first element

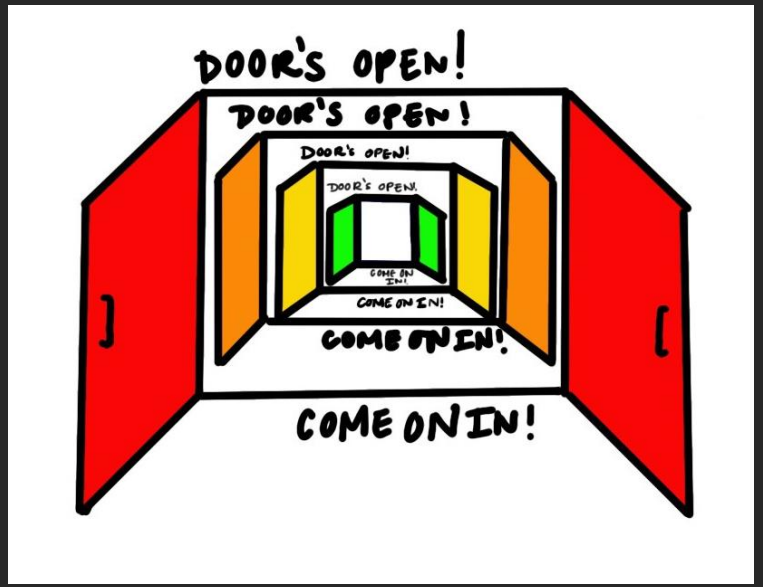
r = A.len

```
Quicksort(A,p,r) {
   if (p<r){
      q = Partition(A,p,r)
      Quicksort(A,p,q-1)
      Quicksort(A,q+1,r)
}}
```

```
Partition(A,p,r) {
   x = A[r]
   i = p-1
   for (j=p; j<=r-1;j++) {
      if A[j]<=x {
         i++
         exchange(A[i],A[j])
   }}
   exchange(A[i+1],A[r])
   return i+1
}
```

Lecture Outline

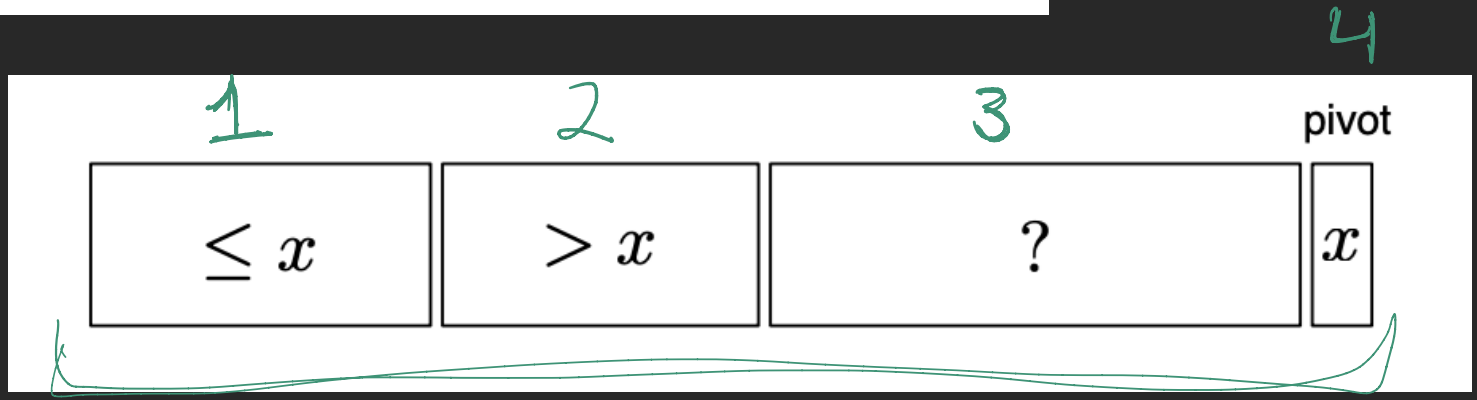
- Poll/Admin
- Divide and Conquer Algorithms
- Recurrence Relations
- ~~Quicksort: Introduction~~
- Correctness
- Runtime



Loop invariants: Correctness

- If an algorithm is *correct*, it yields the proper output on all possible inputs
  - Proving the correctness of Quicksort

