JAVA

DATE:16-12-2025
TOPIC: TYPES OF SORT'S
GROUP NAME: CODE CRAFTERS

GROUP MEMBER & ROLL NUMBERS:

M.VIDHU	- 2305A41063
B.AKSHITHA	- 2305A41090
P.SAI KUMAR	- 2305A41082
R.SUSWARAN	- 2305A41065
P.VRUSHANK	- 2305A41081

DIFFERENT TYPES OF SORTS IN ARRAYS

- **SELECTION SORT**
- **INSERTION SORT**
- **QUICK SORT**
- **COUNTING SORT**

1. SELECTION SORT:

- To implement the Selection Sort algorithm in a programming language, we need:

1. An array with values to sort.
2. An inner loop that goes through the array, finds the lowest value, and moves it to the front of the array. This loop must loop through one less value each time it runs.
3. An outer loop that controls how many times the inner loop must run. For an array with N values this outer loop must run n-1 times

CODES:

```
public class selectionsort {
  public static void main(String[] args) {
     int[] arr = {64, 25, 12, 22, 11};
     selectionSort(arr);
     System.out.println("Sorted array:");
     for (int num : arr) {
        System.out.print(num + " ");
     }
  }

  static void selectionSort(int[] arr) {
     int n = arr.length;

     for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
           if (arr[j] < arr[minIdx]) {
              minIdx = j;
           }
        }
        int temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
     }
  }

}
```

INSERTION SORT:

- To implement the Insertion Sort algorithm in a programming language, we need:

1. An array with values to sort.
2. An outer loop that picks a value to be sorted. For an array with nvalues, this outer loop skips the first value, and must run n−1 times.
3. An inner loop that goes through the sorted part of the array, to find where to insert the value. If the value to be sorted is at index i, the sorted part of the array starts at index 0 and ends at index I−1.

CODE;

```
        public class insertionsort {
  public static void main(String[] args) {
    int[] arr = {64, 25, 12, 22, 11};
    insertionSort(arr);
    System.out.println("Sorted array:");
    for (int num : arr) {
      System.out.print(num + " ");
    }
  }

  static void insertionSort(int[] arr) {
    int n = arr.length;

    for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;

      while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j--;
      }
      arr[j + 1] = key;
    }
  }

}
```

QUICK SORT:

1. To write a 'quickSort' method that splits the array into shorter and shorter sub-arrays we use recursion. This means that the 'quickSort' method must call itself with the new sub-arrays to the left and right of the pivot element. Read more about recursion here.

2. To implement the Quicksort algorithm in a programming language, we need:

3. An array with values to sort.
4. A quickSort method that calls itself (recursion) if the sub-array has a size larger than 1. A partition method that receives a sub-array, moves values around, swaps the pivot element into the sub-array and returns the index where the next split in sub-arrays happens.

<u>CODE:</u>

```
public class quicksort {
   public static void main(String[] args) {
      int[] arr = {64, 25, 12, 22, 11};
      quickSort(arr, 0, arr.length - 1);
      System.out.println("Sorted array:");
      for (int num : arr) {
         System.out.print(num + " ");
      }
   }

   static void quickSort(int[] arr, int low, int high) {
      if (low < high) {
         int pi = partition(arr, low, high);
         quickSort(arr, low, pi - 1);
         quickSort(arr, pi + 1, high);
      }
   }

   static int partition(int[] arr, int low, int high) {
      int pivot = arr[high];
      int i = (low - 1);
      for (int j = low; j < high; j++) {
         if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

}
```

COUNTING SORT:

- To implement the Counting Sort algorithm in a programming language, we need:

1. An array with values to sort.

2. A 'countingSort' method that receives an array of integers.

3. An array inside the method to keep count of the values.

4. A loop inside the method that counts and removes values, by incrementing elements in the counting array.

5. A loop inside the method that recreates the array by using the counting array, so that the elements appear in the right order.

6. One more thing: We need to find out what the highest value in the array is, so that the counting array can be created with the correct size. For example, if the highest value is 5, the counting array must be 6 elements in total, to be able count all possible non negative integers 0, 1, 2, 3, 4 and 5….

CODE:

```
public class countingsort {
    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        countingSort(arr);
        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    static void countingSort(int[] arr) {
        int n = arr.length;
        int max = findMax(arr);
        int[] count = new int[max + 1];
        int[] output = new int[n];

        // Count occurrences of each element
        for (int i = 0; i < n; i++) {
            count[arr[i]]++;
        }

        // Update count array to hold the position of elements
        for (int i = 1; i <= max; i++) {
            count[i] += count[i - 1];
        }

        // Build the output array
        for (int i = n - 1; i >= 0; i--) {
            output[count[arr[i]] - 1] = arr[i];
            count[arr[i]]--;
        }

        // Copy the sorted elements back to the original array
        System.arraycopy(output, 0, arr, 0, n);
    }

    static int findMax(int[] arr) {
        int max = arr[0];
        for (int num : arr) {
            if (num > max) {
                max = num;
            }
        }
        return max;
    }

}
```