Sorting

Data Structures and Algorithms

Warmup

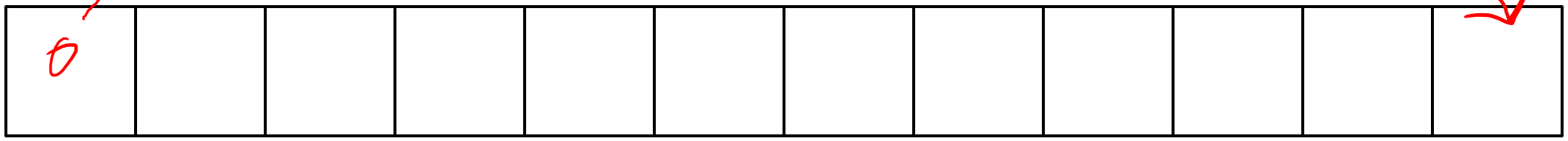
Discuss with your neighbors:

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

Algorithm	Main Idea	Best Case	Worst Case	Average Case	In Place?	Stable?
Selection Sort	Repeatedly find the next smallest element and put in front.	O(n^2) always traverses entire list to be sure given element is a min	O(n^2)	O(n^2)	Yes	Yes
Insertion Sort	Pull the next unsorted element and insert into the proper position.	O(n) already sorted (ascending or descending depends on algorithm	O(n^2)	O(n^2) sorted in reverse order	Yes	Yes
Heap Sort	Repeatedly pull the min element from a heap.	O(n log n) good for consistent sorting times	O(n log n)	O(n log n)	Can Be but reverses order using min heap	~~Yes~~ NO
Merge Sort	Recursively sort then merge the left and right halves.	O(n log n)*	O(n log n)	O(n log n)	No	???



* 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

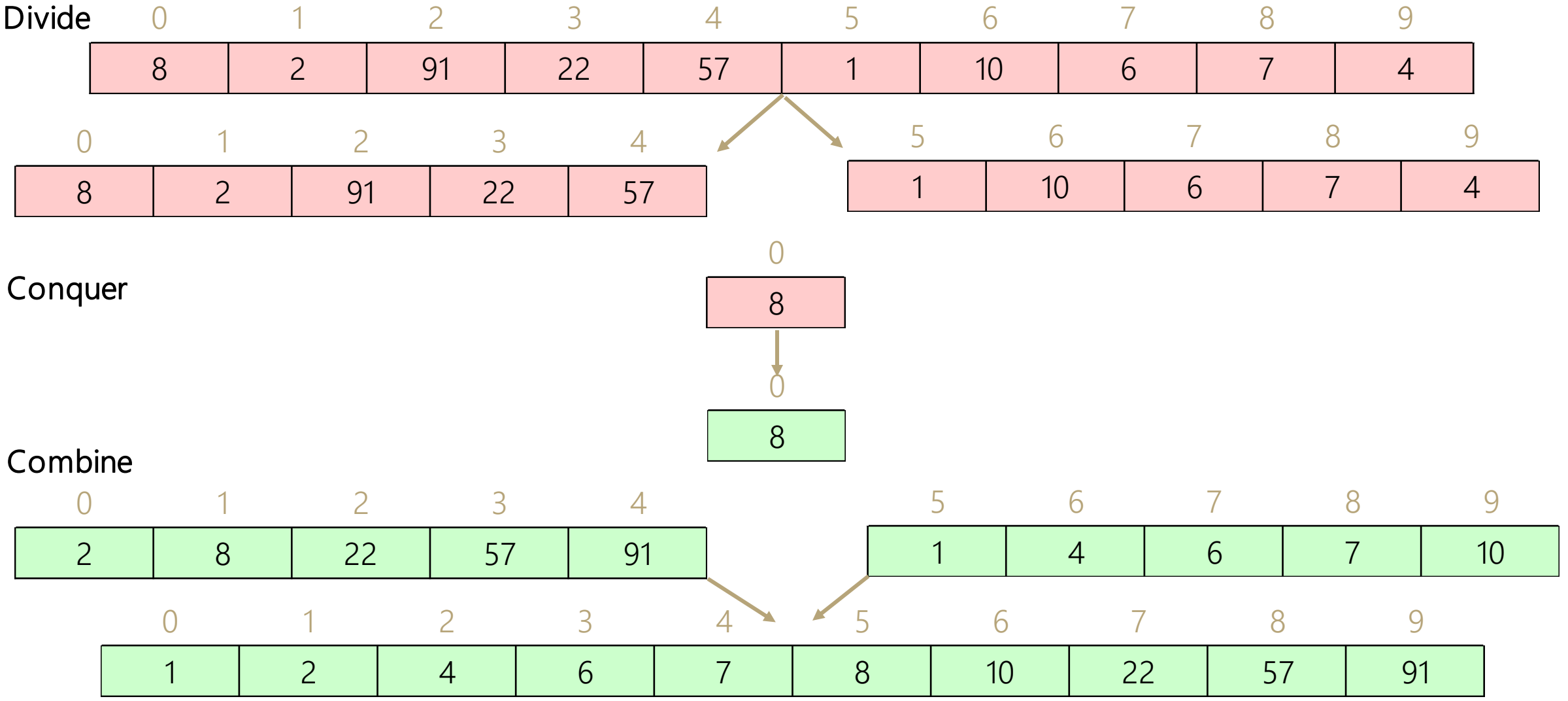
- Should keep HW content closer to lecture content