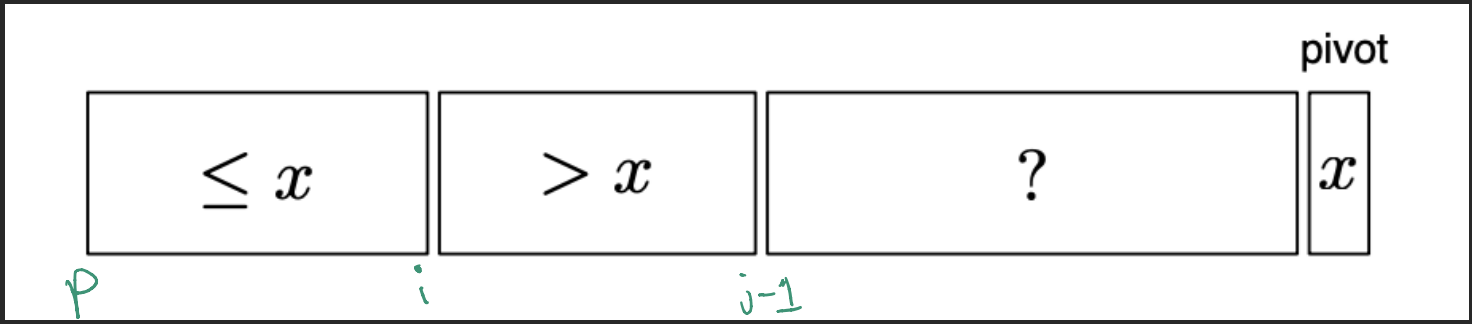
- Region 1: values that are $\leq x$ (between locations p and i)
- Region 2: values that are $> x$ (between locations $i+1$ and $j-1$)
- Region 3: unprocessed values (between locations j and $r-1$)
- Region 4: the value x (location r)



Loop invariants: Correctness

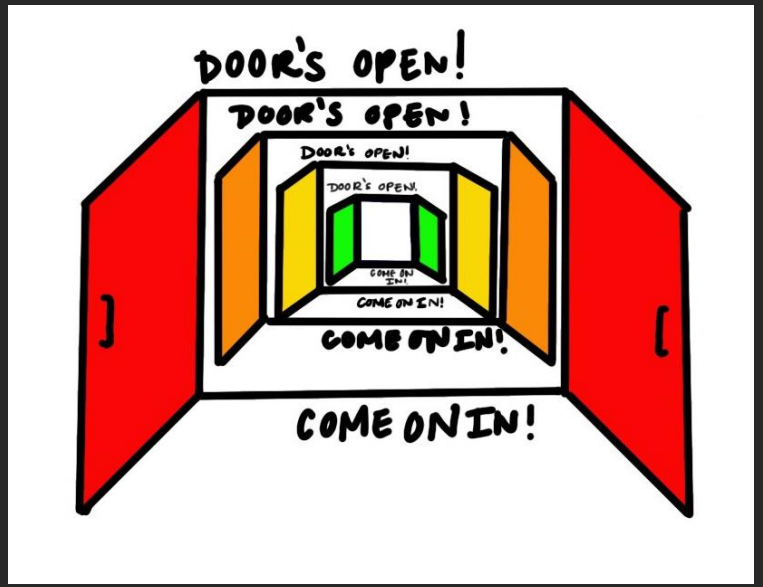
1. If $p \leq k \leq i$ then $A[k] \leq x$
2. If $i+1 \leq k \leq j-1$ then $A[k] > x$
3. If $k = r$ then $A[k] = x$

Chapter 7.1 page: 173



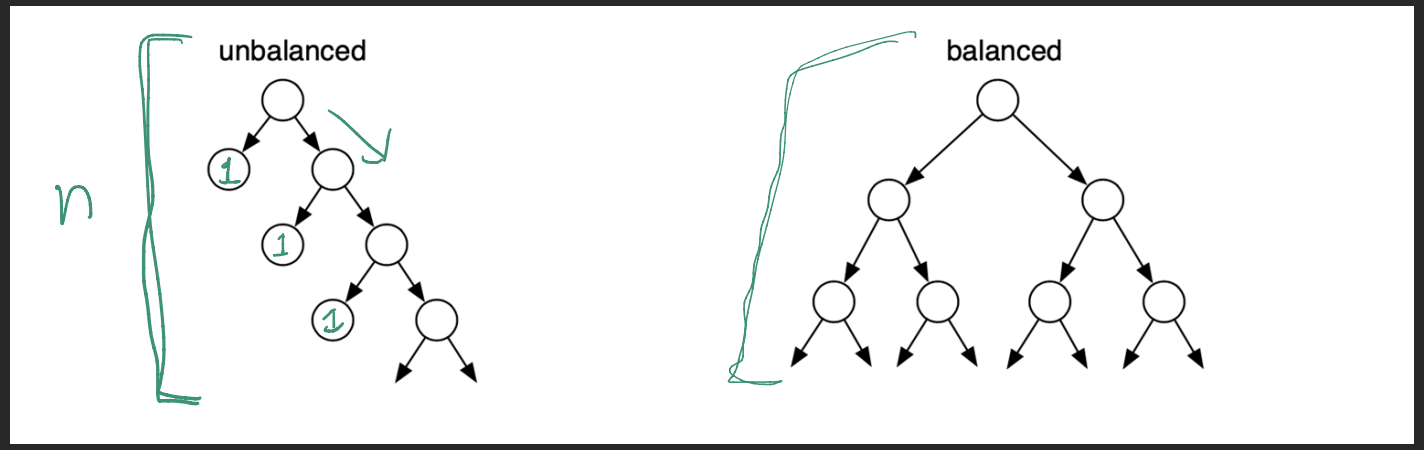
Lecture Outline

- Poll/Admin
- Divide and Conquer Algorithms
- Recurrence Relations
- Quicksort: Introduction
- ~~Correctness~~
- Runtime



