Sorting Algorithms

Sorting Algorithms

- A **sorting algorithm** is an algorithm that puts elements of a list in a certain order.
- Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists

Why Sorting?

- "When in doubt, sort" –one of the principles of algorithm design. Sorting used as a subroutine in many of the algorithms:
  - Searching in databases: we can do binary search on sorted data
  - A large number of computer graphics and computational geometry problems
  - Closest pair, element uniqueness, frequency distribution

Why Sorting? (2)

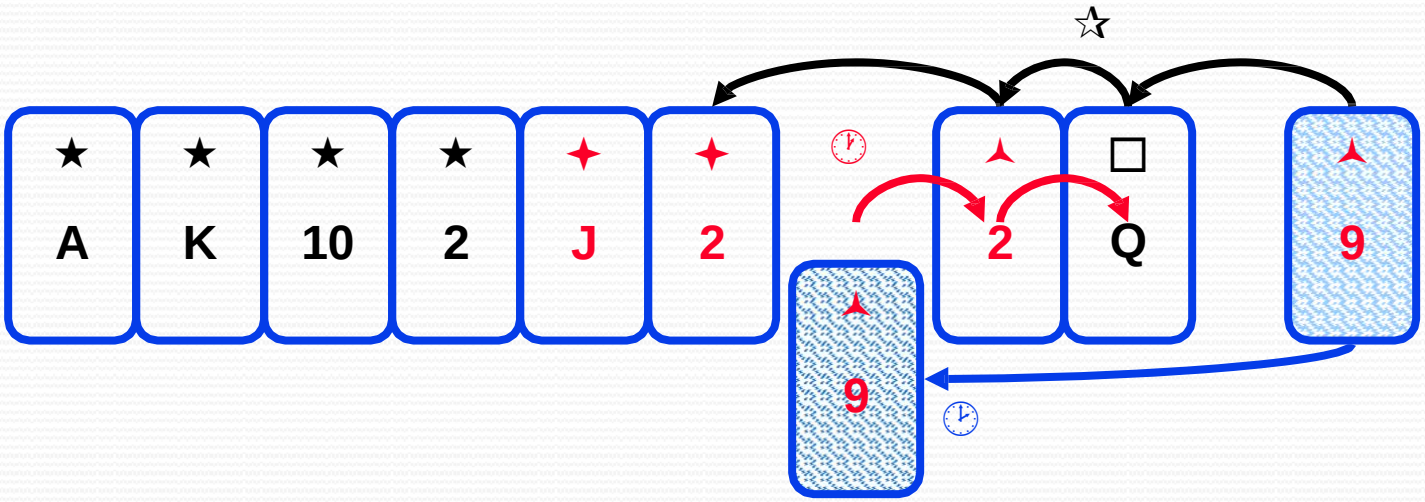
- A large number of algorithms developed representing different algorithm design techniques.
- A lower bound for sorting $\Omega(n \log n)$ is used to prove lower bounds of other problems

Sorting Algorithms so far

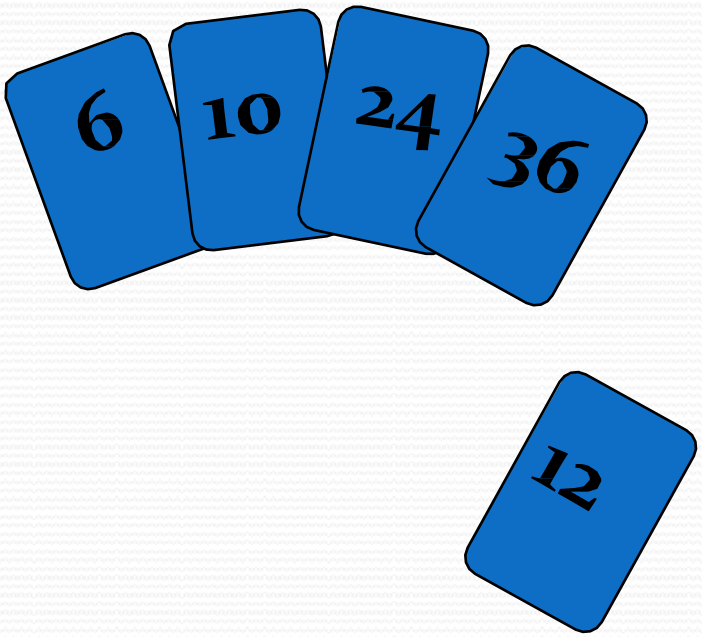
- Insertion sort, selection sort, bubble sort
  - Worst-case running time $\Theta(n^2)$; in-place
- Merge sort
  - Worst-case running time $\Theta(n \log n)$; but requires additional memory

