DSA3 Lab session 3

ATTENTION

- Do NOT modify your test files
- Do NOT hardcode for specific test cases

Stable sort

- What does stable sort mean?

- What advantages are there to a sorting algorithm that is stable?

Why learn so many sorts?

- Why not just remember the one sort with the best Big O and use it everywhere?

- Insertion sort is $O(n^2)$
- Mergesort is O(n log n)

- Why even teach you insertion sort?
  - Hold that thought…

What’s the best and worst data distro for…

- Insertion sort?

- Bubble sort?

What's the best and worst data distro for...

- Insertion sort?
  - Already sorted list is just O(n)
  - Reverse sorted list is $O(n^2)$

What's the best and worst data distro for...

- Bubble sort?
  - Already sorted list is just O(n)
  - Reverse sorted list is $O(n^2)$

  - THIS ASSUMES AN OPTIMIZATION

What's the best and worst data distro for…

- Merge?
- Quick?
  - Assumption: last index pivot as you saw in class
- Bucket?

- Merge?
  - Merge is too cool to care about data

What's the best and worst data distro for…

- Quick
  - Balanced random data is $O(n \log n)$
  - Sorted/reverse sorted data is $O(n^2)$

What’s the best and worst data distro for…

- Bucket
  - Uniform bucket distribution can be O(n)
  - Highly skewed data makes bucket effectively uselesa

Considerations other than speed

- Merge sort
  - Worst: O(n log n)
  - Average: O(n log n)
  - Best: O(n log n)
  - Stable: yes
- John von Neumann, 1945

- Quicksort
  - Worst: O(n2)
  - Average: O(n log n)
  - Best: O(nlog n)
  - Stable: no
- C.A.R. Hoare, 1959

Some practical examples for better understanding

- $S = [s_1, s_2, s_3, \ldots s_n]$ $s_i \in \{0, 1\}$

- How would mergesort handle this?
  - Imagine the data actually flowing through the algorithm

Some practical examples for better understanding

- $S = [s_1, s_2, s_3, \ldots s_n] \quad s_i \in \{0, 1\}$

- How would quicksort handle this?

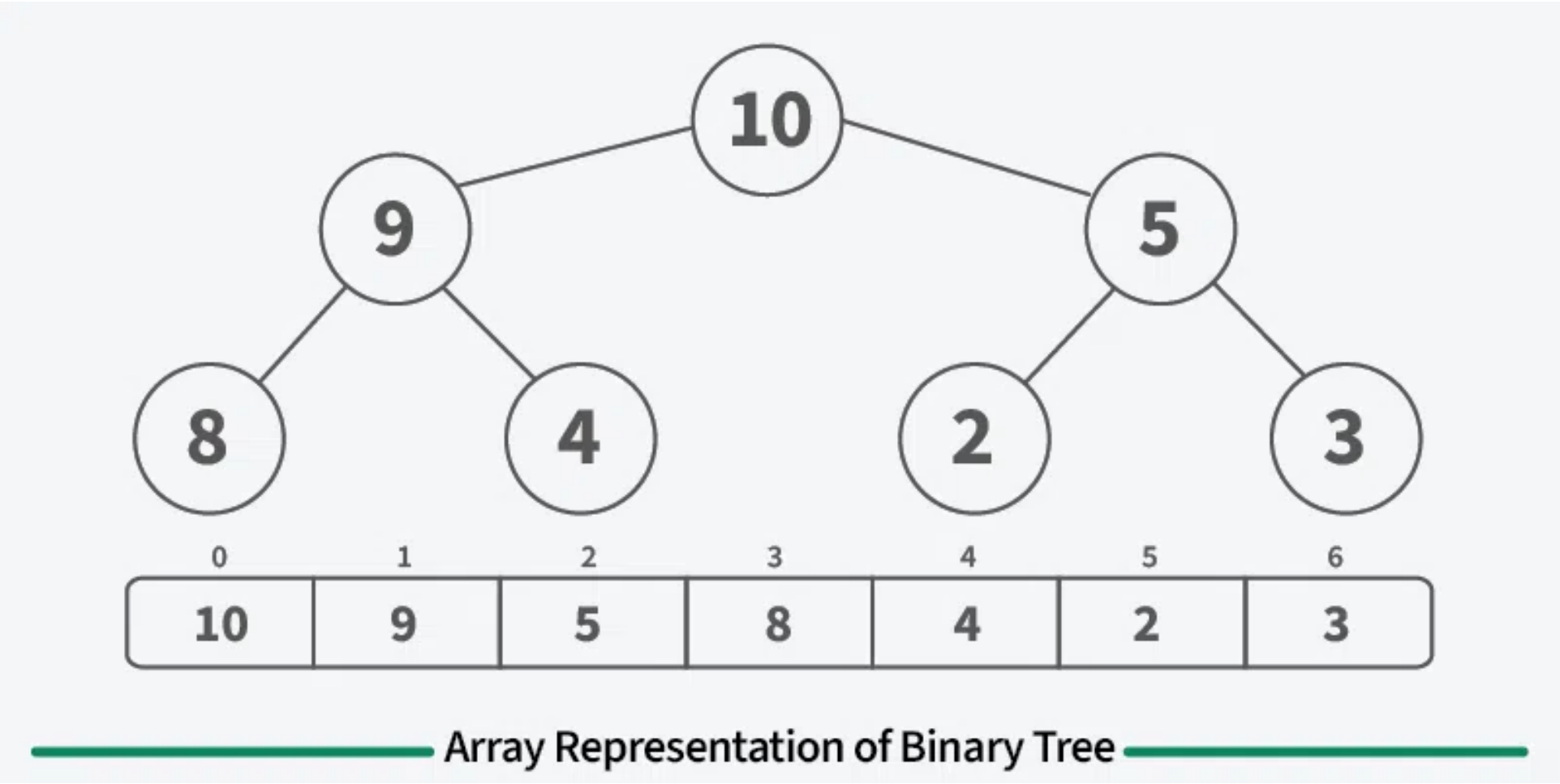
```
def qsort(seq):
  if len(seq <= 1): return seq
  return qsort([x for x in seq if x <  seq[-1]])\ # < p
       +       [x for x in seq if x == seq[-1]]\  # = p
       + qsort([x for x in seq if x >  seq[-1]])  # > p
```

A deeper dive into Mergesort

- We've basically assumed an array-based implementation of mergesort

- What if we implement it as a linked list?
  - Are there any ways this can overperform or underperform array-based merge?

Tree as array… how to implement



Root
Parent
Left
Right
Is_Leaf

Another way to use trees