Review: Selection Sort and Insertion Sort

https://visualgo.net/en/sorting

Merge Sort

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



Merge Sort

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

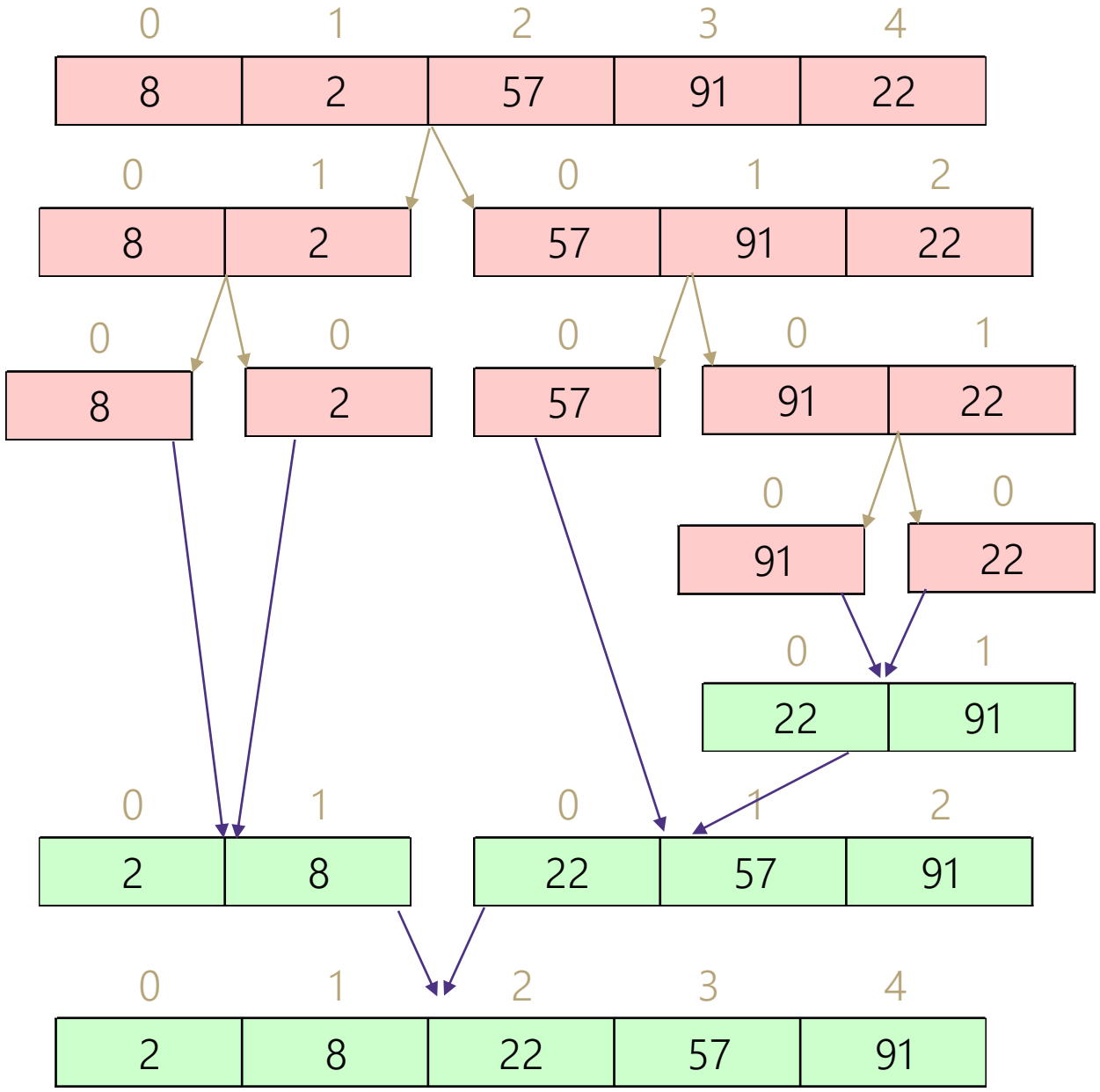
Worst case runtime?

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

Average runtime?

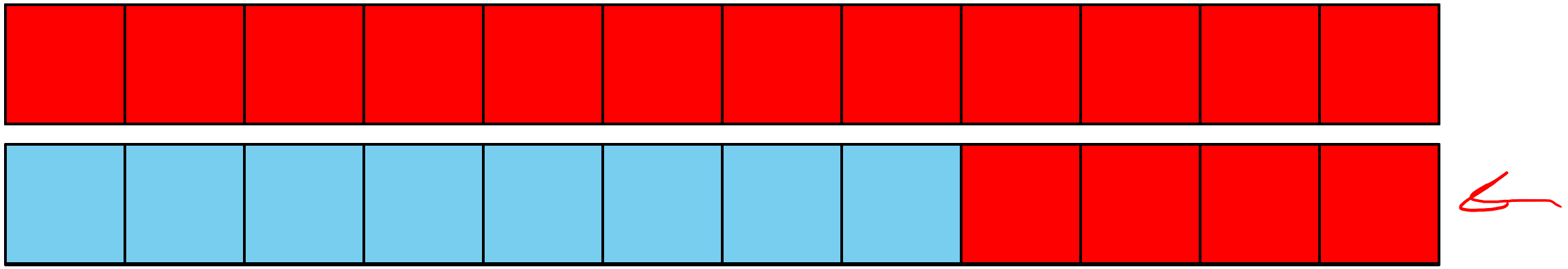
Stable? Yes

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

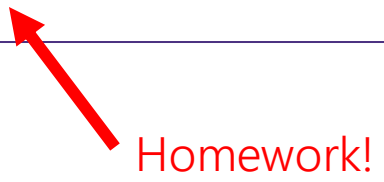
Useful for massive data sets that cannot fit on one machine

Works well for linked-lists and other sequentially accessible data sets

A O(n log n) stable sort!