Insertion sort

- Card players all know how to sort …
  - First card is already sorted
  - With all the rest,
    - ☆ Scan back from the end until you find the first card larger than the new one,
    - ⌘ Move all the lower ones up one slot
    - 🕑 insert it

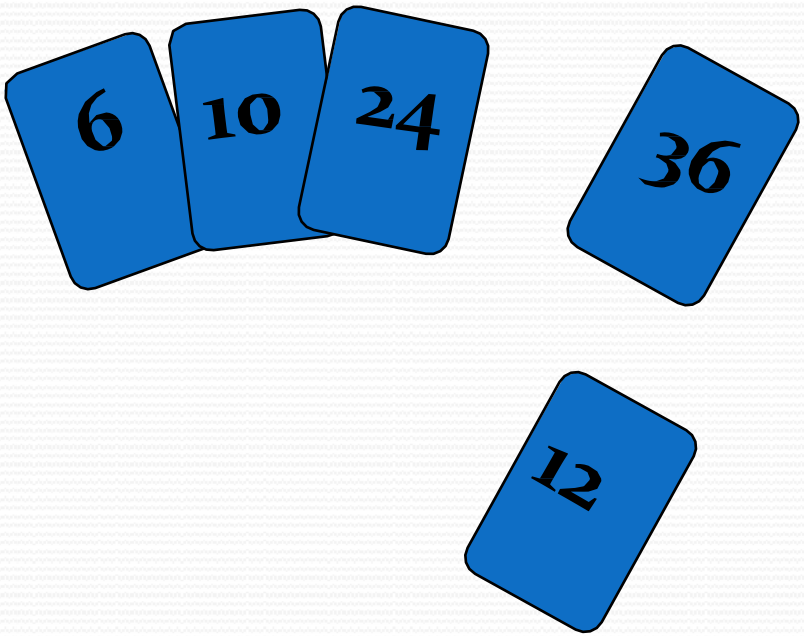


Insertion Sort

