**1.**

2,1,10,6,3,8,7,13,20

Selection:

1. 1,2,10,6,3,8,7,13,20
2. 1,2,3,6,10,8,7,13,20
3. 1,2,3,6,7,8,10,13,20

Insertion:

1. 1,2,10,6,3,8,7,13,20
2. 1,2,6,10,3,8,7,13,20
3. 1,2,3,6,10,8,7,13,20
4. 1,2,3,6,7,10,8,13,20
5. 1,2,3,6,7,8,10,13,20

Bubble:

1. 1,2,10,6,3,8,7,13,20
2. 1,2,6,10,3,8,7,13,20
3. 1,2,6,3,10,8,7,13,20
4. 1,2,6,3,8,10,7,13,20
5. 1,2,6,3,8,7,10,13,20
6. 1,2,3,6,8,7,10,13,20
7. 1,2,3,6,7,8,10,13,20

Merge:

1. 2,1,10,6,3 | 8,7,13,20
2. 2,1,10 | 6,3 | 8,7 | 13,20
3. 2,1 | 10 | 6 | 3 | 8 | 7 | 13 | 20
4. 2 | 1 | 10 | 6 | 3 | 8 | 7 | 13 | 20
5. 1,2 | 10 | 3,6 | 7,8 | 13,20
6. 1,2,10 | 3,6 | 7,8 | 13,20
7. 1,2,3,6,10 | 7,8,13,20
8. 1,2,3,6,7,8,10,13,20

Quick:

1. 2,1,10,6,3,8,7,13,20
2. 2,1,10,6,3,8,7| 13 |20
3. 2,1,6,3 | 7 | 8,10 | 13 |20
4. 2,1 | 3 | 6 | 7 | 8 | 10 | 13 | 20
5. 1 | 2 |3 | 6 |7 | 8 | 10 | 13 | 20

**2.**

* 1,2,3,4,5,6,7,8,9,10

Selection:

1. 1,2,3,4,5,6,7,8,9,10

Insertion:

1. 1,2, 3,4,5,6,7,8,9,10
2. 1,2,3, 4,5,6,7,8,9,10
3. 1,2,3,4, 5,6,7,8,9,10
4. …

Bubble:

1. 1,2,3,4,5,6,7,8,9,10
2. …

Merge:

1. 1,2,3,4,5|6,7,8,9,10
2. 1,2,3|4,5|6,7,8|9,10
3. 1,2|3|4|5|6,7|8|9|10
4. 1|2|3|4|5|6|7|8|9|10
5. 1,2,3,4,5,6,7,8,9,10

Quick:

1. 1,2,3,4,5|6|7,8,9,10
2. 1,2,3|4,5|6|7,8|9,10
3. 1,2|3|4|5|6|7|8|9|10
4. 1,2,3,4,5,6,7,8,9,10

* 10,9,8,7,6,5,4,3,2,1

Selection:

* 1. 1,9,8,7,6,5,4,3,2,10
  2. 1,2,8,7,6,5,4,3,9,10
  3. 1,2,3,7,6,5,4,8,9,10
  4. 1,2,3,4,6,5,7,8,9,10
  5. 1,2,3,4,5,6,7,8,9,10

Insertion:

1. 9,10,8,7,6,5,4,3,2,1
2. 8,9,10,7,6,5,4,3,2,1
3. 7,8,9,10,6,5,4,3,2,1
4. 6,7,8,9,10,5,4,3,2,1
5. 5,6,7,8,9,10,4,3,2,1
6. 4,5,6,7,8,9,10,3,2,1
7. 3,4,5,6,7,8,9,10,2,1
8. 2,3,4,5,6,7,8,9,10,1
9. 1,2,3,4,5,6,7,8,9,10

Bubble:

1. 9,10,8,7,6,5,4,3,2,1
2. 9,8,10,7,6,5,4,3,2,1
3. 9,8,7,10,6,5,4,3,2,1
4. …
5. 9,8,7,6,5,4,3,2,1,10
6. …

Merge:

1. 10,9,8,7|6|5,4,3,2,1
2. 10,9|8,7|6|5,4|3,2,1
3. 10|9|8|7|6|5|4|3|2|1
4. 9,10|7,8|5,6|3,4|1,2
5. 7,8,9,10|5,6|1,2,3,4
6. 1,2,3,4,5,6,7,8,9,10

Quick:

1. 10,9,8,7,6,5,4,3,2|1
2. 10,9,8,7,6,5,4,3|2|1
3. …

* 9, 10, 1, 2, 3, 4, 5, 6, 7, 8

Selection:

1. 1,10,9,2,3,4,5,6,7,8
2. 1,2,9,10,3,4,5,6,7,8
3. 1,2,3,10,9,4,5,6,7,8
4. 1,2,3,4,9,10,5,6,7,8
5. ...

