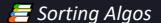




# **Bubble Sort**

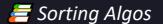
```
void swapValues(int &a, int &b) {
    int temp = a; // Store the value of 'a' in a temporary variable so that we don't lose it during the
    b = temp; // Assign the value of the temporary variable (which holds the original value of 'a') to
int findMinIndex(int arr[], int start, int end) {
    int minIndex = start; // Assume that the minimum value is at the start of the range
    for (int i = start + 1; i <= end; i++) { // Loop through the remaining elements in the range
        if (arr[i] < arr[minIndex]) { // If we find an element that is smaller than the current minimum</pre>
             minIndex = i; // update the index of the minimum value to the index of the new minimum value
    return minIndex; // Return the index of the minimum value
void bubbleUp(int arr[], int start, int end) {
    int minIndex = findMinIndex(arr, start, end); // Find the index of the minimum value in the range
if (minIndex != start) { // If the minimum value is not already at the start of the range (i.e., if
        swapValues(arr[start], arr[minIndex]); // swap it with the element at the start of the range
void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) { // Loop through all elements except the last one
        bubbleUp(arr, i, size - 1); // Move the smallest value in the remaining range to the start of
}
```





## Selection Sort

```
void swap(int &a, int &b)
    int temp = a; // Store the value of 'a' in a temporary variable so that we don't lose it during the
   a = b;
   b = temp;
int findMinIndex(int arr[], int startIndex, int endIndex)
    int minIndex = startIndex;
   for (int i = startIndex + 1; i <= endIndex; i++) // Loop through the remaining elements in the sub-
        if (arr[i] < arr[minIndex]) // If we find an element that is smaller than the current minimum
            minIndex = i; // update the index of the minimum value to the index of the new minimum value
    return minIndex; // Return the index of the minimum value
void selectionSort(int arrayToSort[], int sizeOfArray)
    for (int i = 0; i < sizeOfArray - 1; i++) // Loop through the array from the first element to the
        int minIndex = findMinIndex(arrayToSort, i, sizeOfArray - 1); // Find the index of the minimum
        swap(arrayToSort[i], arrayToSort[minIndex]);
```





#### **Insertion Sort**

```
void swap(int &a, int &b) {
    int temp = a; // Store the value of 'a' in a temporary variable to prevent losing the original value
a = b; // Assign the value of 'b' to 'a' to complete the swap
    b = temp; // Assign the value of the temporary variable to 'b' to complete the swap
void insertionSort(int arrayToSort[], int sizeOfArray) {
     for (int i = 1; i < sizeOfArray; i++) {</pre>
         int j=i; // Set the starting value of 'j' to 'i' to define the range to iterate over while (j>0) { // Iterate over the range until it's completely sorted
               if (arrayToSort[j] < arrayToSort[j - 1]) { // Check if the current element is smaller than</pre>
                    swap(arrayToSort[j], arrayToSort[j - 1]); // Swap the two elements
```



### Merge Sort

```
void merge(int arr[], int left[], int leftSize, int right[], int rightSize)
    while (1 < leftSize && r < rightSize) { // Loop through both arrays as Long as they both have elements remaining
        if (left[1] < right[r]) { //</pre>
            arr[i++] = left[l++];
        } else {
            arr[i++] = right[r++]; // append it to the result array and move the right counter forward
    while (1 < leftSize) { // if there are any elements remaining in the left array
        arr[i++] = left[l++]; // append them to the result array
    while (r < rightSize) { // if there are any elements remaining in the right array
        arr[i++] = right[r++]; // append them to the result array
void split(int arr[], int left[], int right[], int size)
    for (int i = 0; i < mid; i++) { // loop through the first half of the array
        left[i] = arr[i]; // copy the elements into the left array
    for (int i = mid; i < size; i++) { // loop through the second half of the array
        right[i - mid] = arr[i]; // copy the elements into the right array, adjusting the index to start at 0
void mergeSort(int arr[], int size)
    if (size < 2) { // base case: if the array has 0 or 1 elements, it is already sorted
    return; // exit the function</pre>
    int left[mid], right[size - mid]; // create two sub-arrays to hold the left and right halves of the original array
    split(arr, left, right, size); // split the original array into its two halves
    mergeSort(left, mid); // recursively sort the left half
    mergeSort(right, size - mid); // recursively sort the right half
    merge(arr, left, mid, right, size - mid); // merge the sorted halves back into the original array
```



https://github.com/Sharjeelbaig/Sortings

#### **Radix Sort**

```
int findMax(int arr[], int n)
    int maxVal = arr[0]; // Initialize maxVal as the first element of the array
    for (int i = 1; i < n; i++) // Loop through the rest of the array
        if (arr[i] > maxVal) // If the current element is greater than maxVal
            maxVal = arr[i]; // Update maxVal with the current element
    return maxVal; // Return the maximum value in the array
void countDigits(int arr[], int n, int exp, int count[])
        count[(arr[i] / exp) % 10]++; // Increment the count of the digit at the current place value
void updateCount(int count[])
    for (int i = 1; i < 10; i++) // Loop through the count array starting from index 1
        count[i] += count[i - 1]; // Add the count of the current digit to the count of the previous digit
void placeElement(int arr[], int n, int exp, int output[], int count[])
    for (int i = n - 1; i \ge 0; i--) // Loop through the array in reverse order
        output[count[(arr[i] / exp) \% 10] - 1] = arr[i]; // Place the current element in its correct position in the
        count[(arr[i] / exp) % 10]--; // Decrement the count of the digit at the current place value
void radixSort(int arr[], int n)
    int maxVal = findMax(arr, n); // Find the maximum value in the array
    for (int exp = 1; maxVal / exp > 0; exp *= 10) // Loop through the digits, starting with the least significant
        int output[n]; // Initialize an output array to store the sorted elements
        countDigits(arr, n, exp, count); // Count the number of elements with each digit at the current place value
        updateCount(count); // Update the count array to include the counts from previous digits
placeElement(arr, n, exp, output, count); // Place each element in its correct position in the output array
            arr[i] = output[i];
```