# TP 2 - Problem 13 - I

```
1  #include <algorithm>
2  #include <cassert>
3  #include <iostream>
4  #include <random>
5
6  // Sort the range [a[first], a[last]] using the quick sort algorithm.
7  void quick_sort_helper(int a[], int first, int last) {
8    // Recursion is complete when first == last.
9    if (first < last) {
10     int left_arrow{first};
11     int right_arrow{last};
12     // The pivot can be chosen arbitrarily from the set of values.
13     int const pivot_value{a[(first + last) / 2]};
14
15     // Do while the elements of the left and right sides can be
16     // exchanged...
17     do {
18       // The right arrow is moved left until a value less than or equal to
19       // the pivot is encountered.
```

# TP 2 - Problem 13 - II

```
20          while (pivot_value < a[right_arrow])
21            --right_arrow;
22          // The left arrow is moved right until a value greater than or equal
23          // to the pivot is encountered.
24          while (a[left_arrow] < pivot_value)
25            ++left_arrow;
26          // Swap the two elements.
27          if (left_arrow <= right_arrow) {
28            // Optimisation: do not swap if the indexes or the elements are
29            // equal.
30            if ((left_arrow < right_arrow) &&
31                (a[left_arrow] != a[right_arrow])) {
32              int tmp{a[left_arrow]};
33              a[left_arrow] = a[right_arrow];
34              a[right_arrow] = tmp;
35            }
36            // Adjust the arrows: if the two elements are equal to the
37            // pivot value, you enter in a forever loop.
38            --right_arrow;
39            ++left_arrow;
```

# TP 2 - Problem 13 - III

```
40          }
41        } while (left_arrow <= right_arrow);
42        // Divide and conquer: recursive call on the first and the second
43        // partitions.
44        quick_sort_helper(a, first, right_arrow);
45        quick_sort_helper(a, left_arrow, last);
46      }
47    }
48    void quick_sort(int a[], int n) { quick_sort_helper(a, 0, n - 1); }
49
50    int main() {
51      // A list of numbers to sort.
52      int a[]{1, 2, 2, 3, 3, 3, 4, 4, 4, 4};
53      int constexpr n_a{sizeof a / sizeof a[0]};
54
55      // A pseudo random number generator.
56      std::mt19937 prng;
57      for (int j{}; j < 100; ++j) {
58        // Shuffle the list of numbers using the random number generator.
59        std::shuffle(a, a + n_a, prng);
```

# TP 2 - Problem 13 - IV

```
60      // Display the original list.
61      std::cout << "Input\n";
62      for (int const &element : a)
63        std::cout << element << ' ';
64      std::cout << '\n';
65      quick_sort(a, n_a);
66      // Display the sorted list.
67      std::cout << "Output\n";
68      for (int const &element : a)
69        std::cout << element << ' ';
70      std::cout << '\n';
71      // Check if the list is sorted.
72      for (int i{1}; i < n_a; ++i)
73        assert(a[i - 1] <= a[i]);
74    }
75    return 0;
76  }
```

# TP 2 - Problem 13 - V

Possible output:

```
Input
2 4 1 3 3 4 4 2 3 4
Output
1 2 2 3 3 3 4 4 4 4
Input
3 3 4 4 4 1 2 4 3 2
Output
1 2 2 3 3 3 4 4 4 4
...
Input
2 4 3 4 3 3 2 1 4 4
Output
1 2 2 3 3 3 4 4 4 4
```