On your way in...

Pick-Up:
1. Lecture 24 notes
Welcome to CS 134!

Introduction to Computer Science
Iris Howley

- Searching Sorting -
If you like this topic (i.e., algorithms)...

• Enroll in CS136 Data Structures!
  ▪ ...take CS256 Algorithm Design & Analysis
  ▪ ...take CS361 Theory of Computation

• See “Grokking Algorithms” by Bhargava

• “Algorithms to Live By: The Computer Science of Human Decisions” is a lighter read by Christian and Griffiths
Sorting Algorithm Run-times

- Insertion Sort
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$
- Bubble Sort
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$
- Selection Sort
  - Best Case: $O(n^2)$
  - Worst Case: $O(n^2)$
- Quick Sort
  - Best Case: $O(n \log n)$
  - Worst Case: $O(n^2)$
Sorting Algorithm Run-times

- **Insertion Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$

- **Bubble Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$

- **Selection Sort**
  - Best Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- **Quick Sort**
  - Best Case: $O(n \log n)$
  - Worst Case: $O(n^2)$

Why is Quick Sort used over Insertion/Bubble Sort?
Sorting Algorithm Run-times

• Insertion Sort
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

• Bubble Sort
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

• Quick Sort
  - Best Case: $O(n \log n)$
  - Avg Case: $O(n \log n)$
  - Worst Case: $O(n^2)$

Why is Quick Sort used over Insertion/Bubble Sort?
### Sorting Algorithm Run-times

- **Insertion Sort**
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- **Bubble Sort**
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- **Quick Sort**
  - Best Case: $O(n \log n)$
  - Avg Case: $O(n \log n)$
  - Worst Case: $O(n^2)$

**Why is Insertion Sort used over Bubble Sort?**
Insertion Sort vs. Bubble Sort

• It has to do with those constants we drop to get an upper bound on run-time
• Insertion Sort does not have to do quite as many operations as Bubble Sort

Insertion Sort

6 5 3 1 8 7 2 4

Bubble Sort

6 5 3 1 8 7 2 4

• Bubble Sort $\rightarrow \frac{n^2}{2}$
• Insertion Sort $\rightarrow \frac{n^2}{4}$ (slightly faster)
Sorting Algorithm Run-times

- Insertion Sort
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- Bubble Sort
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- Quick Sort
  - Best Case: $O(n \log n)$
  - Avg Case: $O(n \log n)$
  - Worst Case: $O(n^2)$

Why might Insertion Sort be used over Quick Sort?
Insertion Sort vs. Quick Sort

• Insertion Sort
  ▪ Used when n is small → low overhead
    o Recursive calls cost memory!!
  ▪ Used when data is nearly sorted
    o $O(n) < O(n \log n)$
  ▪ It’s stable
    o Doesn’t change the relative order of elements with equal keys

• Quick Sort
  ▪ Worst case is also $O(n^2)$
  ▪ Average case is much better $O(n \log n) < O(n^2)$

Many sorts use Insertion Sort up to a certain list size, then switch to Quick Sort!
MERGE SORT

More sorting!
How many levels to sort?

2 3 4 5

4 elements = 2 levels
How many levels for a list of length 8?
Merge Sort Levels

• 2 elements = 1 level
• 4 elements = 2 levels
• 8 elements = 3 levels
• 16 elements = 4 levels

• What is this growth rate similar to?
  ▪ Paper folding from Wednesday!
  ▪ Number of levels grows by $O(\log n)$
How many operations at each level?
How many operations/level for a list of length 8?

If you’re thinking it should be $n/2$ operations per level, remember, we drop constants!
Merge Sort Operations

- n operations at each level

**What is this growth rate?**
- Number of operations at each level grows by $O(n)$
- Add another level? Add another n operations

**What is the run-time of Merge Sort?**
- Number of levels * Number of operations/level
- $O(\log n \times n)$
- $O(n \log n)$

What does the list look like for a Best Case scenario?

