Introduction Algorithms and Data Structures

This challenge consists of 4 parts:

1. Data parsing
2. Linear search
3. Sorting
4. Sorting research

Part 0 (data parsing):

In this part you must correctly parse input data from standard input (`stdin`). The input data is always given in 3 lines:

1. A single value N, that represents the size of the list on line 2
2. List of N numbers, space separated
3. A single value K

Example:

```
10
2 4 6 7 3 4 5 6 3 6
2
```

Create a separate module in the given project that correctly parses these 3 lines. Hint: use `scanf` to read data from `stdin`. More information about `scanf`: type `man scanf` on your Linux terminal.

Please note: make sure all your code complies with the semester 3 coding guidelines! These can be found in the GIT toolbox.

Part 1 (linear search):

Implement the following algorithm:
Given a list of **N** numbers a_1, a_2, a_3........a_n, find the smallest number from the list that is repeated in the list exactly **K** number of times.

Input Format

Input will be read from the standard input as explained in part 0.

Check out code-example next to this document and understand how it works.

Build targets:
make → builds executable main
make test → builds and runs unit tests
You can run the executable e.g. like this:

```
./main 1 < ../testdata/inputfile
```

where file `input_file` *contains the data as described above.*

Output Format

Smallest integer value that is repeated exactly **K** number of times

Constraints

- 0 < N <= 100000
- 0 <= K <= 100000
- 0 <= a_i <= 100000

NOTE

There will be at least one variable which is repeated **K** times.

Example

Input:
10
2 4 6 7 3 4 5 6 3 6
2

Output:
3

Explanation:
The smallest number of the list 2 4 6 7 3 4 5 6 3 6
that appears exactly 2 times in the list is number 3.

Challenge

Implement the function `FindSmallestNumberThatIsRepeatedKTimes` in `challenge.c`.

Tip:

You can test your code by creating unit tests (see test/find_test.c) and/or the following input files:

in1_1 should give result 1
in1_2 should give result 3
in1_3 should give result 1
in1_4 should give result 1
in1_5 should give result 98297
in1_6 should give result 3

Part 2 (sorting):

Implement the following algorithm:

Given a list of **N** numbers a_1, a_2, a_3........a_n, find the difference between the maximum and minimum sum of **K** elements of the list.

Input Format

Input will be read from the standard input as explained in part 0.

Output Format

Difference between the maximum sum of **K** elements and minimum sum of **K** elements in the given list.

Constraints

- $0 < N < 1000$
- $1 <= K < 999$
- $0 <= a_i < 10000$

Example

Input:
10
2 4 6 7 2 4 5 6 3 6
4

Output:
14

Explanation:
The maximum sum of 4 elements is 25.
The minimum sum of 4 elements is 11.
The difference is 14.

Challenge

Implement the method
`ComputeDifferenceBetweenMaxAndMinSumOfKElements_0` in `challenge.c`.

You can run the executable of assignment 2 by:

```
./main 2 < ../testdata/inputfile
```

Tip:

You can test your code by using unit tests and/or the following input files:
in2_1 should give result 17627
in2_2 should give result 7770
in2_3 should give result 19210
in2_4 should give result 999

Part 3 (sorting research):

If you haven't used sorting in assignment 2 yet, re-implement assignment 2 by using sorting.

Extend assignment 2 in the following way:
Next to the sorting algorithm used in assignment 2 implement 2 other sorting algorithms. This means, the functionality of assignment 2 will be run 3 times with 3 different algorithms.
You should use algorithms with at least 2 different big O values.

To study different sorting algorithms, use the following links or any other reliable source:
https://en.wikipedia.org/wiki/Sorting_algorithm
https://betterexplained.com/articles/sorting-algorithms/

Run all the algorithms with these sets of data:

1. N = 10
2. N = 100
3. N = 10000

Document describing used algorithms, motivation of their choice and which sources were used for study.

Further, the document will contain comparisons of the 3 chosen algorithms for N=10, N=100 and N = 10000 and conclusion.

Comparison should show:

- Time of execution *(you can call the provided getRealTime function before and after calling your sort function and compute the difference)*
- Number of swap operations
- Number of comparisons

Challenge

Implement the sorting algorithms using the following methods in `challenge.c`:

- `ComputeDifferenceBetweenMaxAndMinSumOfKElements_1`
- `ComputeDifferenceBetweenMaxAndMinSumOfKElements_2`
- `ComputeDifferenceBetweenMaxAndMinSumOfKElements_3`

You can run the executable of challenge 3 by:

```
./main 31 < ../testdata/inputfile

./main 32 < ../testdata/inputfile

./main 33 < ../testdata/inputfile
```

Adjust main.c to your likings as needed for performing the timed measurements. We look forward to smart solutions to automate your testing.

Demonstrate

The requested research document and the following code-files:

- challenge.c
- challenge_test.c (the use of unit tests is strongly encouraged)
- main.c