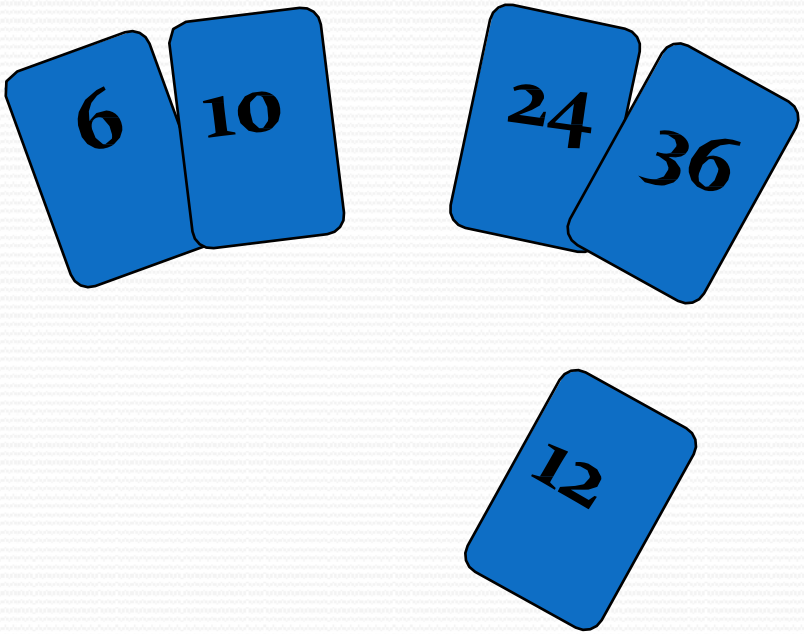


To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



Insertion Sort

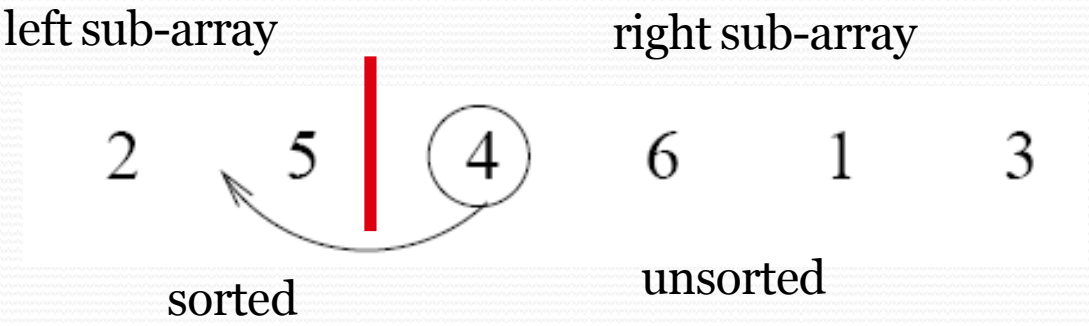


Insertion Sort

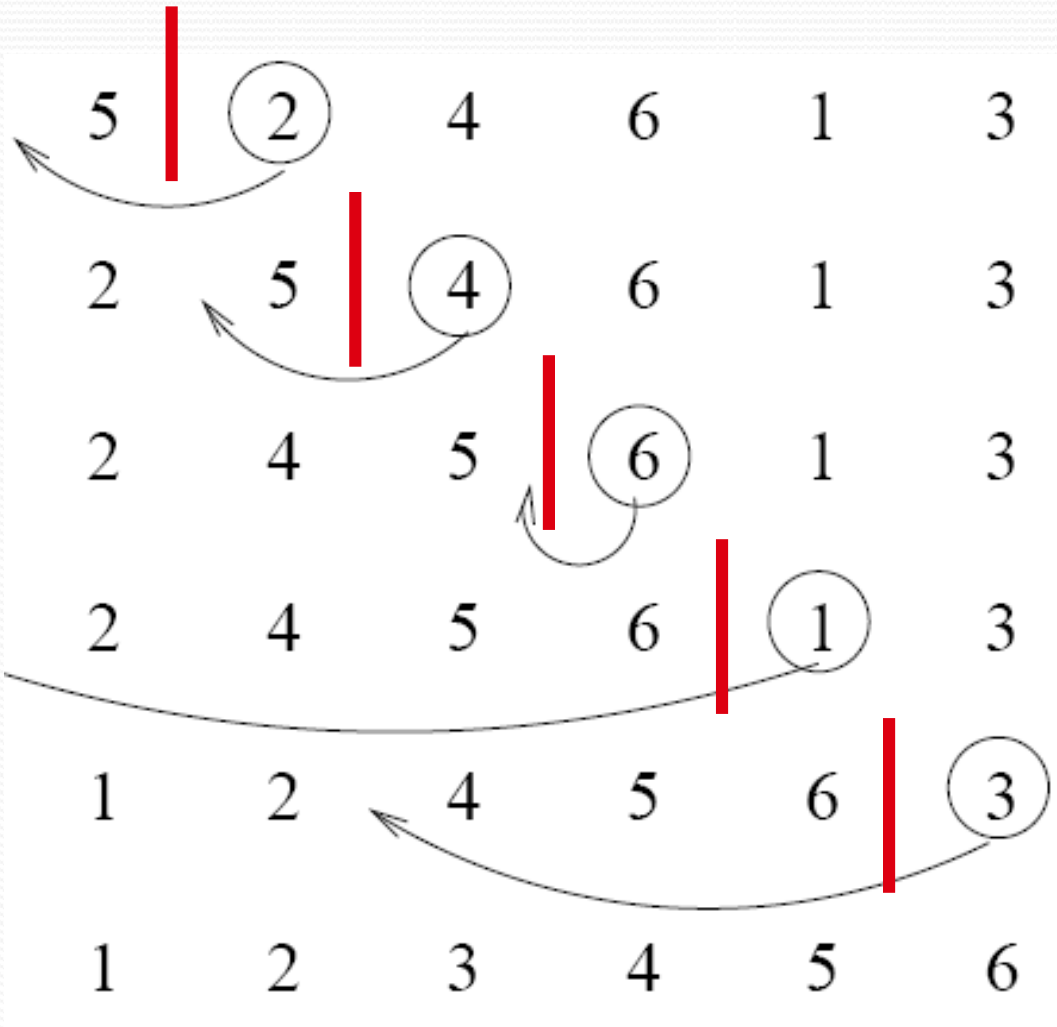
input array

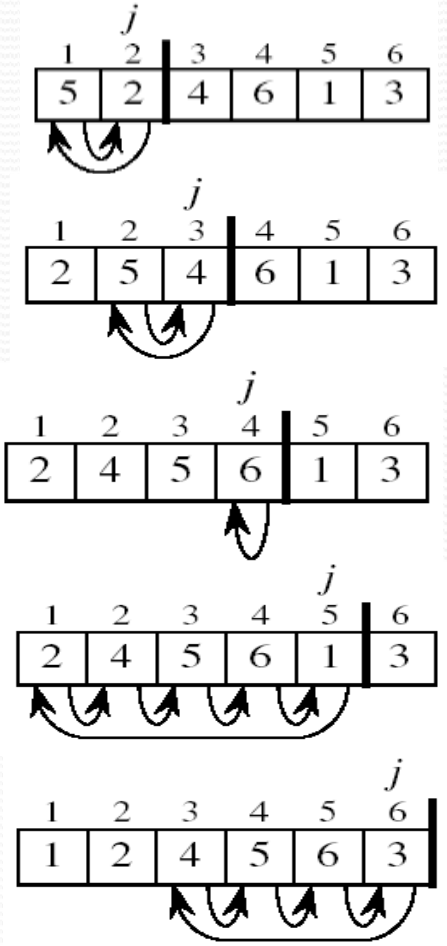
5 2 4 6 1 3

at each iteration, the array is divided in two sub-arrays:



