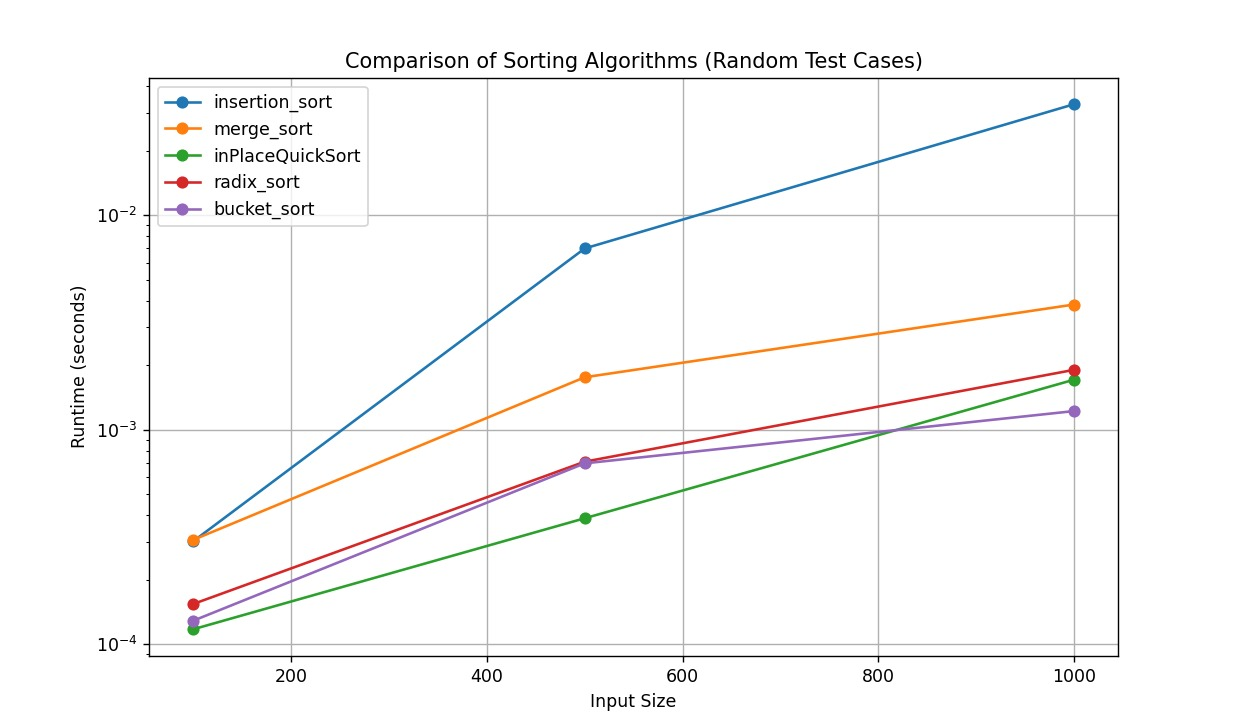
19CSE302: Design and Analysis of Algorithms

Sorting Algorithms

Complexity Analysis.





1. In-place Quick Sort:

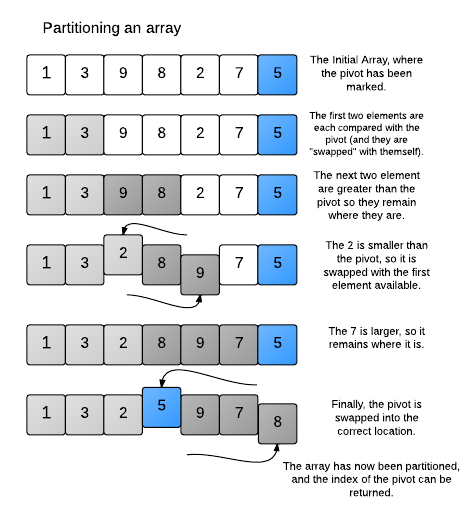
divide-and-conquer paradigm

comparison-based sorting by selecting a "pivot"

left sub-array {elements < pivot}

right sub-array {elements > pivot}

recursive sorting of sub-arrays



Cases:

Best case:

pivot chosen is median and leads to balanced partitions

O(n log n)

Worst case:

pivot is the smallest or largest element in the array leading in highly unbalanced partitions

O(n^2)

Average case:

pivot chosen is random

O(n log n)

1. Merge Quick Sort:

divide-and-conquer, stable sorting paradigm

does require additional memory space for the temporary arrays used in merging

divide the input into smaller halves, sort each half recursively, and then merge the sorted halves



Cases:

Best case:

input array is already sorted, no swapping cost while merging

O(n log n)

Worst case:

input array is in reverse order

O(n log n)

