# PRACTICAL TUTORIAL
# WORKSHEET N°3: SORTING ALGORITHM

**Exercise 1:** Gnome sort is a sorting algorithm. Here is the source code:

```
void gnomeSort(int *A, int n)
{   int i = 0;
    while (i < n) {
        if (i == 0) i++;
        if (A[i] >= A[i-1]) i++;
        else { swap(A, i, i-1);  // Swap of A[i] with A[i-1]
               i--; }
    }
}
```

1. Unroll this algorithm on array [4, 2, 5, 1].
2. Analyze best case complexity (array already sorted) and worst case (Reverse sorted array).
3. Gnome sort is inspired by two sorting algorithms seen in class. Which ones?

**Exercise 2:** Write selection sort and insertion sort algorithms recursively.

**Exercise 3:** Insertion sort can be improved by using binary search to find where to insert the item.

1. Propose an implementation of binary insertion sort.
2. Compare the complexity of the proposed implementation versus classical insertion sort in the worst case (in number of comparisons and permutations).

**Exercise 4:** The partition function is executed at each iteration of the quicksort. Give the array obtained after execution of this function on the following array depending on whether the chosen pivot is: (1) The last element of the array, (2) The first element of the array, (3) The median element.

| 50 | 47 | 92 | 78 | 76 | 7 | 36 | 59 | 30 | 52 | 43 |
|---|---|---|---|---|---|---|---|---|---|---|

**Exercise 5:** Cocktail sort is a kind of bidirectional bubble sort which is done in 3 steps:

1. Traverse the array from left to right comparing each element with the next and swap if the one on the right is lower.
2. Traverse the array from right to left comparing each element with the previous one and swap if the one on left is greater.
3. Repeat steps 1 and 2 with the sub-array without the first and last elements.

Write a recursive function that implements this sort method and analyze its complexity.