Easy to implement!

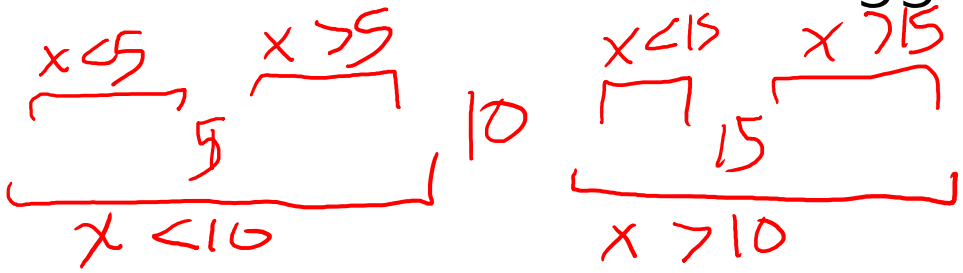
```
mergeSort(input) {
   if (input.length == 1)
      return
   else
      smallerHalf = mergeSort(new [0, ..., mid])
      largerHalf = mergeSort(new [mid + 1, ...])
      return merge(smallerHalf, largerHalf)
}
```



"Quick" Sort

Worst Case $O(n^2)$

Main Idea: Divide and Conquer – "smaller" "half" and "bigger" "half"

