Week 12 - Friday

COMP 2100

Last time

- What did we talk about last time?
- Sorting
- Insertion sort
- Merge sort

Questions?

Project 4

Assignment 6

Exam 2 Post Mortem

Quicksort

Quicksort

- Pros:
  - **Best** and **average** case running time of $O(n \log n)$
  - Very simple implementation
  - In-place
  - Ideal for arrays
- Cons:
  - Worst case running time of $O(n^2)$
  - Not stable

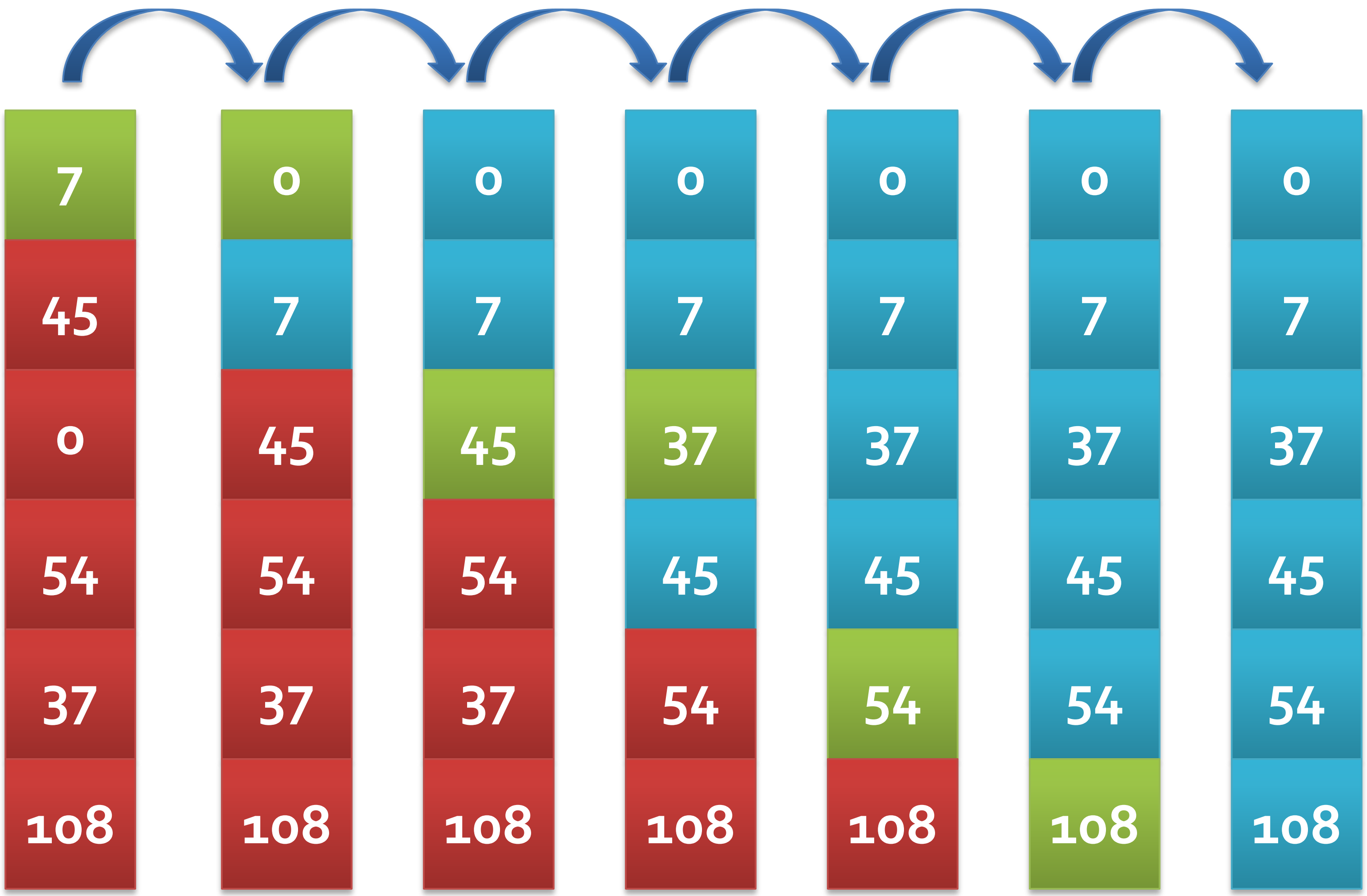
Quicksort algorithm

1. Pick a pivot
2. Partition the array into a left half smaller than the pivot and a right half bigger than the pivot
3. Recursively, quicksort the left half
4. Recursively quicksort the right half