Average case:

random input array

O(n log n)

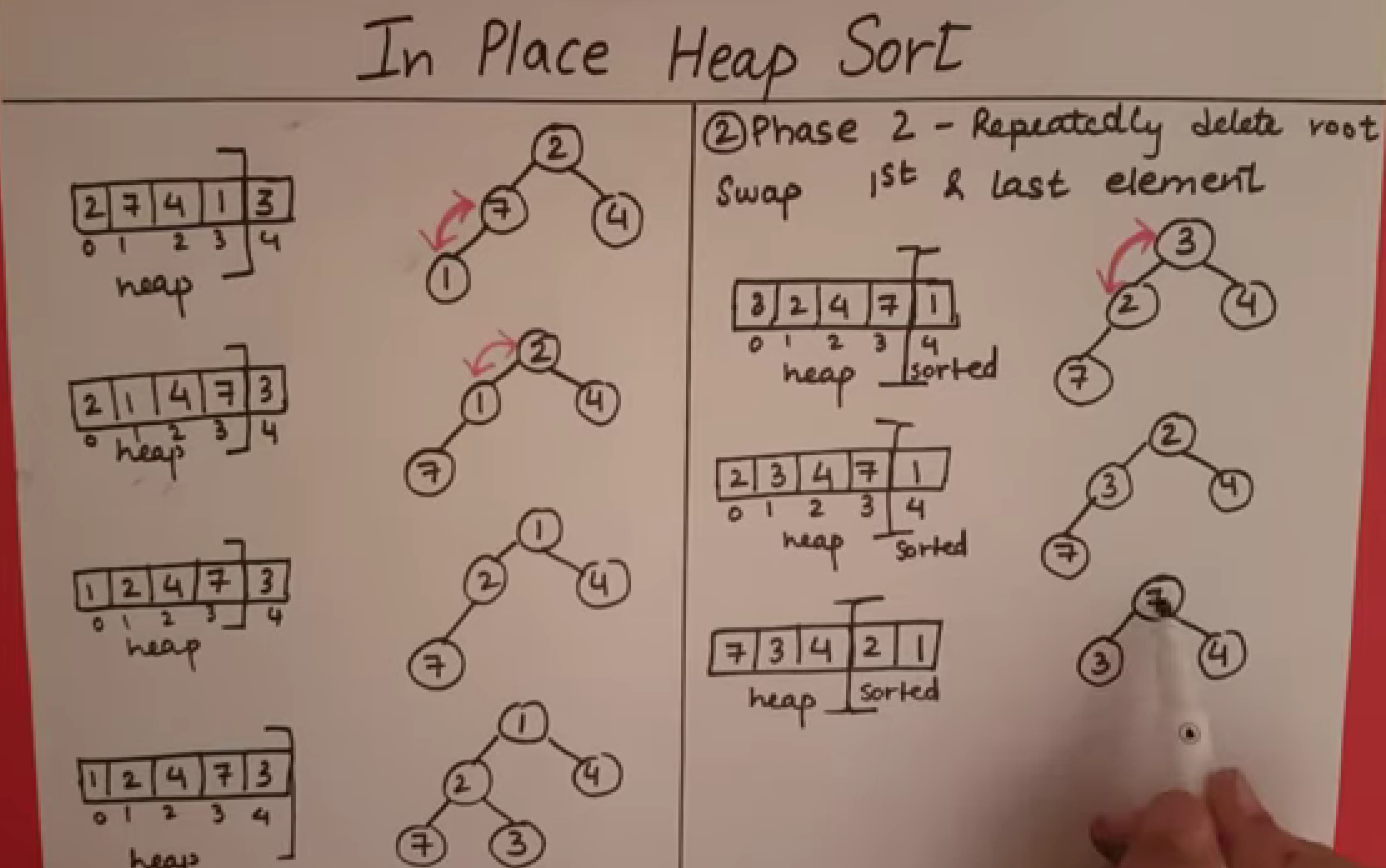
1. In-place Heap Sort:

doesn’t use additional memory

build max\_heap using heapify function

remove the root and insert it in the last of array

heapify again



Cases:

Best case and Average case:

random input array

O(n log n)

Worst case:

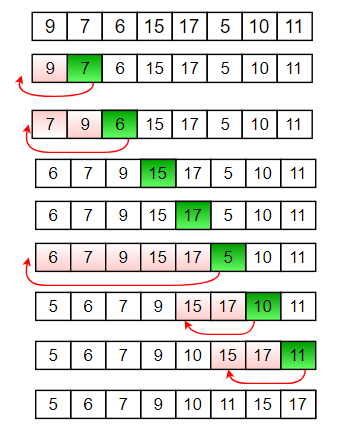
input array is in reverse order

O(n log n)