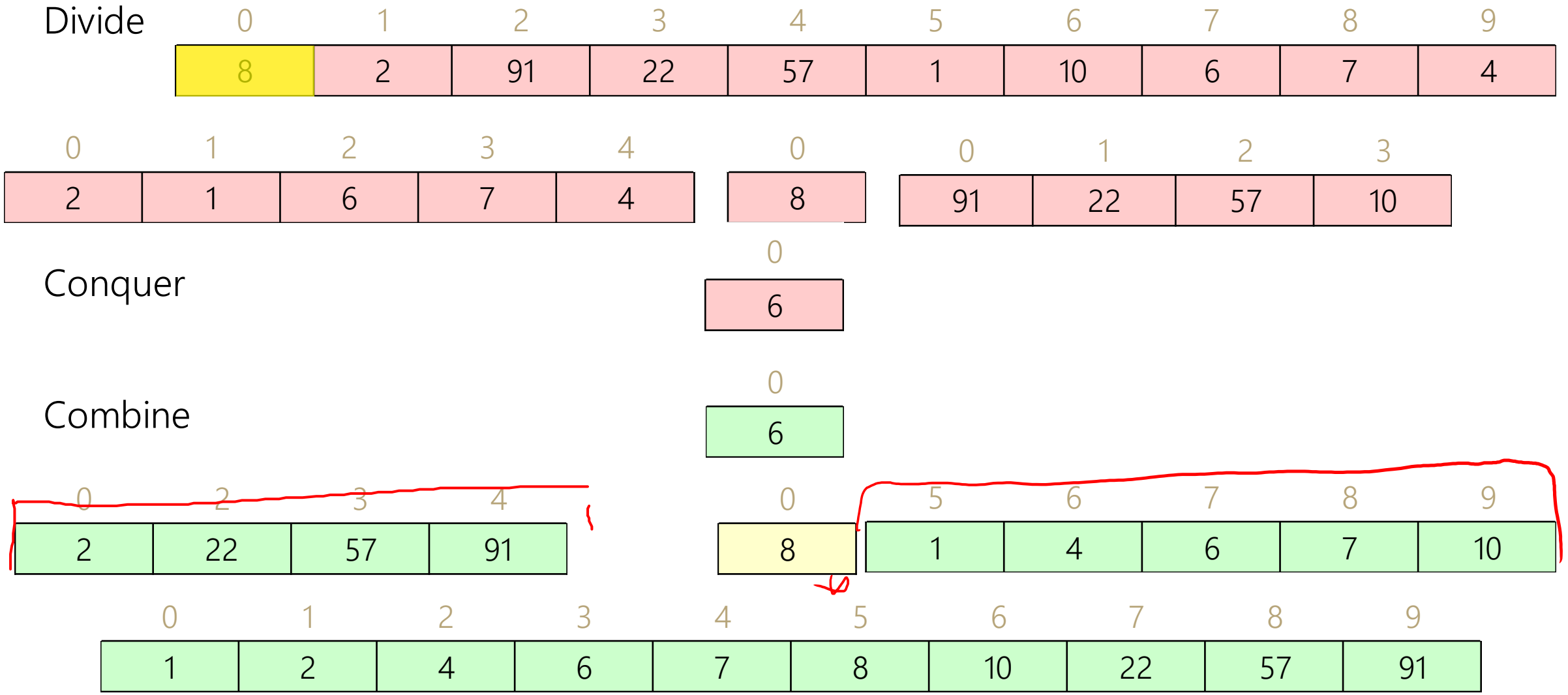


"smaller" and "bigger" relative to some **pivot** element

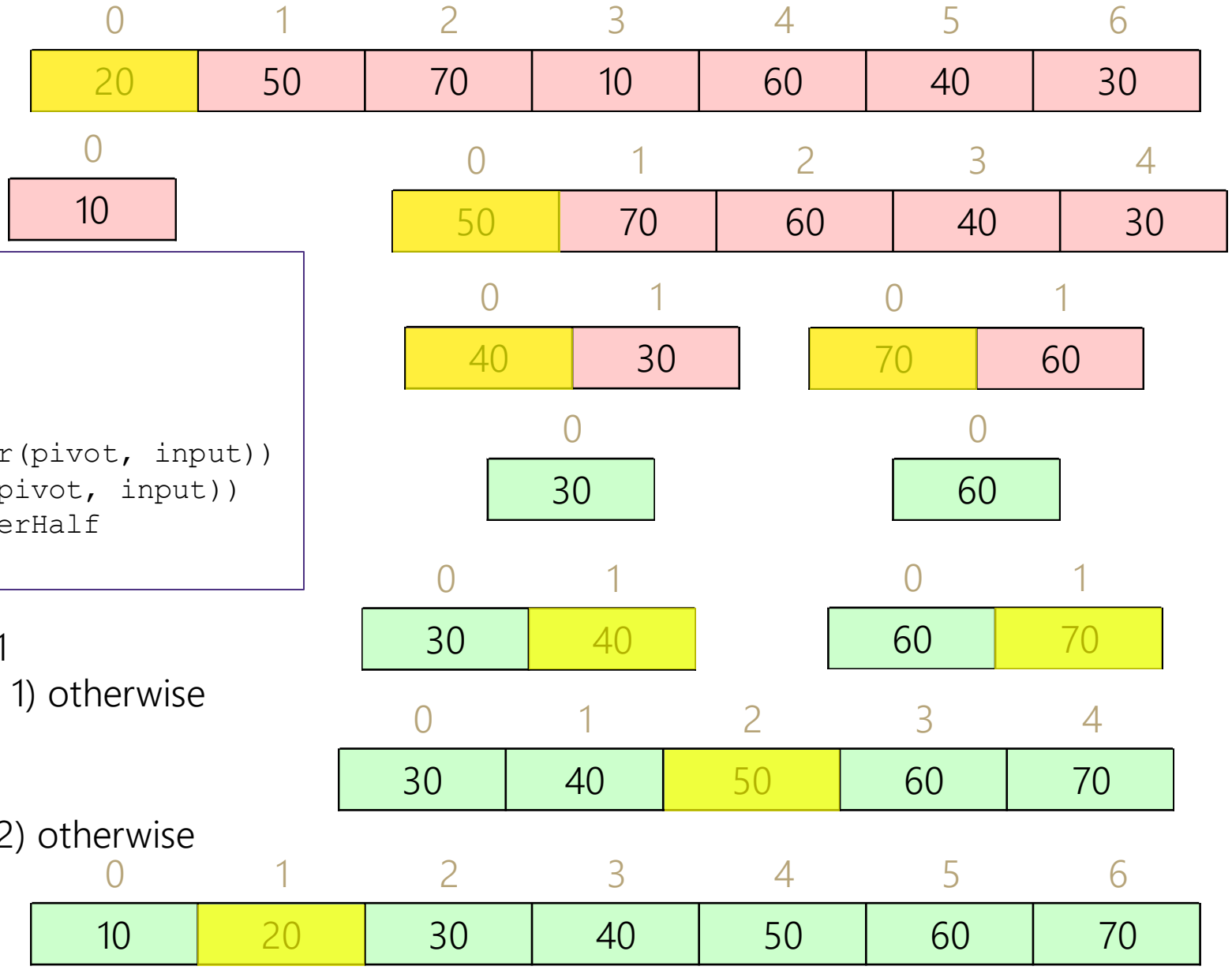
"half" doesn't always mean half, but the closer it is to half, the better

