Sorting algorithms are algorithms that arrange elements in a specific order, such as ascending or descending.

Bubble Sort is a simple sorting algorithm that repeatedly steps through a list of elements, compares adjacent elements, and swaps them if they're in the wrong order. The algorithm gets its name because smaller elements "bubble" to the top of the list while larger elements "sink" to the bottom.

Here's how Bubble Sort works:

Start at the beginning of the list. Compare the first two elements. If the first element is larger than the second, swap them. Move to the next pair of elements and repeat step 2. Continue this process until the end of the list is reached. Now, the largest element is at the end of the list. Repeat the process for all remaining elements except the last one. After each iteration, the last unsorted element is in the correct position. Repeat the process until the entire list is sorted. Bubble Sort has a time complexity of O(n^2) in the worst and average cases, where n is the number of elements to be sorted.

In [1]:
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):  # Number of passes
        for j in range(n - i - 1):  # Compare and swap adjacent elements
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Example usage
numbers = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(numbers)
print("Sorted array:", numbers)
```

```
Sorted array: [11, 12, 22, 25, 34, 64, 90]
```

Insertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time. It's like sorting a deck of playing cards by repeatedly inserting the next card into its proper position among the already sorted cards.

Here's how Insertion Sort works:

Start from the second element (index 1) and consider it as the "key." Compare the key with the previous elements. If the key is smaller, shift the previous elements to the right until a proper position is found for the key. Repeat this process for each element, moving from right to left. The final result is a sorted array. Insertion Sort has a time complexity of O(n^2) in the worst and average cases, where n is the number of elements to be sorted. However, it can be efficient for small input sizes or partially sorted lists.

In [2]:
```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# Example usage
numbers = [12, 11, 13, 5, 6]
insertion_sort(numbers)
print("Sorted array:", numbers)
```

```
Sorted array: [5, 6, 11, 12, 13]
```

Selection Sort is a simple sorting algorithm that works by repeatedly selecting the smallest (or largest, depending on the order) element from the unsorted part of the list and putting it at the beginning (or end) of the sorted part of the list.

Here's how Selection Sort works:

Find the minimum (or maximum) element from the unsorted part of the list. Swap the found minimum (or maximum) element with the first element of the unsorted part. Move the boundary between the sorted and unsorted parts one element to the right (or left). Repeat steps 1-3 until the entire list is sorted. Selection Sort has a time complexity of O(n^2) in the worst, best, and average cases, where n is the number of elements to be sorted. This makes it less efficient than some other sorting algorithms like Merge Sort or Quick Sort.

In [3]:
```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]

# Example usage
numbers = [64, 25, 12, 22, 11]
selection_sort(numbers)
print("Sorted array:", numbers)
```

```
Sorted array: [11, 12, 22, 25, 64]
```

In [ ]:
```
```