course_content

top

W7: Sorting and Searching

W8: Graphs

W9: Trees

W10: Heaps

W11: Greedy Algorithms

Data Structures and Algorithms

Searching and Sorting

Week 7

Time:
5 min

Research Exercise

- Groups of 3-4
- Over the week, you will analyse a sorting algorithm.
- You should explain how it works through a demonstration - anything of your choice
- You should analyse it’s run time explaining how it works in best case, average case and worst case as well as space complexity
- Is there recursion or iteration or both in your algorithm?
- Presentations now!

Time:
5 min

Sorting Algorithms

- Best-case
- Average-case
- Worst-case
- Memory
- Recursion?
- In-place (Internal or External)
- Stable

Time:
5 min

BubbleSort

- Loops through multiple times exchanging adjacent item in the wrong order
- Largest item gets pushed to end, then second largest etc
- Many unnecessary exchanges
- Can force it to stop if no exchanges in a pass
- Each item ‘bubbles’ to its place
- Good Memory usage
- Poor Time usage

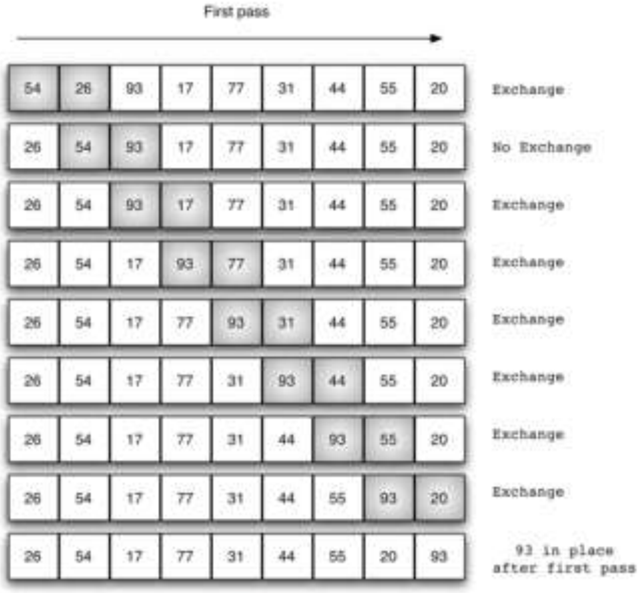


Figure 1: bubbleSort : The First Pass

Time:
5 min

SelectionSort

- Improvement on BubbleSort - one exchange per pass
- Finds max and moves to end repeatedly
- Same number of comparisons, fewer exchanges
- Can’t stop if max is already at the end
- ‘Selects’ maximum and sorts it
- Good memory usage
- Still poor time usage but better than Bubble

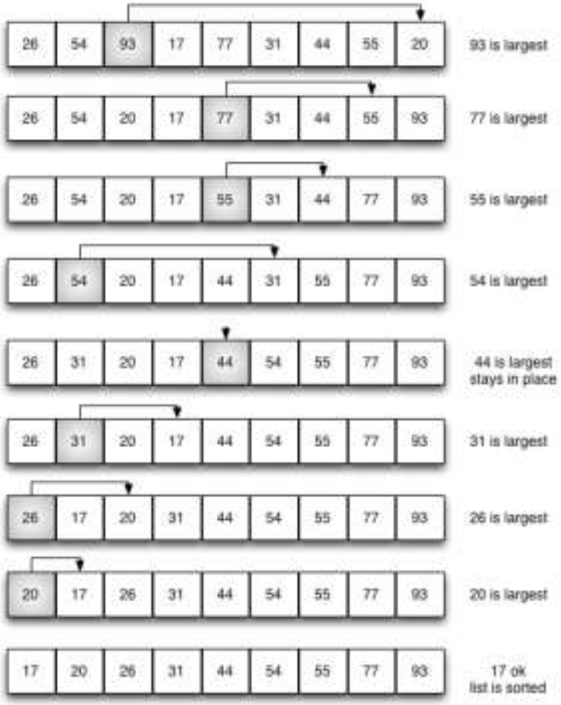


Figure 3: selectionSort

Time:
5 min

InsertionSort

- Builds sorted array from the start by taking k sorted elements and inserting next element in the right place moving items over until correct place is found
- Does this for k=1....n-1 until all elements are
- Shifting is faster than exchanging (⅓ time)
- 'Inserts' items into the right place
- Each pass can take just one comparison (best)
- Good memory usage
- Poor time usage but better than selection

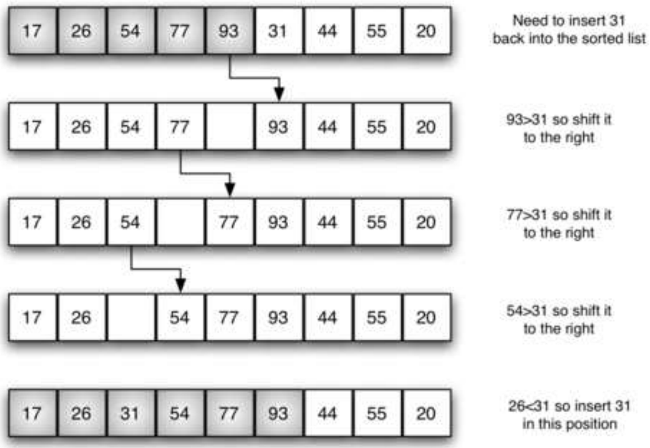


Figure 5: `insertionSort` : Fifth Pass of the Sort

Time:
5 min

ShellSort

- Breaks down list into sublists based on increment/gap (e.g. 3)
- Applies insertion sort on each sublist
- Applies a full insertion sort again at the end
- Breaks down array into 'shells'
- Initial insertions make final one much faster
- Good memory usage
- Much better time usage (~$n^{4/3}$)