...trick question!
All lists end up with the same run-time growth. $O(n \log n)$ is pretty good!
How many levels for a SORTED list of length 8?
Sorting Algorithm Run-times

- **Insertion Sort**
  - Best Case: $O(n)$
  - Avg Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- **Quick Sort**
  - Best Case: $O(n \log n)$
  - Avg Case: $O(n \log n)$
  - Worst Case: $O(n^2)$

- **Merge Sort**
  - Best Case: $O(n \log n)$
  - Avg Case: $O(n \log n)$
  - Worst Case: $O(n \log n)$

Why use Quick Sort over Merge Sort?
Quick Sort vs. Merge Sort

• Quick Sort
  ▪ No extra memory! In-place, lots of swapping!
  ▪ If you choose the right pivot, you can typically avoid worst case performance

• Merge Sort
  ▪ Always $O(n \log n)$
  ▪ But requires additional memory to do the sorting
Merge Sort

• `def _merge(a, b):
  • if not a:
    o return b
  • if not b:
    o return a
  • if a[0] <= b[0]:
    o return [a[0]] + _merge(a[1:], b)
  • else:
    o return [b[0]] + _merge(a, b[1:])`

Simple case of merging a list with an empty list

Put the lower first element of each list at the front, and then merge the rest
Merge Sort

• def mergeSort(d):
  ▪ n = len(d)
  ▪ if n >= 2:
    ○ mid = n//2  Otherwise, find midpoint of list
    ○ left = mergeSort(d[:mid])  Sort/Split in half, recursively
    ○ right = mergeSort(d[mid:])
    ○ return _merge(left,right)  Merge in sorted order
  ▪ else:
    ○ return d  Base case of only sorting <3 elements

Code for sorts is stored in shared/examples/
SEARCHING

Finding things is very important
HOW DO YOU FIND A WORD IN A DICTIONARY?
Finding Words in a Paper Dictionary

1. Recall: a paper dictionary is a *sorted* list of words + definitions
2. Split book in half
3. Is the term before/after this page?
4. Split that in half
5. Is the term before/after this page?
6. Split that in half
7. Is the term before/after this page?
8. Split that in half
9. Is the term before/after this page?
10. ...continue until you find the location where the value should be
Binary Search

• Best Case: ???
  - $O(1)$

• Average Case: ???
  - $O(\log n + 1/2) \rightarrow$ average Worst + Best Cases
  - Drop the constants
  - $O(\log n)$

• Worst Case: ???
  - $O(\log n)$

$O(\log_2 n) \rightarrow$ The 2 stems from splitting lists in half!
Binary Search

- **def bs(d, v):** Set-up our starting and ending (start w. whole list)
  - `low = 0`
  - `high = len(d)`
  - **while** `low < high:`
    - `mid = (high + low)//2` Find the mid-point
    - `# note: mid is never high`
    - `if v <= d[mid]:` If our value is less than what’s at the midpoint, then midpoint is our new ‘high’
      - `high = mid`
    - `else:` Otherwise, we want to look at the upper half, so midpoint is our new ‘low'
      - `low = mid+1`
  - **return** `low`

When `low==high`, then we’ve made the list so small, we’ve found where the item *should* be located!
Binary Search

https://www.youtube.com/watch?v=iP897Z5Nerk
First image of a black hole captured.
Kate Bouman, PhD
One of the creators of the algorithm to combine the data of 8 radio telescopes around the world.

Photographing a Black Hole

• Data processing, algorithm discussed in paper “First M87 Event Horizon Telescope Results. III. Data Processing and Calibration”

• Cites some interesting python modules:
  ▪ Numpy (van der Walt et al. 2011)
  ▪ Scipy (Jones et al. 2001)
  ▪ Pandas (McKinney 2010)
  ▪ Jupyter (Kluyver et al. 2016)
  ▪ Matplotlib (Hunter 2007)

• Reddit is “pretty sure” it’s python & matplotlib in the previous Facebook photo

https://iopscience.iop.org/article/10.3847/2041-8213/ab0c57
QUESTIONS?
Leftover Slides
QUICK SORT

More [recursive] sorting!
Quick sort

• Sort these lists:

\[
\begin{array}{c}
7 \\
\end{array}
\]

• if \( n < 2 \):
  • Stop!

