2. Insertion Sort, Merge Sort

Link:https://www.youtube.com/watch?v=Kg4bqzAqRBM&t=21s&index=5&list=PLSX2U_ZE4Huk19DPn34oZlygPbsig380X

Finding a median array in A[0:n] (unsorted) -> B[0:n] (sorted) (B[n/2]

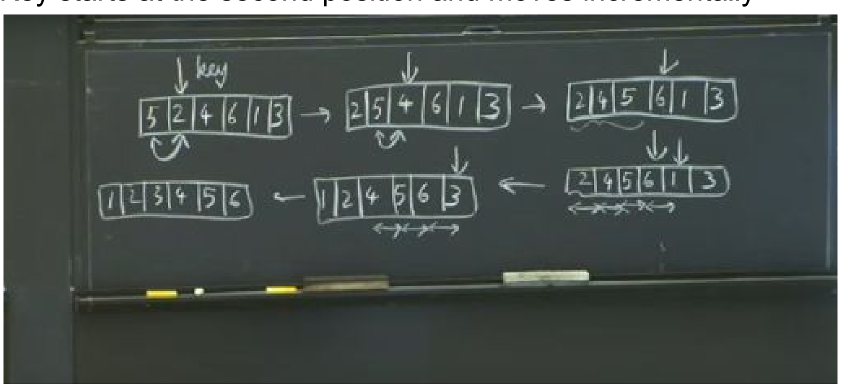
- It's constant time to find the median once you have a sorted list

Uses of Sorting Algorithms

- Binary Search
    - A[0:n] looking for specific item can take linear time if searching one by one
    - B[0:n] takes logarithmic time to complete if array is sorted by dividing the list in half

- Compression Algorithms
    - If data exists multiple times in a file, it can be represented as one item and duplicated upon decompression
    - Document distance
    - Computer graphics

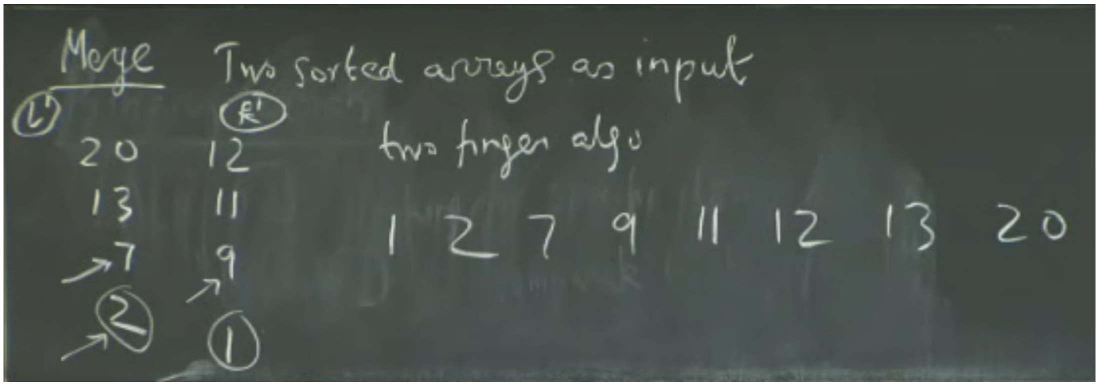
Insertion Sort

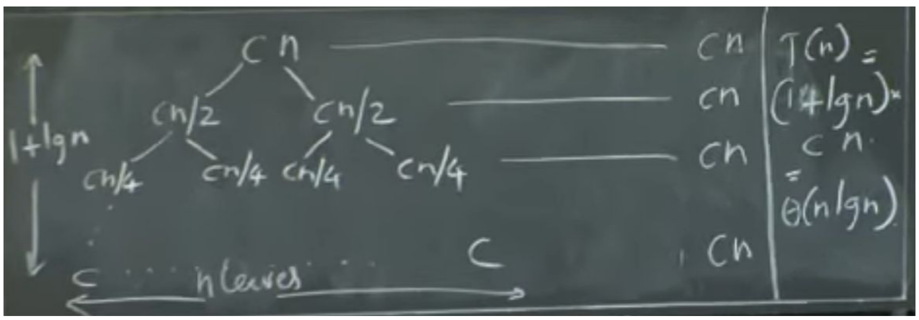
- For i = 1,2, ..n
    - Insert A[i] into sorted array A[0:i-1] by pair wise swaps down to the correct position
    - Key starts at the second position and moves incrementally
    - 

    - O(n) steps (key positions)
    - Each step is O(n) comparisons and swaps
    - If compares are more computationally expensive than swaps and you're more concerned about O(n^2) comparison cost than O(n) swap cost:
        - You can insert A[1] into sorted array A[0:I-1]by pairwise swaps (binary search) down to the correct position

Merge Sort

- Divide and conquer recursive algorithm

- Merge - two sorted arrays as input
- 

- The overall complexity is O(nlgn)
- Complexity T(n) = C1 (Divide) + 2T(n/2) (Recursion) + c * n (Merge component)
- Proof
- 

- 2T(n/2) + O(n) + O(1)

Comparison between Insert and Merge Sort

- The advantage insertion sort has over merge sort has to do auxiliary space
  - Merge sort requires O(n) auxiliary space in order to recursively handle the split arrays
  - Insertion sort has O(1) auxiliary space since it’s done in place
    - Great for high volume arrays
  - In place merge sort
    - Impractical
  - Merge sort in python = 2.2nlg(n) ms
  - Insertion sort in python = .2n^2 ms
  - Insertion sort in c = 0.01n^2 ms