Quick Sort

https://www.youtube.com/watch?v=ywWBy6J5gz8



Quick Sort




```
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? $T(n) = \begin{cases} 1 \text{ if } n <= 1 \\ n + T(n-1) \text{ otherwise} \end{cases}$

Best case runtime?

Average runtime? $T(n) = \begin{cases} 1 \text{ if } n <= 1 \\ n + 2T(n/2) \text{ otherwise} \end{cases}$

Stable? No

In-place? No

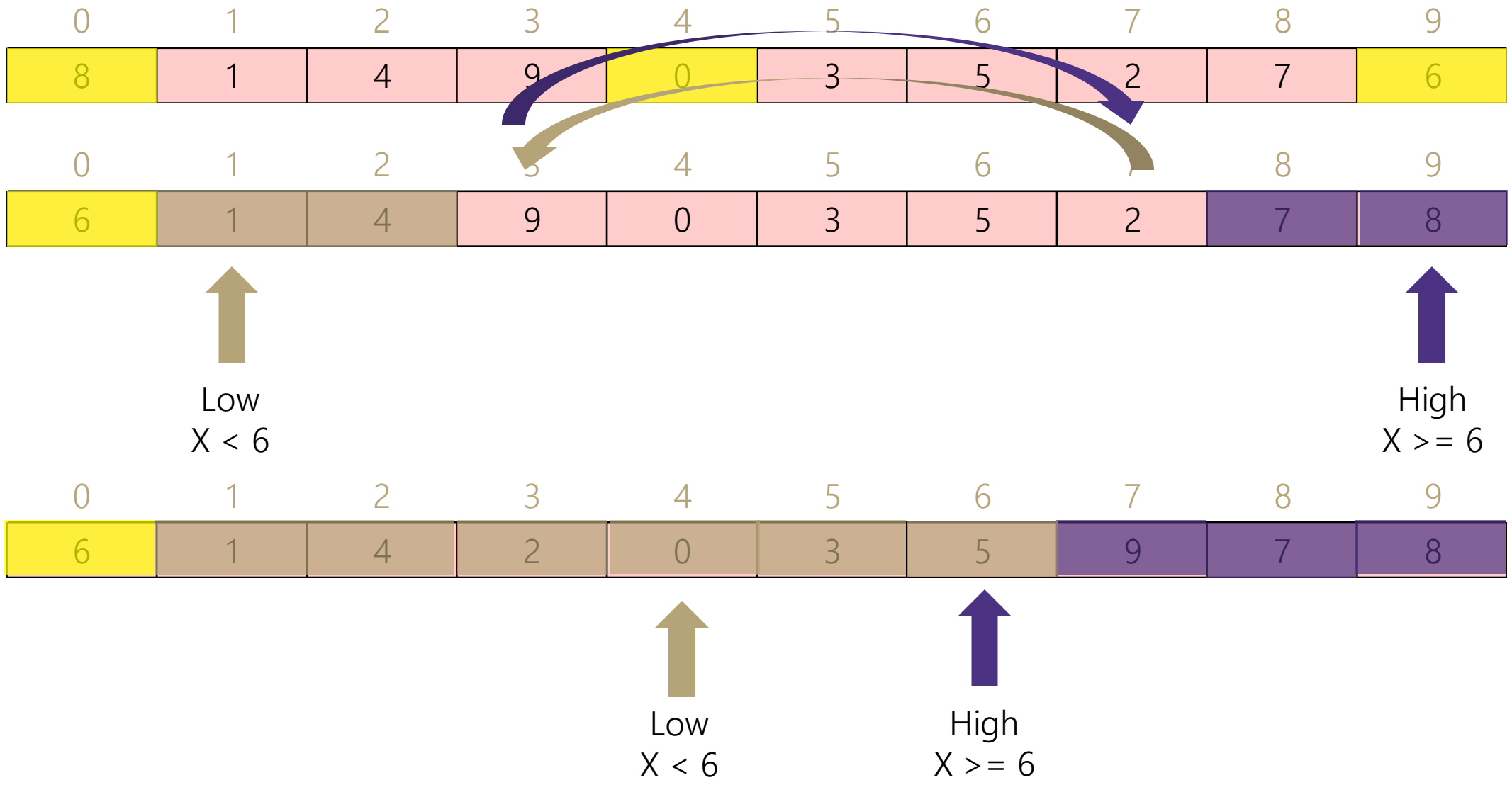
Can we do better?

Pick a better pivot

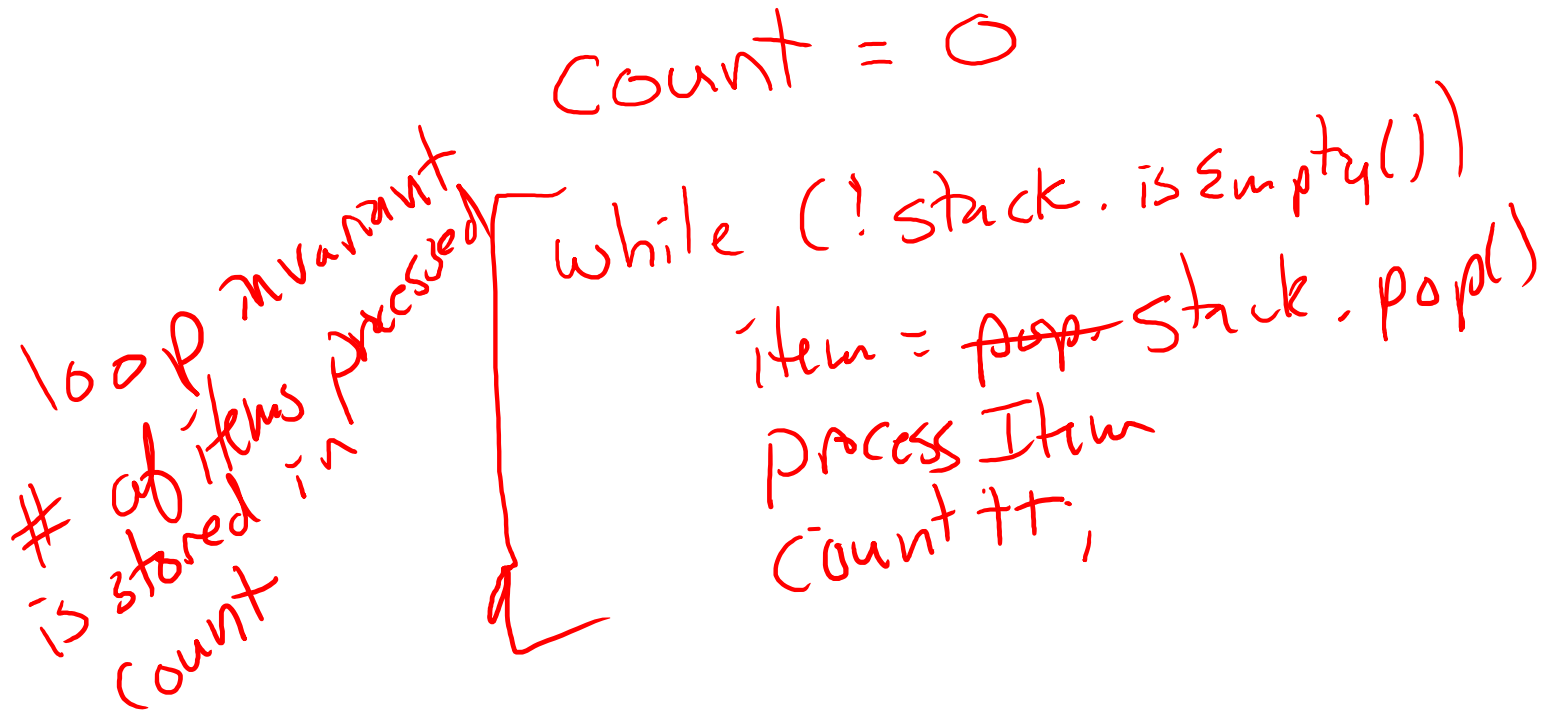
- Pick a random number
- Pick the median of the first, middle and last element

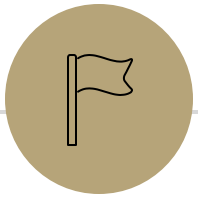
Sort elements by swapping around pivot in place

Better Quick Sort



Project 2: Invariants, Pre-conditions, and post-conditions





Introduction to Graphs

Inter-data Relationships

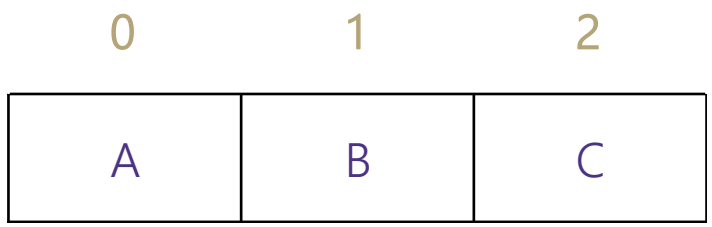
Arrays

Categorically associated

Sometimes ordered