Partition algorithm

- Input: ***array***, ***index***, ***left***, ***right***
- Set ***pivot*** to be ***array***[***index***]
- Swap ***array***[***index***] with ***array***[***right***]
- Set ***index*** to ***left***
- For ***i*** from ***left*** up to ***right*** – 1
  - If ***array***[***i***] ≤ ***pivot***
    - Swap ***array***[***i***] with ***array***[***index***]
    - ***index***++
- Swap ***array***[***index***] with ***array***[***right***]
- Return ***index*** *//so that we know where pivot is*

Quicksort Example



Quicksort issues

- Everything comes down to picking the right pivot
  - If you could get the median every time, it would be great
- A common choice is the first element in the range as the pivot
  - Gives O($\boldsymbol{n}^2$) performance if the list is sorted (or reverse sorted)
  - Why?
- Another implementation is to pick a random location
- Another well-studied approach is to pick three random locations and take the median of those three
- An algorithm exists that can find the median in linear time, but its constant is **HUGE**

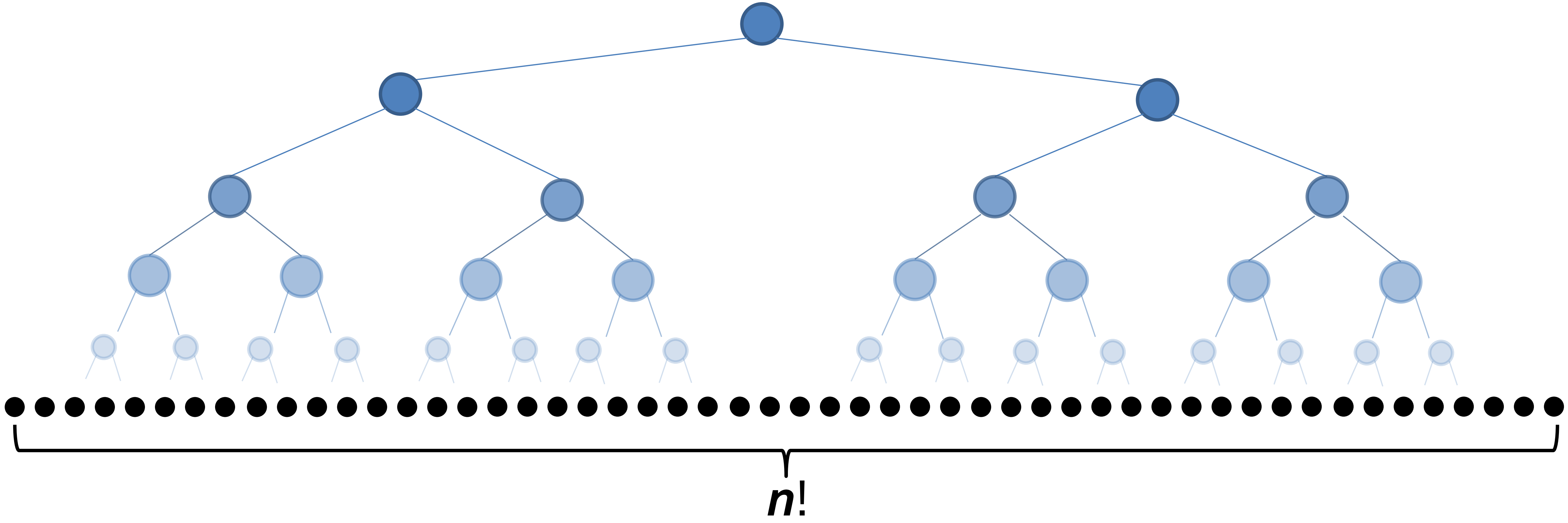
Lower Bound on Sorting

The fastest sort

- How many ways are there to order n items?
- n different things can go in the first position, leaving $n - 1$ to go in the second position, leaving $n - 2$ things to go into the third position…
- $n\ (n - 1)\ (n - 2)\ \ldots\ (2)(1) = n!$
- In other words, there are $n!$ different orderings, and we have to do some work to find the ordering that puts everything in sorted order

