1. Bubble Sort

```
const bubbleSort = arr => {
  for (let i = 0; i < arr.length - 1; i++) {
    let change = false;
    for (let j = 0; j < arr.length - (i + 1); j++) {
      if (arr[j] > arr[j + 1]) {
        change = true;
        arr[j] ^=arr[j+1]
        arr[j+1] ^=arr[j]
        arr[j] ^=arr[j+1]
      }
    },
    if (!change) break;
  }
  return arr;
};
```

2. Selection Sort

```
const selectionSort = arr=>{
  var minIdx, temp,
      len = arr.length;
  for(var i = 0; i < len; i++){
    minIdx = i;
    for(var  j = i+1; j<len; j++){
       if(arr[j]<arr[minIdx]){
          minIdx = j;
       }
    }
    temp = arr[i];
    arr[i] = arr[minIdx];
    arr[minIdx] = temp;
  }
  return arr;
}
```

3. Insertion Sort

```
const insertionSort = arr => {
    for (let i = 1; i < arr.length; i++) {
        let key = arr[i];
        let j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
    return arr;
};
```

4. Merge Sort

```
function mergeSort(arr){
   var len = arr.length;
   if(len <2)
      return arr;
   var mid = Math.floor(len/2),
       left = arr.slice(0,mid),
       right =arr.slice(mid);
   return merge(mergeSort(left),mergeSort(right));
}

function merge(left, right){

  var result = [],
      lLen = left.length,
      rLen = right.length,
      l = 0,
      r = 0;
  while(l < lLen && r < rLen){
     if(left[l] < right[r]){
       result.push(left[l++]);
     }
     else{
```

```
      result.push(right[r++]);
    }
  }
  return result.concat(left.slice(l)).concat(right.slice(r));
}
```

5. Quick Sort

```
const quickSort = arr => {

  if (arr.length < 2) return arr;

  const pivot = arr[Math.floor(Math.random() * arr.length)];

  let left = [];
  let equal = [];
  let right = [];

  for (let element of arr) {
    if (element > pivot) right.push(element);
    else if (element < pivot) left.push(element);
    else equal.push(element);
  }

  return quickSort(left)
    .concat(equal)
    .concat(quickSort(right));
};
```

6. Heap Sort

```
function heapify(input, i,arrayLength) {

  var left = 2 * i + 1;
  var right = 2 * i + 2;
  var largest = i;

  if (left < arrayLength && input[left] > input[largest]) {
      largest = left;
  }

  if (right < arrayLength && input[right] > input[largest]) {
      largest = right;
  }

  if (largest != i) {
      swap(input, i, largest);
      heapify(input, largest,arrayLength);
  }
}

function swap(input, index_A, index_B) {
  var temp = input[index_A];
  input[index_A] = input[index_B];
  input[index_B] = temp;
}

function heapSort(input) {
  arrayLength = input.length;
  for (var i = Math.floor(arrayLength / 2); i >= 0; i --) {
      heapify(input, i,arrayLength);
  }
  for (var i = input.length - 1; i > 0; i--) {
      swap(input, 0, i);
      heapify(input, 0, --arrayLength);
  }
  return input
}
```

7. Shell Sort

```
const shellSort = arr => {
    var increment = arr.length / 2;
    while (increment > 0) {
        for (i = increment; i < arr.length; i++) {
            var j = i-increment;
            var temp = arr[i];
```

```
            while (j >= 0 && arr[j] > temp) {
                arr[j+increment] = arr[j];
                j = j - increment;
            }
            arr[j+increment] = temp;
        }
        if (increment == 2) {
            increment = 1;
        } else {
            increment = parseInt(increment*5 / 11);
        }
    }
  return arr;
}
```

8. Bucket Sort

```
function insertionSort(array) {
  var length = array.length;

  for(var i = 1; i < length; i++) {
    var temp = array[i];
    for(var j = i - 1; j >= 0 && array[j] > temp; j--) {
      array[j+1] = array[j];
    }
    array[j+1] = temp;
  }

  return array;
}

function bucketSort(array, bucketSize) {
  if (array.length === 0) {
    return array;
  }

  var i,
      minValue = array[0],
      maxValue = array[0],
      bucketSize = bucketSize || 5;

  array.forEach(function (currentVal) {
        if (currentVal < minValue) {
                minValue = currentVal;
        } else if (currentVal > maxValue) {
                maxValue = currentVal;
        }
  })

  var bucketCount = Math.floor((maxValue - minValue) / bucketSize) + 1;
  var allBuckets = new Array(bucketCount);

  for (i = 0; i < allBuckets.length; i++) {
    allBuckets[i] = [];
  }

  array.forEach(function (currentVal) {
        allBuckets[Math.floor((currentVal - minValue) / bucketSize)].push(currentVal);
  });

  array.length = 0;

  allBuckets.forEach(function(bucket) {
        insertionSort(bucket);
        bucket.forEach(function (element) {
                array.push(element)
        });
  });

  return array;
}
```

9. Radix Sort

```
const radixSort = arr => {
  const maxNum = Math.max(...arr) * 10;
  let divisor = 10;
```

```
  while (divisor < maxNum) {
    let buckets = [...Array(10)].map(() => []);

    for (let num of arr) {
      buckets[Math.floor((num % divisor) / (divisor / 10))].push(num);
    }

    arr = [].concat.apply([], buckets);
    divisor *= 10;
  }
  return arr;
};
```

10. Counting Sort

```
let countingSort = (arr, min, max) => {
    let len = arr.length,  count = [];
    for (let i=min; i <= max; i++) {
        count[i] = 0;
    }
    for (let i = 0; i < len; i++) {
        count[arr[i]] += 1;
    }
    let j = 0;
    for (i = min; i <= max; i++) {
        while (count[i] > 0) {
            arr[j] = i;
            j++;
            count[i]--;
        }
    }
    return arr;
};
```