All code should be properly documented, commented and have proper unit testing implemented in main(). To facilitate things you should incorporate the README file for each task as a comment in the beginning of the corresponding file.

***OBS! When uploading source code do it as text files (.c or .java) - i.e. not screen shots or similar - we should be able to compile and run your code from what you have uploaded to Canvas.***

Assume that the data stored in each element is a character or integer (you may choose whichever you think is more convenient)

You are not allowed to use Java library implementations of the data structures (queues, lists etc.) The idea is that you should learn to implement the internals yourself!

**Programming assignments for grade E**:

Basis for assignments 1 and 2: Implement insertionsort. Augment the sorting process so that all the content of the array that is being sorted is printed after each inner loop iteration. Write a unit test in main() which allows the user to define the size of the input (N) and then input (N) integers from stdin which is to be sorted.

1. Augment the above implementation so that it prints the number of swaps performed when sorting the array.
2. Add a method which counts the number of inversions in the input array and prints a list of all inversions on the format [i,a[i]], [j, a[j]] where i and j are indices and a[i], a[j] are the values of the elements. Call the method from main() before the array is sorted. Calculate the time complexity for the algorithm.
3. Implement a function in C which takes an array of integers (both positive and negative) and orders the elements in the array so that all negative elements come before the positive. You are not allowed to sort the array (i.e. by any of the sorting methods) - only collect all negative values first. The algorithm should only use O(1) extra memory (i.e. be in-place [Wikipedia: In-place algorithm (Länkar till en externa sida.)Länkar till en externa sida.](https://en.wikipedia.org/wiki/In-place_algorithm))
4. Compare the execution times for sorting large arrays of integers with insertionsort, merge sort and quicksort. When should one select one method over the others? Upload code, tests and a graphs depicting the execution times as a function of input (what parameters in the input could be relevant?). (you need to test for a range of input sizes)
5. Experiment with the cut-off to insertionsort in mergesort. How is the execution time affected by different values for the cut-off? A suitable range for cut-off values to test with could be [0-30]. Upload code, tests and a graphs.
6. Compare the execution times of quicksort where the first element in each sub-array is selected as partitioning element to that of quicksort with median-of-three partitioning
7. Augment the test code from assignment 1&2 so that the array is sorted in descending order instead of ascending order (you may add O(N) operations) *Clarification: You should not change (not alter/modify any code in) the sorting method, nor should you sort the array an extra time. You may traverse the array once before sorting and once after sorting. During these traversals you may not move (re-order) any elements. (Hint: you need not and should not use any extra memory)*

Note: your investigations to determine execution times should be good enough to be significant (i.e one should be able to draw conclusions from them that are valid)

Upload code, tests, explanatory texts and for the time measurements graphs showing the results

**Questions to answer in the seminar**:

Show means that you should be able to project the code/files you have uploaded in Canvas from Canvas from your own computer or the teachers. I.e. you cannot use another file than the one you have uploaded in Canvas. You should be able to explain your code and your design choices in detail as well as how you have come up with the answers to the questions.

1. Explain the code from assignment 1, use the input [1, 2, 5, 3, 4, 0]
2. Explain the code from assignment 2 and how you calculate its time complexity, use the input [1, 2, 5, 3, 4, 0]
3. Explain the code from assignment 3 (separating negatives from positives) and how you calculate its time complexity
4. Explain the code from assignment 4 and how you performed the time measurements, why your results are valid and show the results
5. Explain the code from assignment 5 and how you performed the time measurements, why your results are valid and show the results
6. Explain the code from assignment 6 and how you performed the time measurements, why your results are valid and show the results
7. Explain how yo solved the problem

[4, 3, 2, 1]

[index: 0, value: 4], [index: 1, value: 3]

[index: 1, value: 4], [index: 2, value: 2]

[index: 2, value: 4], [index: 3, value: 1]

[index: 0, value: 3], [index: 1, value: 2]

[index: 0, value: 3], [index: 1, value: 3]

[index: 0, value: 4], [index: 1, value: 3]