1. Insertion Sort:

divided into sorted and unsorted arrays

elements from the unsorted part are picked and placed at the correct position in the sorted part



Best case:

input array is already sorted or partially sorted and when the size is small

O(n)

Worst case:

input array is in reverse order

O(n^2)

Average case:

random input array

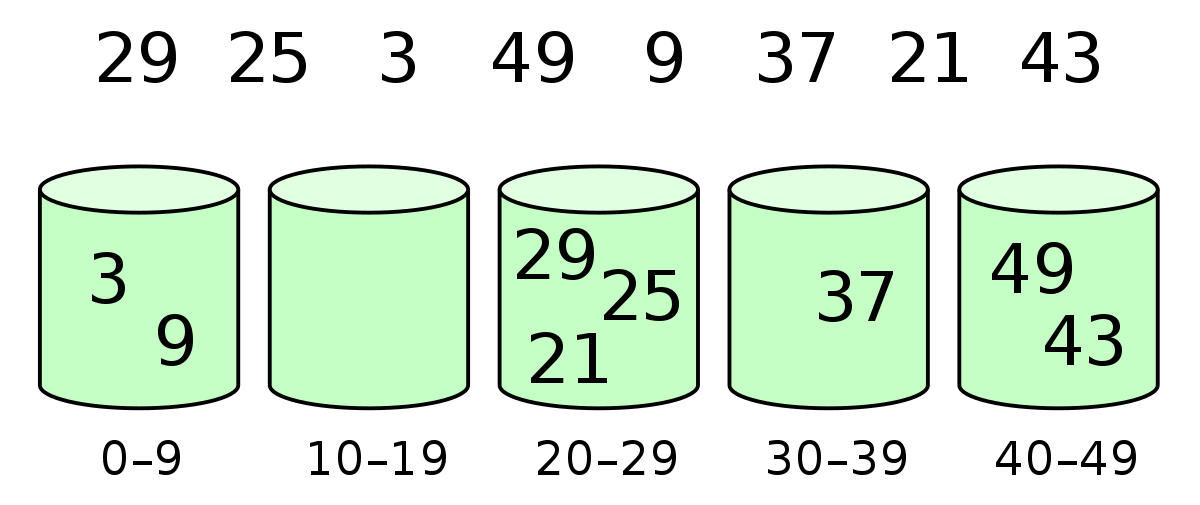
O(n^2)

1. Bucket Sort:

divide elements into various buckets

sort the buckets using insertion sort

concatenate the sorted buckets



Best case:

input data is uniformly distributed across the buckets

O(n + k)

Worst case:

all elements are placed in a single bucket

O(n^2)

Average case:

elements are reasonably evenly distributed across the buckets, but not perfectly uniform

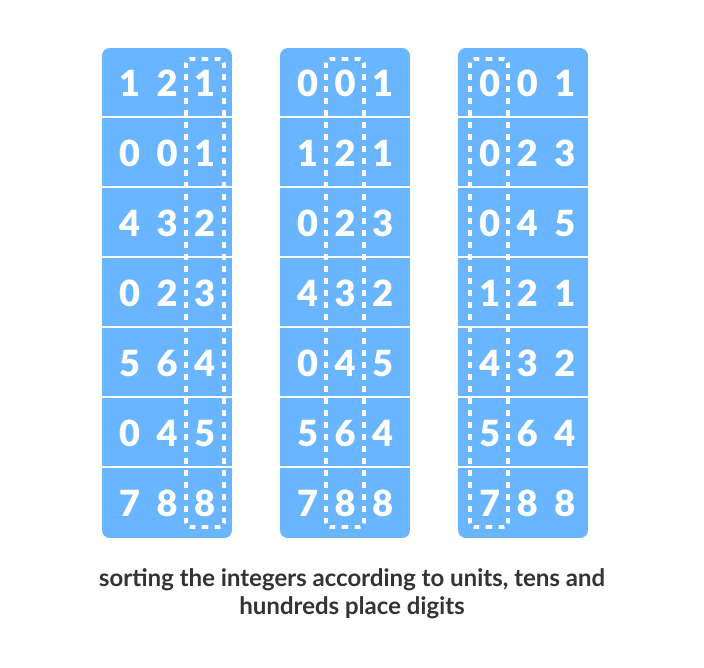
O(n^2)

1. Radix Sort:

processing them digit by digit from least significant digit

elements are put into buckets based on the value of the current digit

concatenate the sorted



Best case:

all the elements in the input array have the same number of digits

O(n \* k)

Worst case:

elements have significantly different number of digits

O(n \* k)

Average case:

random range of values

O(n^2)