Running Time: Computation Tree

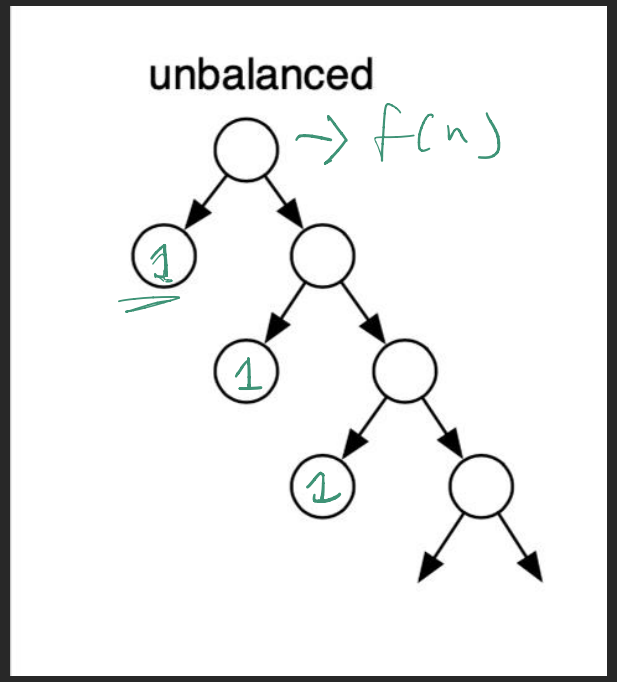
- Analyzing time complexity using computation trees



n^2

```
Partition(A,p,r) {
   x = A[r]
   i = p-1
   for (j=p; j<=r-1;j++) {
      if A[j]<=x {
         i++
         exchange(A[i],A[j])
   }}
   exchange(A[i+1],A[r])
   return i+1
}
```

Analysis: Worst Case



$$T(n) = a\,T(g[n]) + f(n)$$

$g[n] = n - b$, b is a constant

$A = [1, \cdots]$

$b = O(1)$

$T(n) = a\,T(g[n]) + \Theta(n)$

$T(n) = T(n-1) + \Theta(1) + \Theta(n)$

$T(n) = T(n-2) + \Theta(n-1) + \Theta(n)$

$T(n) = \Theta(1) + \cdots \Theta(n-2) + \Theta(n-1) + \Theta(n)$

$T(n) = \sum_{i=1}^{n} \Theta(i) = \frac{n(n+1)}{2}$

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

$n^{\log_2(4)} = n^2$

Chapter 4.3

Analysis: Best Case/Master Theorem

$n^2 = n$

$n^2 = n^{\log_b a}$ $\quad c = 2$ $\quad b = 2$ $\quad a = 2$

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 non-negative integers by the recurrence,

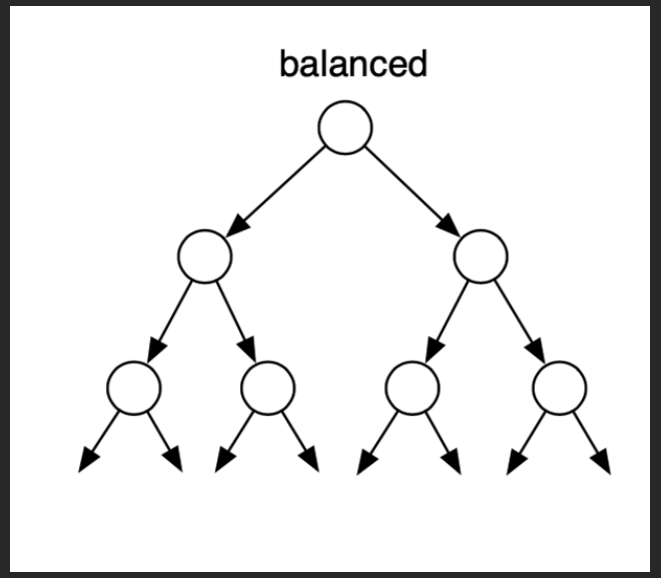
$$T(n) = a\,T(n/b) + f(n) \ ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

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

Chapter 4.3

Analysis: Best Case/Master Theorem



$$T(n) = aT(g[n]) + f(n)$$

$g[n] = \frac{n}{b}$, $b = 2$, $a = 2$

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

$n^{\log_2 2} = n$

$f(n) = \Theta(n)$

$T(n) = n \log(n)$

$f(n) = n$