Typically independent

Elements only store pure data, no connection info



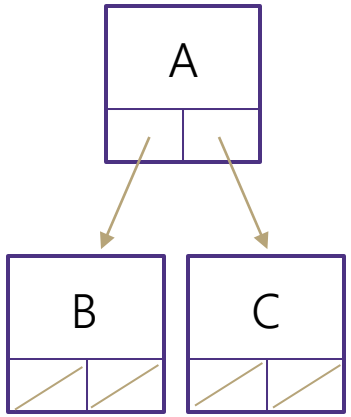
Trees

Directional Relationships

Ordered for easy access

Limited connections

Elements store data and connection info



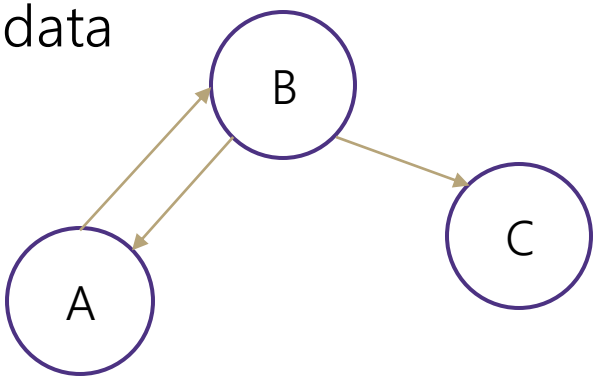
Graphs

Multiple relationship connections

Relationships dictate structure

Connection freedom!

Both elements and connections can store data



Graph: Formal Definition

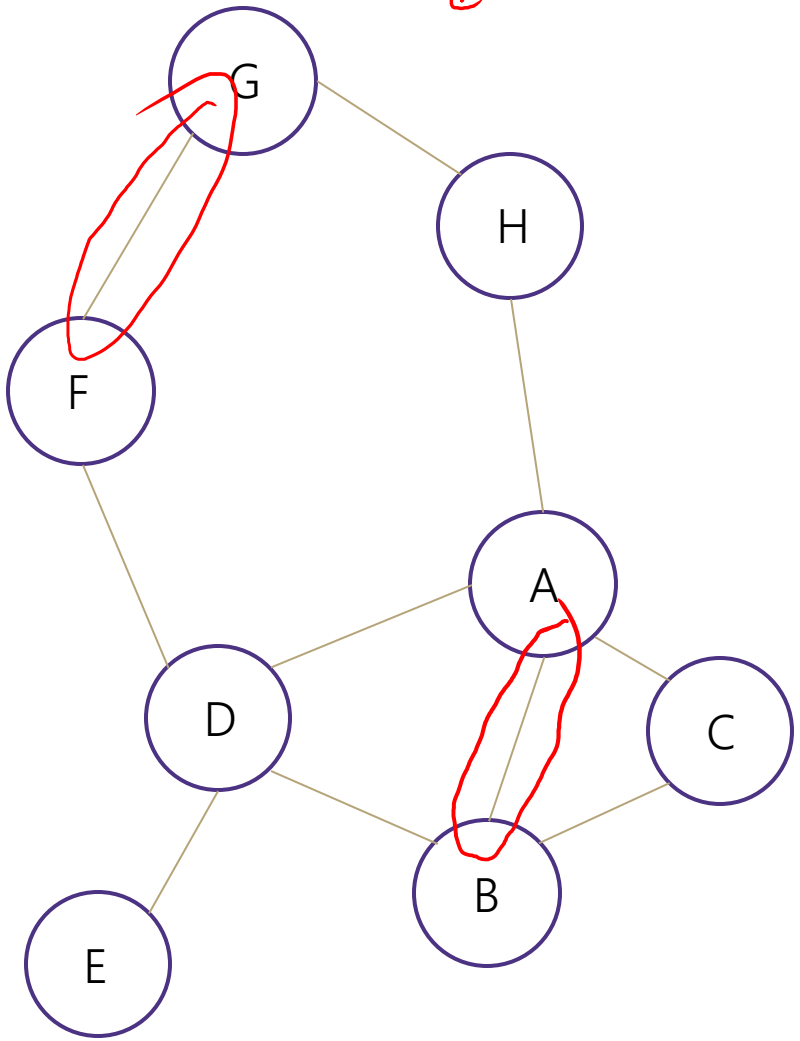
A **graph** is defined by a pair of sets G = (V, E) where…

- V is a set of **vertices**
  - A vertex or "node" is a data entity

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

- E is a set of **edges**
  - An edge is a connection between two vertices

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

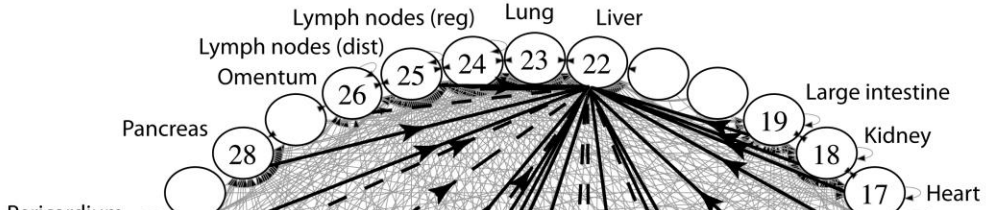
Relationships
- Social media graphs
  - Vertices are accounts, edges are follower relationships