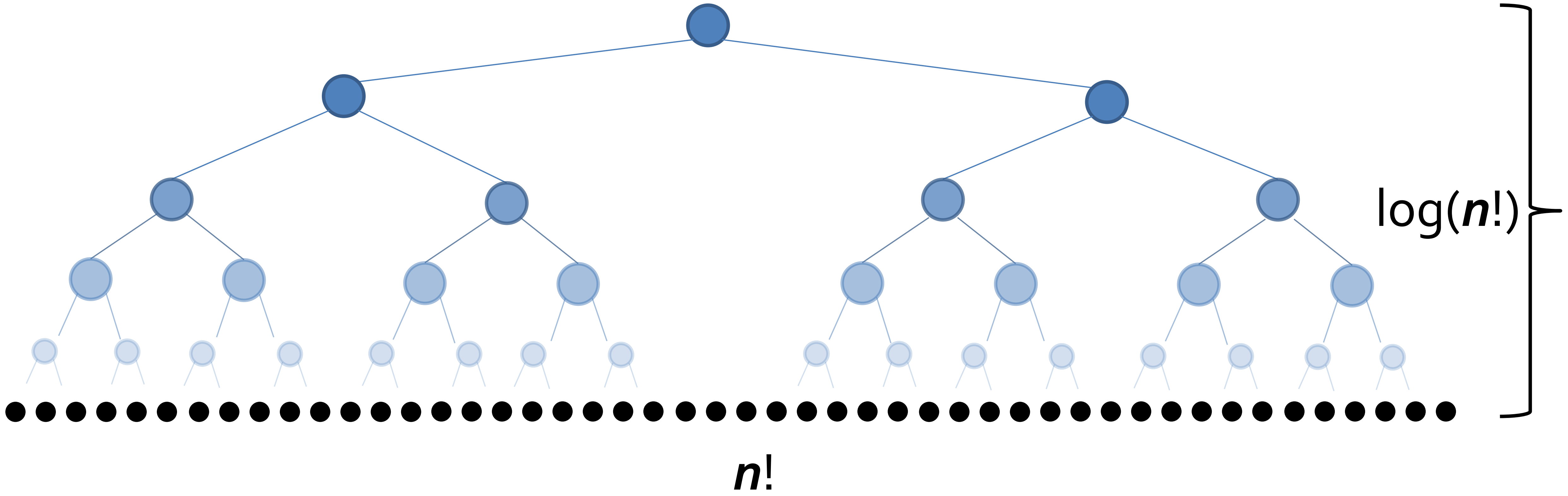
A different kind of tree

- Imagine a tree of **decisions**
- Some sequence of decisions will lead to a leaf of the tree
- Each leaf of the tree **represents** one of those ***n***! orders



Tree height

- What is the smallest height the tree could have?
- A perfectly balanced binary tree with k leaves will have a height of $\log_2(k)$
- Since we have $n!$ leaves, the smallest height will be $\log_2(n!)$



Comparison-based sorts

- **Any** comparison-based sort is going to compare two values and make a decision based on that
- No matter what your algorithm is, if each comparison is a decision in the tree that leads you down to a sorted order, the best you can possibly do is $\log_2(\boldsymbol{n}!)$
- But what is $\log_2(\boldsymbol{n}!)$?
- I wish I could show you the math that backs this up, but Stirling's approximation says that $\log_2(\boldsymbol{n}!)$ is $\Theta(n \log n)$
- **Take away:** No comparison-based sort can **ever** be better than $\Theta(n \log n)$ for worst-case running time

Quiz

Upcoming

Next time…

- Counting sort
- Radix sort
- Heaps
- Heapsort
- TimSort

Reminders

- Work on Project 4
- Finish Assignment 6
  - **Due tonight by midnight!**
- Read Section 2.4