Insertion:

1. 9,10,1,2,3,4,5,6,7,8
2. 1,9,10,2,3,4,5,6,7,8
3. 1,2,9,10,3,4,5,6,7,8
4. 1,2,3,9,10,4,5,6,7,8
5. ...

Merge:

1. 9,10,1,2|3|4,5,6,7,8
2. 9,10|1,2|3|4,5|6,7,8
3. 9|10|1|2|3|4|5|6|7|8
4. 9,10|1,2|3,4|5,6|7,8
5. 1,2,3,4,5,6,7,8,9,10

**3.**

Hàm swap:

**void** Swap(**int** \*a, **int** \*b){

**int** temp = \*a;

    \*a = \*b;

    \*b = temp;

}

Selection Sort:

**void** selectionSort(**int** arr[], **int** length)

{

**int** i, j, min\_idx;

**for** (i = 0; i < length - 1; i++) {

        min\_idx = i;

**for** (j = i + 1; j < length; j++) {

**if** (arr[j] < arr[min\_idx])

                min\_idx = j;

        }

**if** (min\_idx != i)

            swap(&arr[min\_idx],&arr[i]);

    }

}

Insertion Sort:

**void** insertionSort(**int** arr[], **int** length)

{

**int** i, key, j;

**for** (i = 1; i < length; i++) {

        key = arr[i];

        j = i - 1;

**while** (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

Bubble Sort:

**void** bubbleSort(**int** arr[], **int** n){

**int** i, j;

**bool** swapped;

**for** (i = 0; i < n - 1; i++) {

        swapped = **false**;

**for** (j = 0; j < n - i - 1; j++) {

**if** (arr[j] > arr[j + 1]) {

                swap(&arr[j], &arr[j + 1]);

                swapped = **true**;

            }

        }

**if** (swapped == **false**)

**break**;

    }

}

Merge Sort (cre https://www.geeksforgeeks.org/):

**void** merge(**int** array[], **int** **const** left, **int** **const** mid,

**int** **const** right)

{

**int** **const** subArrayOne = mid - left + 1;

**int** **const** subArrayTwo = right - mid;

    // Create temp arrays

**auto** \*leftArray = **new** **int**[subArrayOne],

         \*rightArray = **new** **int**[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]

**for** (**auto** i = 0; i < subArrayOne; i++)

        leftArray[i] = array[left + i];

**for** (**auto** j = 0; j < subArrayTwo; j++)

        rightArray[j] = array[mid + 1 + j];

**auto** indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;

**int** indexOfMergedArray = left;

    // Merge the temp arrays back into array[left..right]

**while** (indexOfSubArrayOne < subArrayOne

           && indexOfSubArrayTwo < subArrayTwo) {

**if** (leftArray[indexOfSubArrayOne]

            <= rightArray[indexOfSubArrayTwo]) {

            array[indexOfMergedArray]

                = leftArray[indexOfSubArrayOne];

            indexOfSubArrayOne++;

        }

**else** {

            array[indexOfMergedArray]

                = rightArray[indexOfSubArrayTwo];

            indexOfSubArrayTwo++;

        }

        indexOfMergedArray++;

    }

    // Copy the remaining elements of

    // left[], if there are any

**while** (indexOfSubArrayOne < subArrayOne) {

        array[indexOfMergedArray]

            = leftArray[indexOfSubArrayOne];

        indexOfSubArrayOne++;

        indexOfMergedArray++;

    }

    // Copy the remaining elements of

    // right[], if there are any

**while** (indexOfSubArrayTwo < subArrayTwo) {

        array[indexOfMergedArray]

            = rightArray[indexOfSubArrayTwo];

        indexOfSubArrayTwo++;

        indexOfMergedArray++;

    }

**delete**[] leftArray;

**delete**[] rightArray;

}

Quick Sort(cre: https://www.geeksforgeeks.org/):

**int** partition(**int** arr[], **int** start, **int** end)

{

**int** pivot = arr[start];

**int** count = 0;

**for** (**int** i = start + 1; i <= end; i++) {

**if** (arr[i] <= pivot)

            count++;

    }

    // Giving pivot element its correct position

**int** pivotIndex = start + count;

    swap(arr[pivotIndex], arr[start]);

    // Sorting left and right parts of the pivot element

**int** i = start, j = end;

**while** (i < pivotIndex && j > pivotIndex) {

**while** (arr[i] <= pivot) {

            i++;

        }

**while** (arr[j] > pivot) {

            j--;

        }

**if** (i < pivotIndex && j > pivotIndex) {

            swap(arr[i++], arr[j--]);

        }

    }

**return** pivotIndex;

}

**void** quickSort(**int** arr[], **int** start, **int** end)

{

    // base case

**if** (start >= end)

**return**;

    // partitioning the array

**int** p = partition(arr, start, end);

    // Sorting the left part

    quickSort(arr, start, p - 1);

    // Sorting the right part

    quickSort(arr, p + 1, end);

}