### 1. Bubble Sort

**Idea:**

Repeatedly swap adjacent elements if they are in the wrong order.

**Steps:**

1. Traverse from index 0 to n-1.
2. Swap if arr[i] > arr[i+1].
3. After each pass, the largest element "bubbles up" to the end.

**Time Complexity:**

| Case | Complexity |
|---|---|
| Best | O(n) |
| Average | $O(n^2)$ |
| Worst | $O(n^2)$ |

- Space: O(1)

**Example Inputs:**

- Best: [1, 2, 3, 4, 5] (Already sorted)
- Worst: [5, 4, 3, 2, 1] (Reverse sorted)

### 2. Selection Sort

**Idea:**

Find the minimum element and move it to the correct position.

**Steps:**

1. For each position i, find the minimum from i+1 to n-1.
2. Swap it with arr[i].

**Time Complexity:**

| Case | Complexity |
|---|---|
| Best | $O(n^2)$ |
| Average | $O(n^2)$ |
| Worst | $O(n^2)$ |

- Space: O(1)

**Example Inputs:**

- All inputs take the same number of comparisons.

### 3. Insertion Sort

**Idea:**

Build sorted array one item at a time.

**Steps:**

1. Assume arr[0] is sorted.
2. Take arr[i], insert it into the sorted left part at correct position.

**Time Complexity:**

| Case | Complexity |
|---|---|
| Best | O(n) |
| Average | $O(n^2)$ |
| Worst | $O(n^2)$ |

- Space: O(1)

**Example Inputs:**

- Best: [1, 2, 3, 4, 5]
- Worst: [5, 4, 3, 2, 1]

### 4. Merge Sort

**Idea:**

Divide and conquer. Divide the array, sort each part, merge them.

**Steps:**

1. Divide array into halves until 1 element each.
2. Merge two sorted arrays back recursively.

**Time Complexity:**

| Case | Complexity |
|---|---|
| Best | O(n log n) |
| Average | O(n log n) |
| Worst | O(n log n) |

- Space: O(n) due to merging

**Example Inputs:**

- All cases: [4, 1, 5, 2, 6], [1, 2, 3, 4, 5], [5, 4, 3, 2, 1] → All same complexity.

### 5. Quick Sort

**Idea:**

Pick a pivot, partition the array such that left < pivot < right.

**Steps:**

1. Choose a pivot (middle, last, random).
2. Partition array so that left < pivot and right > pivot.
3. Recurse on left and right.

**Time Complexity:**

| Case | Complexity |
|---|---|
| Best | O(n log n) |
| Average | O(n log n) |
| Worst | O(n²) |

- Space: O(log n) average (recursive stack)

**Example Inputs:**

- Best: Random unsorted [4, 1, 3, 2, 5]
- Worst: [1, 2, 3, 4, 5] (if pivot = first/last element)

**Comparison Table**

| Algorithm | Best | Average | Worst | | Space |
|---|---|---|---|---|---|
| Bubble Sort | O(n) | O(n²) | O(n²) | | O(1) |
| Selection Sort | O(n²) | O(n²) | O(n²) | | O(1) |
| Insertion Sort | O(n) | O(n²) | O(n²) | | O(1) |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) | | O(n) |
| Quick Sort | O(n log n) | O(n log n) | O(n²) | | O(log n) |

**When to Use What?**

| Situation | Suggested Sort |
| --- | --- |
| Small input, mostly sorted | Insertion Sort |
| Guaranteed O(n log n) | Merge Sort |
| Average-case fast, in-place | Quick Sort |