Insertion Sort





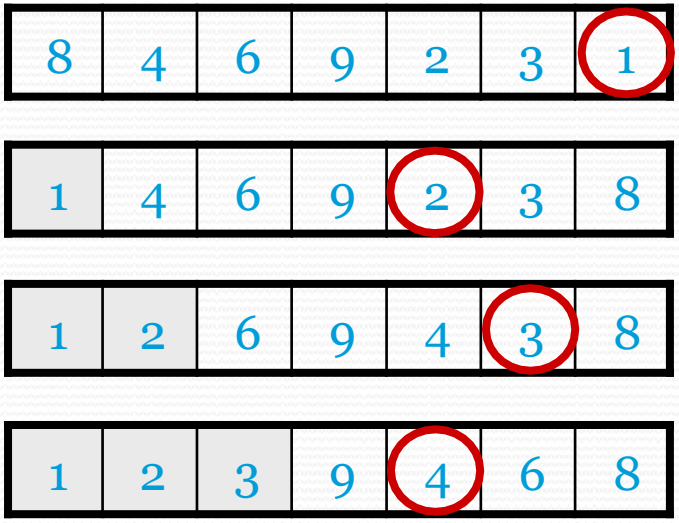
Insertion Sort - Summary

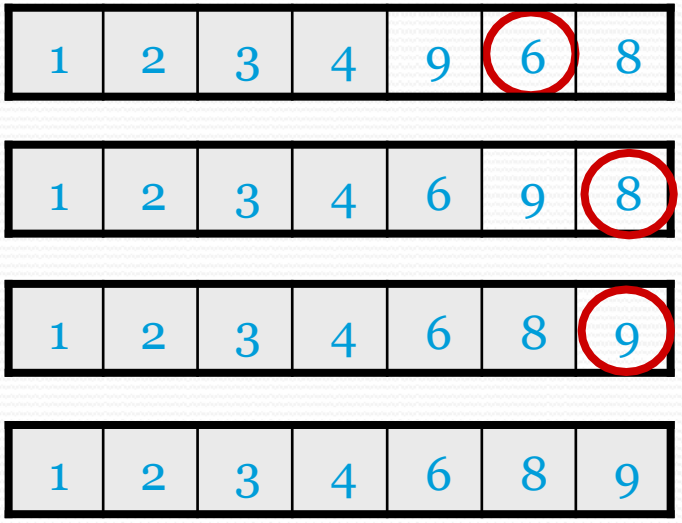
- Advantages
  - Good running time for "almost sorted" arrays $\Theta(n)$
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case
  - $\approx n^2/2$ comparisons and exchanges