When \( n = 0 \), or \( n = 1 \)
Quicksort \( n = 2 \) *

- Check if first element is smaller than the second
- If it isn’t, swap them

*After next slide: note that this step is actually picking the leftmost item to be the pivot, and then partitioning into a [3] “less-than” sublist and an empty “greater-than” sublist
Quicksort \( n = 3 \)

- Pick an element, we call it the *pivot*
  - Let’s start with the leftmost item as the pivot
Quicksort \( n = 3 \)

- Find all the elements smaller than the pivot, and all the elements larger than the pivot
  - We call this *partitioning*
- We now have a sub-list of all elements smaller, and a sub-list of all elements larger
- The two sub-lists are not sorted! Just partitioned based on our pivot
Quicksort \( n = 3 \)

- If they were sorted, then we could just combine the lowerList + pivot + greaterList (but we haven’t sorted them yet)

- We call quicksort on our lower-list [3] and it’s only one element! We know how to sort that!

- We call quicksort on our greater list [7], and ...ditto

- Now we can combine lowerList + pivot + greaterList
Quicksort $n > 2$

• Pivot is 5
• Partition!
• We have a lower list of 2 and a greater list of 1, we know how to sort those!
  ▪ Call quicksort again, on each
• Sorted!
Quicksort

No lower list!

After each quicksort call:

We know our pivot is sorted, but nothing else!

Combine all sub-lists at the end!
How many levels of quicksort calls?
How many levels for a list of length 8?
Pre-sorted Quicksort

- 2 elements = 2 levels of calls to quicksort
- 4 elements = 4
- 8 elements = 8
- 16 elements = 16

- What is this growth rate?
  - Number of levels grows by $O(n)$

Levels of recursive calls to quicksort works a little like our “outer loop” for iterative sorting methods.
For each call to quicksort, how many operations?

During partitioning, has to look at each element to see if it’s less than or greater than the pivot!

As always, this is technically: $n$

But we drop the constants!
Quick Sort Operations

- n operations for each call

- **What is this growth rate?**
  - Number of operations at each level grows by \(O(n)\)
  - Add another level? Add another n operations

- What is the run-time of Quick Sort when the list is sorted?
  - \(O(\text{Number of calls } \times \text{Number of operations/call})\)
  - \(O(n \times n)\)
  - \(O(n^2)\)

\(O(n^2)\) isn’t great. (Especially for a recursive algorithm)
Let’s pick a random pivot!

Quicksort

How many levels quicksort calls?
How many call levels for a list of length 8?
Pre-sorted Quicksort

- 2 elements = 1 levels of calls to quicksort
- 4 elements = 2
- 8 elements = 3
- 16 elements = 4

- **What is this growth rate similar to?**
  - Paper folding from Wednesday!
  - Number of levels grows by $O(\log n)$
Quick Sort Operations

• Number of operations?
  ▪ n operations for each call (still)

• What is this growth rate?
  ▪ Number of operations at each level grows by $O(n)$
  ▪ Add another level? Add another n operations

• What is the run-time of Quick Sort when the list is sorted?
  ▪ $O(\text{Number of calls} \times \text{Number of operations/call})$
  ▪ $O(n \times \log n)$
  ▪ $O(n \log n)$  **Best case scenario!**

**Worst Case?** We already saw it, sorted list with a bad pivot selected repeatedly: $O(n^2)$
Sorting Algorithm Run-times

- **Insertion Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$

- **Bubble Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$

- **Selection Sort**
  - Best Case: $O(n^2)$
  - Worst Case: $O(n^2)$

- **Quick Sort**
  - Best Case: $O(n \log n)$
  - Worst Case: $O(n^2)$
Quick Sort

https://www.youtube.com/watch?v=ywWBy6J5gz8
If you like this topic (i.e., algorithms)...

• Enroll in CS136 Data Structures!
  ▪ ...take CS256 Algorithm Design & Analysis
  ▪ ...take CS361 Theory of Computation

• See “Grokking Algorithms” by Bhargava

• “Algorithms to Live By: The Computer Science of Human Decisions” is a lighter read by Christian and Griffiths