# WRITTEN ASSIGNMENTS

The written tasks are related (at least tangentially) to tasks of the programming assignment.

And may even require you to do some more programming. To get points for each task, you

need to submit by the deadline and be prepared to discuss your answers at the exercise

session following the deadline.

Upload your answers as a single pdf to Moodle. The length of your submission should not

exceed 5 pages in total. Each task is worth 0.5 points for a total of 2.5 points.

## Task 1

What did you do and why for the programming tasks 1-3? What did you find / What results did you get? The answer should be brief and to the point, a total of ~1 page at most.

1. Programming task 1:
   1. Programming task 1 took the longest form this set of exercises (except task 3). The actual get and set functions were easy to think out theoretically. However, creating the main function and make file took a long time to get right as I had to learn a lot of functions from the standard library such as fstream, iostream, chrono, ect and bit functions to actually implement the functions.
2. Programming task 2:
   1. After programming task 1, programming task 2 was quite easy. Unfortunately, I accidently used \_\_builtin\_popcountl instead of \_\_builtin\_popcountll in my initial code. This took a few hours of different de-bugging to figure out as it only caused an error for numbers above 64.
3. Programming task 3:
   1. Unfortunately, I was not able to Finnish task 3, I’ve spent a few hours trying different solutions however as of now, I haven’t been able to do it. If I have time in the next week, I would like to give it another shot.

## Task 2

What methods did you use for timing total program execution and parts of the execution?

Can you find alternatives? What are some good and bad sides of different approaches to performance timing?

1. In my code I used chrono::system\_clock paired with chrono::duration to find the runtime. chrono::system\_clock simply looks at the system clock which shows the current date/time, then duration compares two system system\_clock type objects to give the difference in seconds. An alternative I found is std::chrono::steady\_clock, it works simmilarily to chrono::system\_clock however acts like a stopwatch. Finally, chrono::high\_resolution\_clock works as a type alias for either steady\_clock or system\_clock. It works similarly to the two however has the highest accuracy.

## Task 3

What effect does sorting have on performance? Why?

For task 1, set and get:

File A3 Unsorted:

set\_up time: 1.8537e-05s

querry time: 0.00325857s

file A3 Unsorted:

set\_up time: 6.4601e-05s

querry time: 0.00224766s

for task 2, sum:

File A3 Unsorted:

set\_up time: 3.4668e-05s

querry time: 0.00271412s

file A3 Unsorted:

set\_up time: 4.4721e-05s

querry time: 0.00250762s

I found that sorting the input had about a 30% average time cut for the get function while provided almost now time save for the sum function. Although I’m not 100% sure, I think this is because having the array be sorted allows for less movement in memory when setting and getting an item in the array.

## Task 4

Look around on the web and find a way to measure the peak memory usage of each of your

programs, and the sizes of the data structures used. Explain the method you used and

where you found it online (if you found it there). Record the peak memory usage of your

programs and data structures. Think about variation between programs and runs with

different data, if any.

Ran out of time / couldn’t get it to work

## Task 5

Any ideas on efficient (better than O(logn)) implementations for the location queries of

programming task 3? Why is location a more complicated problem than sum?

Didn’t finish task 3.