Selection Sort

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted
- Disadvantage:
  - Running time depends only slightly on the amount of order in the file

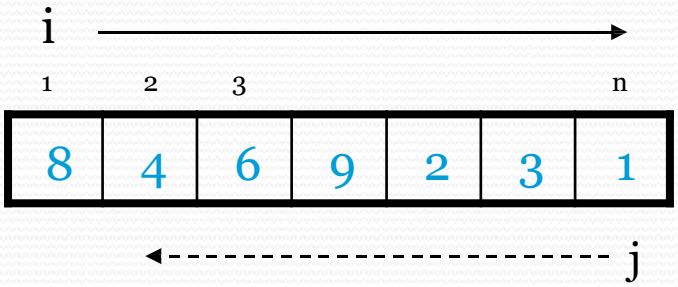
Example-Selection Sort





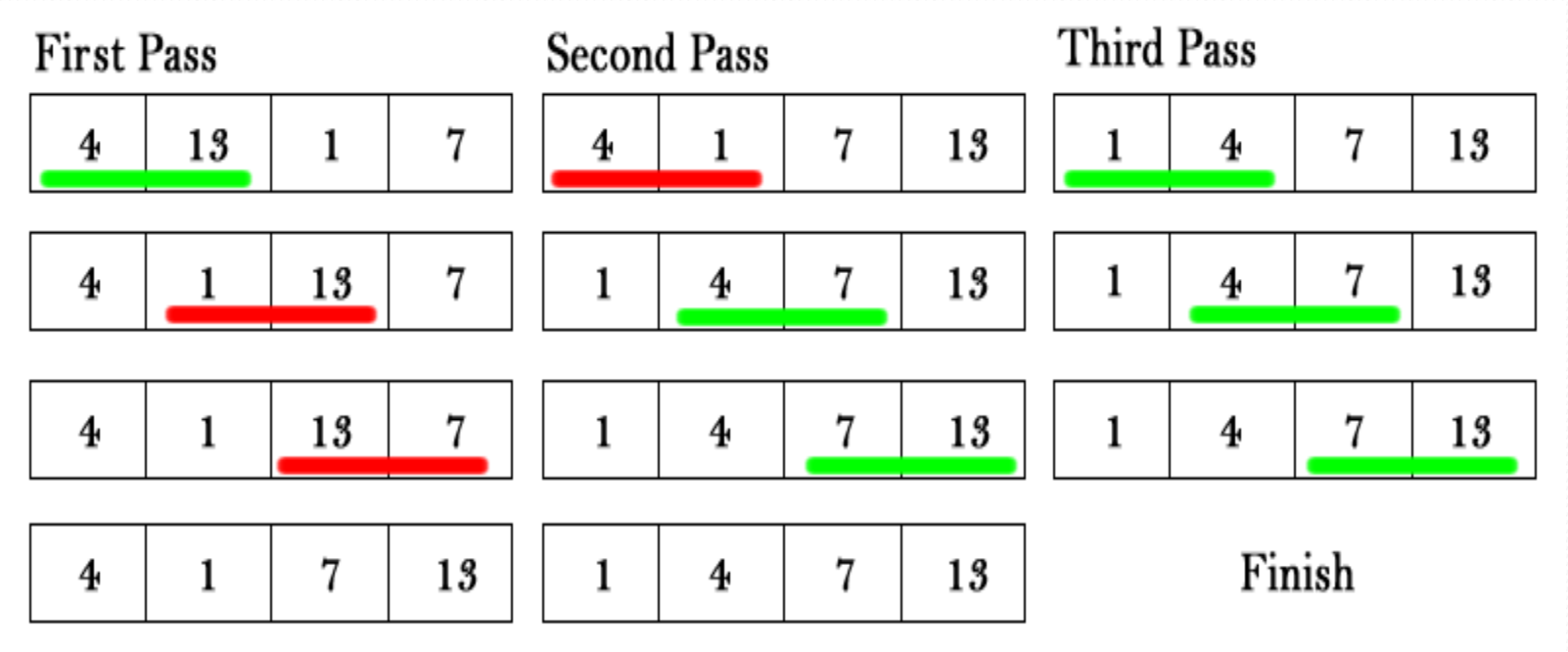
Bubble Sort

- Idea:
  - Repeatedly pass through the array
  - Swaps adjacent elements that are out of order



- Easier to implement, but slower than Insertion sort

Example - Bubble Sort



Quicksort

- Efficient sorting algorithm
  - Discovered by C.A.R. Hoare
- Example of Divide and Conquer algorithm
- Two phases
  - Partition phase
    - Divides the work into half
  - Sort phase
    - Conquers the halves!

