**1. Selection Sort**

* **Example:** Sorting an array of integers: [29, 10, 14, 37, 13].
* **Explanation:** Selection sort repeatedly selects the smallest element from the unsorted part and swaps it with the first unsorted element.
* **Time Complexity:**
  + **Best, Average, Worst:** O(n²)
  + **Explanation:** For each element, it scans the entire unsorted portion to find the minimum, leading to n-1 comparisons for the first element, n-2 for the second, and so on.

**2. Bubble Sort**

* **Example:** Sorting a list of student scores: [78, 65, 90, 55, 70].
* **Explanation:** Bubble sort compares adjacent elements and swaps them if they are in the wrong order, "bubbling" larger elements to the end of the list.
* **Time Complexity:**
  + **Best:** O(n) (when the array is already sorted)
  + **Average, Worst:** O(n²)
  + **Explanation:** In the worst case, it compares every element with every other element.

**3. Merge Sort**

* **Example:** Sorting a list of product IDs: [5, 1, 3, 2, 4].
* **Explanation:** Merge sort divides the list into halves, sorts each half, and merges them back together in a sorted manner.
* **Time Complexity:**
  + **Best, Average, Worst:** O(n log n)
  + **Explanation:** The list is repeatedly divided in half (log n), and merging takes linear time (n).

**4. Quick Sort**

* **Example:** Sorting a list of sales figures: [34, 7, 23, 32, 5, 62].
* **Explanation:** Quick sort selects a 'pivot' and partitions the array into elements less than and greater than the pivot.
* **Time Complexity:**
  + **Best, Average:** O(n log n)
  + **Worst:** O(n²) (when the pivot is the smallest or largest element)
  + **Explanation:** Similar to merge sort, but the partitioning can lead to worst-case scenarios if the pivot is poorly chosen.

**5. Binary Search Tree (BST)**

* **Example:** Storing employee IDs in a BST: 50, 30, 70, 20, 40, 60, 80.
* **Explanation:** In a BST, each node has at most two children, where the left child is less than the parent and the right is greater.
* **Time Complexity:**
  + **Best, Average:** O(log n)
  + **Worst:** O(n) (when the tree is unbalanced)
  + **Explanation:** Searching involves traversing the height of the tree; a balanced tree will have a height of log n.

**6. Linked List**

* **Example:** Managing a playlist of songs: Song A -> Song B -> Song C.
* **Explanation:** A linked list consists of nodes where each node points to the next, allowing efficient insertion and deletion.
* **Time Complexity:**
  + **Access:** O(n) (to find an element)
  + **Insertion/Deletion:** O(1) (if the position is known)
  + **Explanation:** Access requires traversal from the head to the node, while insertion/deletion can be done if you have a reference to the node.

**7. ArrayList**

* **Example:** Storing a dynamic list of favorite movies: ["Inception", "Avatar", "Titanic"].
* **Explanation:** An ArrayList can dynamically resize to accommodate new elements.
* **Time Complexity:**
  + **Access:** O(1) (direct index access)
  + **Insertion/Deletion:** O(n) (in the worst case, if resizing is needed or if elements need to be shifted)
  + **Explanation:** Accessing an element is fast, but adding or removing elements may require shifting existing elements.

**How to Calculate Time Complexity**