- Code bases
  - Vertices are classes, edges are usage

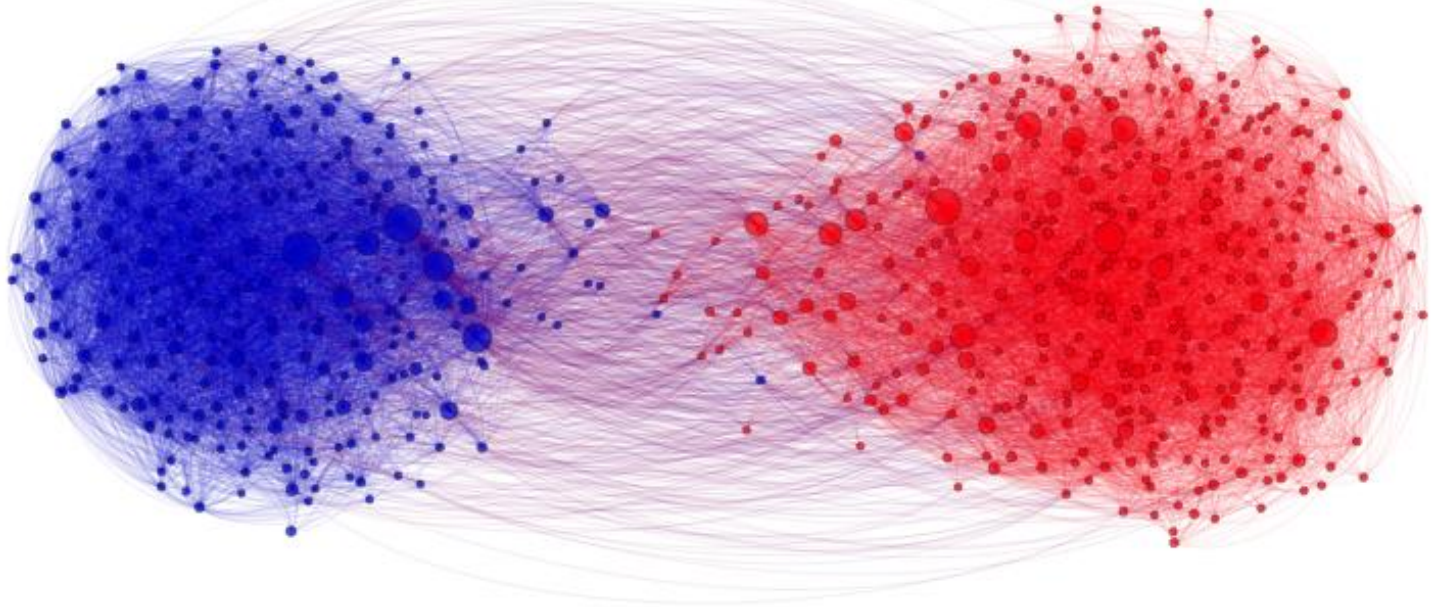
Influence
- Biology
  - Vertices are cancer cell destinations, edges are migration paths

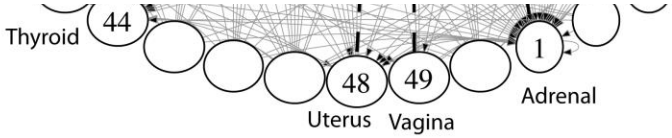
Related topics
- Web Page Ranking
  - Vertices are web pages, edges are hyperlinks
- Wikipedia
  - Vertices are articles, edges are links

SO MANY MORREEEE

www.allthingsgraphed.com







Graph Vocabulary

Graph Direction

- **Undirected graph** – edges have no direction and are two-way

  V = { A, B, C }

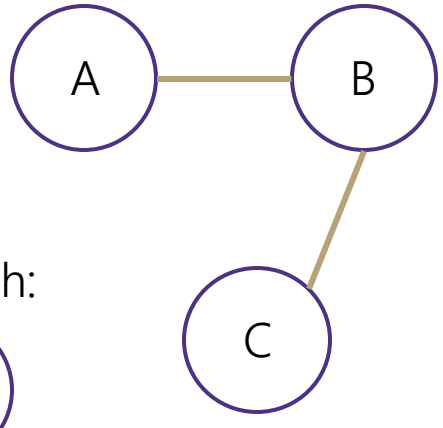
  E = { (A, B), (B, C) } *inferred (B, A) and (C,B)*

- **Directed graphs** – edges have direction and are thus one-way

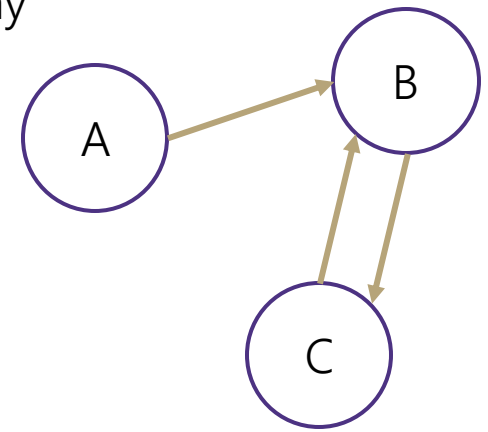
  V = { A, B, C }

  E = { (A, B), (B, C), (C, B) }

Undirected Graph:



Undirected Graph:



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

