Sorting Algorithms - Inplace, Outplace, Stable/Unstable

# Inplace Sorting Algorithms

1. Bubble Sort - Stable  
2. Insertion Sort - Stable  
3. Selection Sort - Unstable  
4. Quick Sort - Unstable  
5. Heap Sort - Unstable  
6. Shell Sort - Unstable

# Outplace Sorting Algorithms

1. Merge Sort - Stable  
2. Radix Sort - Stable  
3. Counting Sort - Stable  
4. Bucket Sort - Stable

# Iterative Merge Sort (C++)

void merge(int arr[], int left, int mid, int right) {  
 int n1 = mid - left + 1;  
 int n2 = right - mid;  
  
 int L[n1], R[n2];  
  
 for (int i = 0; i < n1; i++)  
 L[i] = arr[left + i];  
 for (int j = 0; j < n2; j++)  
 R[j] = arr[mid + 1 + j];  
  
 int i = 0, j = 0, k = left;  
  
 while (i < n1 && j < n2) {  
 if (L[i] <= R[j]) {  
 arr[k] = L[i];  
 i++;  
 } else {  
 arr[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 arr[k] = L[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 arr[k] = R[j];  
 j++;  
 k++;  
 }  
}  
  
void mergeSortIterative(int arr[], int n) {  
 int current\_size;  
 int left\_start;  
  
 for (current\_size = 1; current\_size <= n - 1; current\_size = 2 \* current\_size) {  
 for (left\_start = 0; left\_start < n - 1; left\_start += 2 \* current\_size) {  
 int mid = min(left\_start + current\_size - 1, n - 1);  
 int right\_end = min(left\_start + 2 \* current\_size - 1, n - 1);  
  
 merge(arr, left\_start, mid, right\_end);  
 }  
 }  
}

# Merging Two Sorted Lists (C++)

vector<int> mergeTwoSortedLists(vector<int>& list1, vector<int>& list2) {  
 vector<int> mergedList;  
 int i = 0, j = 0;  
  
 while (i < list1.size() && j < list2.size()) {  
 if (list1[i] < list2[j]) {  
 mergedList.push\_back(list1[i]);  
 i++;  
 } else {  
 mergedList.push\_back(list2[j]);  
 j++;  
 }  
 }  
  
 while (i < list1.size()) {  
 mergedList.push\_back(list1[i]);  
 i++;  
 }  
  
 while (j < list2.size()) {  
 mergedList.push\_back(list2[j]);  
 j++;  
 }  
  
 return mergedList;  
}