Sorting

Data Structures and Algorithms
Warmup

Discuss with your neighbors:

What considerations do we think about when choosing a sorting algorithm?

So far we have seen: selection sort, insertion sort, and heap sort. What is the “main idea” behind each one? What are their properties? In which contexts are they better or worse?
### Warmup

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Main Idea</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
<th>In Place?</th>
<th>Stable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Sort</td>
<td>Repeatedly find the next smallest element and put in front.</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>Pull the next unsorted element and insert into the proper position.</td>
<td>O(n)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Heap Sort</td>
<td>Repeatedly pull the min element from a heap.</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>Can Be</td>
<td>???</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>Recursively sort then merge the left and right halves.</td>
<td>O(n log n)*</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>No</td>
<td>???</td>
</tr>
</tbody>
</table>

* there are O(n) best case variants of merge-sort used in practice
Announcements

Individual Homework Due Tonight

Project 2 is assigned – it’s a one week project (so due on Friday)

Also by Friday: sign up for partner for project 3! https://goo.gl/forms/KYVCv4QddVN5Rbyi1
- Remember to sign up for a partner – you won’t automatically be re-partnered with the same person
  - (for random partnering, we’ll assume your availability is the same as last time)

Course format change: Smaller homeworks, more frequently
- Should keep HW content closer to lecture content
Review: Selection Sort and Insertion Sort

https://visualgo.net/en/sorting
Merge Sort

https://www.youtube.com/watch?v=XaqR3G_NVoo

Divide

Conquer

Combine
Merge Sort

mergeSort(input) {
  if (input.length == 1)
    return
  else
    smallerHalf = mergeSort(new [0, ..., mid])
    largerHalf = mergeSort(new [mid + 1, ...])
    return merge(smallerHalf, largerHalf)
}

Worst case runtime?

Best case runtime?  \( T(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
2T(n/2) + n & \text{otherwise} 
\end{cases} \)

Average runtime?

Stable?  Yes

In-place?  No
Merge Sort Optimization

Use just two arrays – swap between them

Another Optimization: Switch to Insertion Sort for small arrays (e.g. n < 10)
Merge Sort Benefits

Useful for massive data sets that cannot fit on one machine
Works well for linked-lists and other sequentially accessible data sets
A $O(n \log n)$ stable sort!
Easy to implement!

```
mergeSort(input) {
    if (input.length == 1)
        return
    else
        smallerHalf = mergeSort(new [0, ..., mid])
        largerHalf = mergeSort(new [mid + 1, ...])
        return merge(smallerHalf, largerHalf)
}
```

Homework!
Quick Sort

Main Idea: Divide and Conquer – “smaller” “half” and “bigger” “half”

\[
\begin{array}{c}
\text{if } x < 5 & \text{then } x < 10 \\
\text{if } x > 5 & \text{then } x > 10 \\
\text{if } x = 5 & \text{then } x = 10 \\
\end{array}
\]

“smaller” and “bigger” relative to some pivot element

“half” doesn’t always mean half, but the closer it is to half, the better
Quick Sort

Divide

Conquer

Combine

https://www.youtube.com/watch?v=ywWB6J5gz8
Quick Sort

```java
quickSort(input) {
    if (input.length == 1)
        return
    else
        pivot = getPivot(input)
        smallerHalf = quickSort(getSmaller(pivot, input))
        largerHalf = quickSort(getBigger(pivot, input))
    return smallerHalf + pivot + largerHalf
}
```

Worst case runtime?

\[ T(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
 n + T(n - 1) & \text{otherwise} 
\end{cases} \]

Best case runtime?

\[ T(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
 n + 2T(n/2) & \text{otherwise} 
\end{cases} \]

Average runtime?

\[ T(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
 n + 2T(n/2) & \text{otherwise} 
\end{cases} \]

Stable? No

In-place? No
Can we do better?

Pick a better pivot
- Pick a random number
- Pick the median of the first, middle and last element

Sort elements by swapping around pivot in place
Better Quick Sort

Low
X < 6

High
X >= 6
Project 2: Invariants, Pre-conditions, and post-conditions

```
Count = 0

while (!stack.is_empty())
    item = stack.pop()
    process Item
    Count++
```
Introduction to Graphs
Inter-data Relationships

Arrays
Categorically associated
Sometimes ordered
Typically independent
Elements only store pure data, no connection info

Trees
Directional Relationships
Ordered for easy access
Limited connections
Elements store data and connection info

Graphs
Multiple relationship connections
Relationships dictate structure
Connection freedom!
Both elements and connections can store data
Graph: Formal Definition

A graph is defined by a pair of sets $G = (V, E)$ where...
- $V$ is a set of vertices
  - A vertex or “node” is a data entity
- $E$ is a set of edges
  - An edge is a connection between two vertices

$V = \{ A, B, C, D, E, F, G, H \}$

$E = \{ (A, B), (A, C), (A, D), (A, H),
(C, B), (B, D), (D, E), (D, F),
(F, G), (G, H) \}$
Applications

Physical Maps
- Airline maps
  - Vertices are airports, edges are flight paths
- Traffic
  - Vertices are addresses, edges are streets

Relationships
- Social media graphs
  - Vertices are accounts, edges are follower relationships
- Code bases
  - Vertices are classes, edges are usage

Influence
- Biology
  - Vertices are cancer cell destinations, edges are migration paths

Related topics
- Web Page Ranking
  - Vertices are web pages, edges are hyperlinks
- Wikipedia
  - Vertices are articles, edges are links

SO MANY MORREEEE
www.allthingsgraphed.com
Graph Vocabulary

Graph Direction
- Undirected graph – edges have no direction and are two-way
  \[ V = \{ A, B, C \} \]
  \[ E = \{ (A, B), (B, C) \} \text{ inferred } (B, A) \text{ and } (C,B) \]
- Directed graphs – edges have direction and are thus one-way
  \[ V = \{ A, B, C \} \]
  \[ E = \{ (A, B), (B, C), (C, B) \} \]

Degree of a Vertex
- Degree – the number of edges containing that vertex
  \[ A : 1, B : 1, C : 1 \]
- In-degree – the number of directed edges that point to a vertex
  \[ A : 0, B : 2, C : 1 \]
- Out-degree – the number of directed edges that start at a vertex
  \[ A : 1, B : 1, C : 1 \]
Food for thought

Is a graph valid if there exists a vertex with a degree of 0? Yes

A has an “in degree” of 0
B has an “out degree” of 0
Are these valid?

Yup

C has both an “in degree” and an “out degree” of 0
Sure

Yes!
Graph Vocabulary

**Self loop** – an edge that starts and ends at the same vertex

![Self loop diagram](image)

**Parallel edges** – two edges with the same start and end vertices

![Parallel edges diagram](image)

**Simple graph** – a graph with no self-loops and no parallel edges