csci 210: Data Structures

Sorting
Problem

• Input: an array of elements that can be compared to each other
  • \( A = [x_1, x_2, x_3, \ldots, x_n] \)

• Output:
  • a new array \( B \) that contains the elements in \( A \) in increasing order
    • \( B[0] \leq B[1] \leq B[2] \ldots \)
  • or the same array \( A \), rearranged so that the elements are in increasing order

• Permuting the input array is advantageous because it does not use extra space (memory).

• A sorting algorithm that permutes the input array and does NOT create a new output array is called “in-place”
Sorting algorithms

- **Bubble-sort**
  - idea: a pass through the array swaps elements that are out-of-order
  - need n-1 passes in the worst-case
  - analysis: $O(n^2)$, in-place

- **Insertion sort**
  - idea: the elements processed so far are kept in order. take the next element in the input and insert it in the right spot in the sorted array.
  - analysis: $O(n^2)$, in-place

- **Selection sort**
  - idea: select the smallest, put it in position 0; select the next smallest, put it in position 1, and so on.
  - analysis: $O(n^2)$, in-place

- **Mergesort**
  - idea: split into 2 halves. sort each half recursively. merge.
  - analysis: $O(n \log n)$, not in-place

- **Quicksort**
  - idea: pick an element and call it pivot. re-arrange the input so that all element $\leq$ pivot are to its left, and all element $> pivot$ are to its right. sort each part recursively. no need of merging.
  - analysis: $O(n^2)$, in-place

---

no proof here [in csci 231]