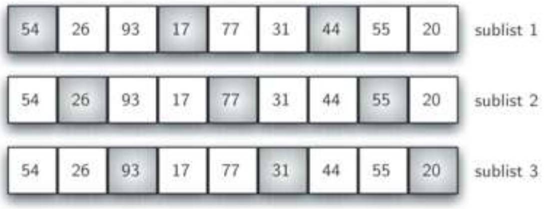


Figure 6: A Shell Sort with Increments of Three

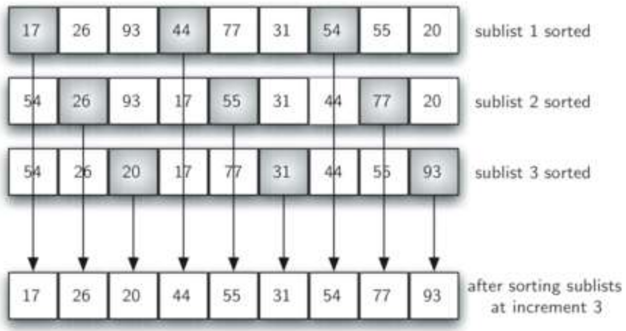


Figure 7: A Shell Sort after Sorting Each Sublist

Time:
5 min

MergeSort

- Breaks down list into half each time until small enough to trivially sort
- Then merges two ordered list into one ordered list
- Does this repeatedly until largest lis
- Space to store each half
- Highly Parallelizable
- Poor memory
- Great time (nlogn)

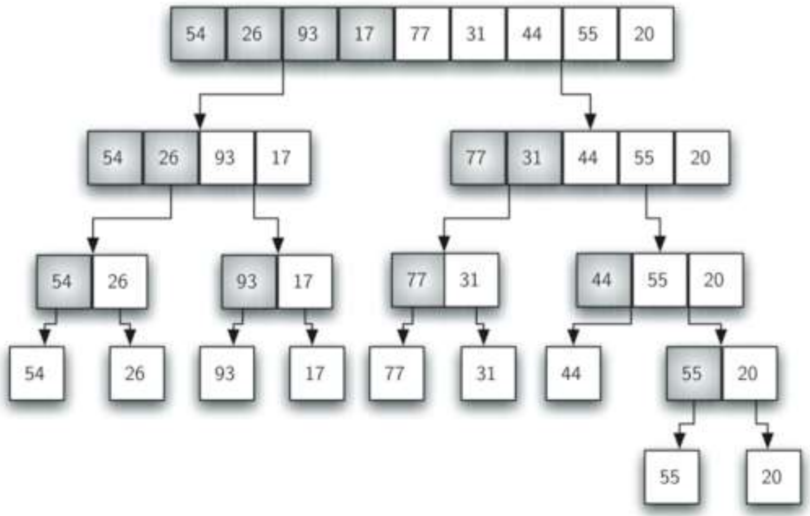
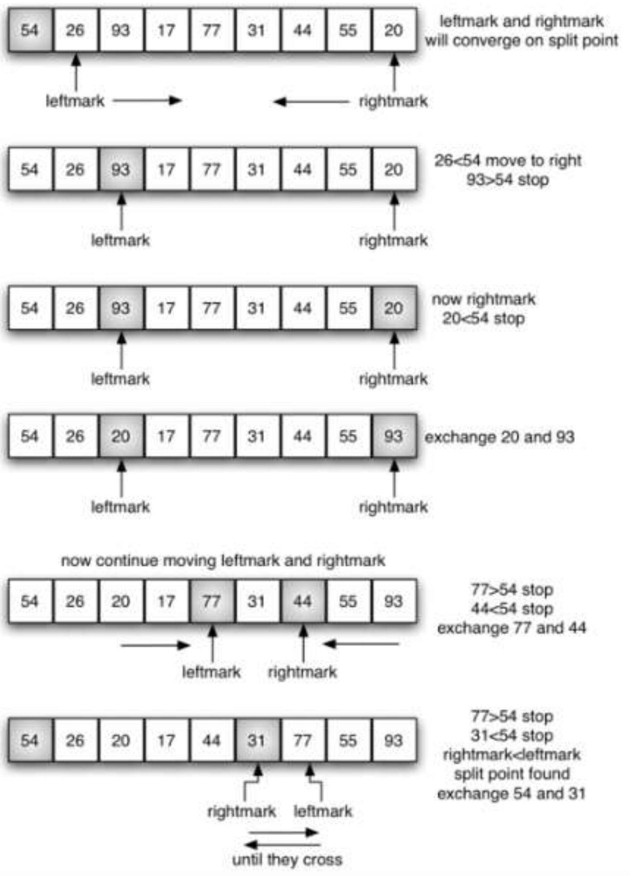


Figure 10: Splitting the List in a Merge Sort

Time:
5 min

QuickSort

- Divide and Conquer but no storage used
- Choose pivot, move to split point using partition
- Locate leftmark and rightmark
- Move items on the wrong side to the right side
- Find new location of pivot
- N^2 worst case and nlog n average case
- Good memory usage
- Fairly complex to implement



Time:
5 min

Sorting Visualisations

Click here to see how these algorithms work

Time:
1 min

Questions?