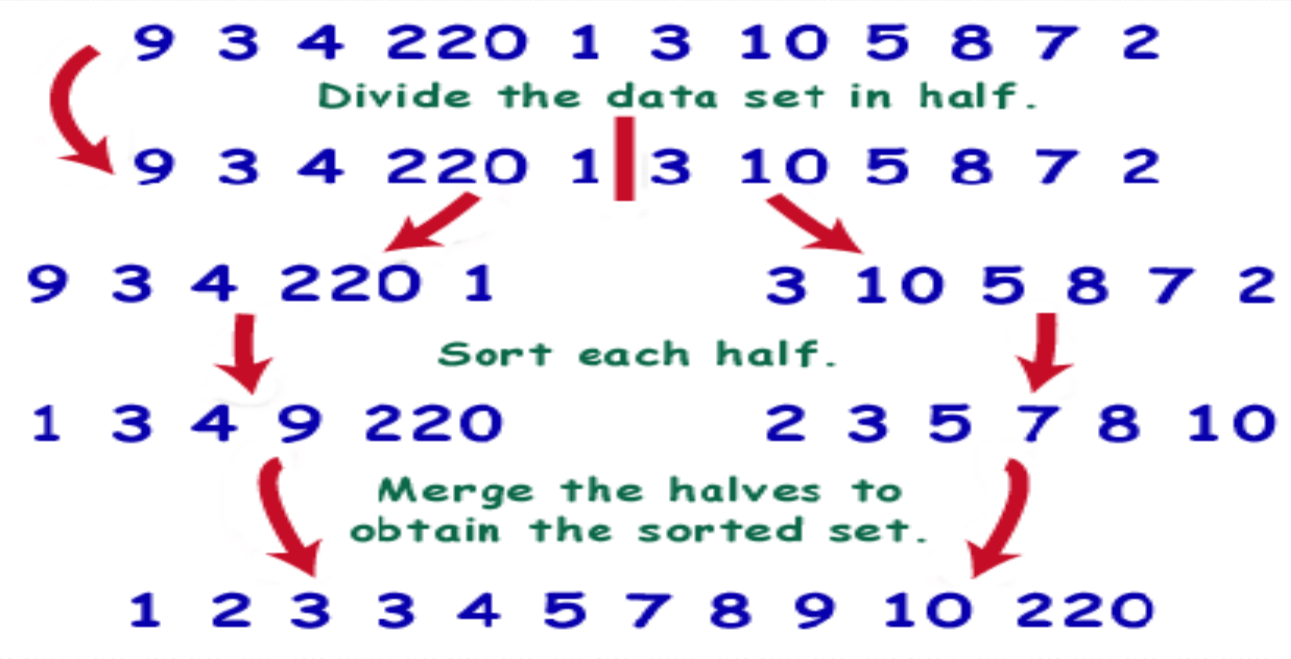
Quicksort

- Partition / Divide
  - Choose a pivot
  - Find the position for the pivot so that
    - all elements to the left are less
    - all elements to the right are greater

Quicksort

- Conquer
  - Apply the same algorithm to each half

Quicksort - Example



Review of Algorithms

- **Selection Sort**
  - An algorithm which orders items by repeatedly looking through remaining items to find the least one and moving it to a final location
- **Bubble Sort**
  - Sort by comparing each adjacent pair of items in a list in turn, swapping the items if necessary, and repeating the pass through the list until no swaps are done
- **Insertion Sort**
  - Sort by repeatedly taking the next item and inserting it into the final data structure in its proper order with respect to items already inserted.
- **Merge Sort**
  - An algorithm which splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence
- **Quick Sort**
  - An in-place sort algorithm that uses the divide and conquer paradigm. It picks an element from the array (the pivot), partitions the remaining elements into those greater than and less than this pivot, and recursively sorts the partitions.

THANK YOU……!!!