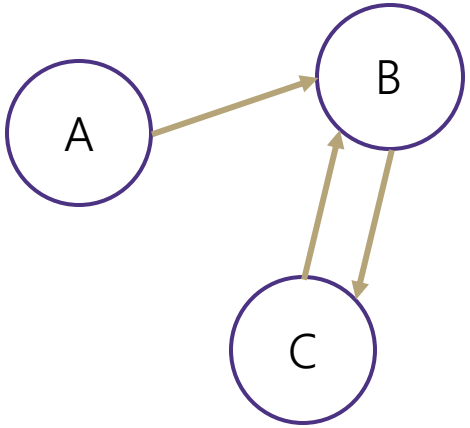
  A : 0, B : 2, C : 1

- **Out-degree** – the number of directed edges that start at a vertex

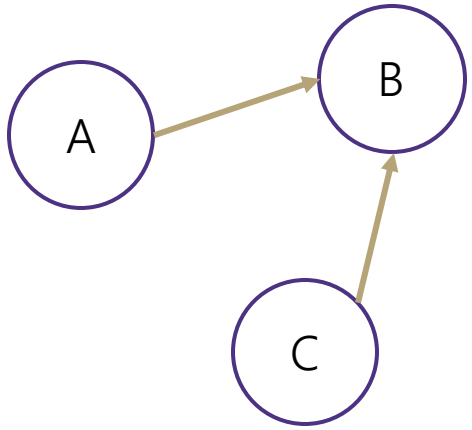
  A : 1, B : 1, C : 1

Food for thought

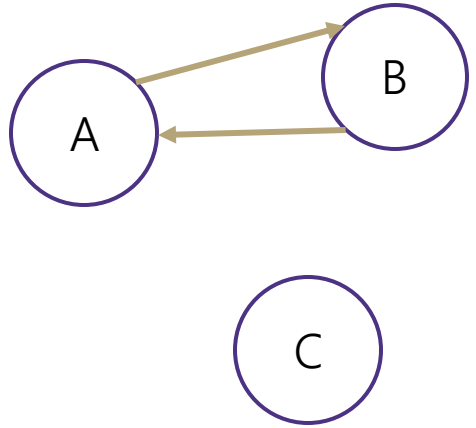
Is a graph valid if there exists a vertex with a degree of 0? Yes



A has an "in degree" of 0

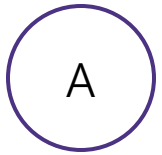


B has an "out degree" of 0



C has both an "in degree" and an "out degree" of 0

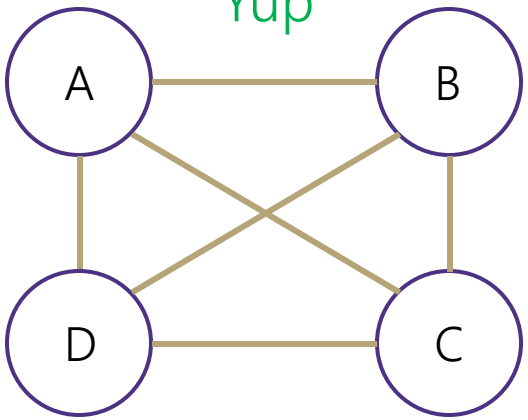
Is this a valid graph?

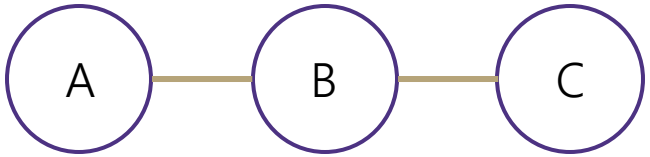


Yes!

Are these valid?

Yup

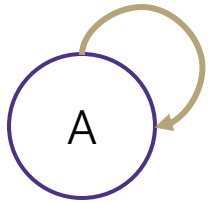




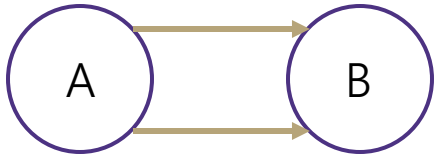
Sure

Graph Vocabulary

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



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



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