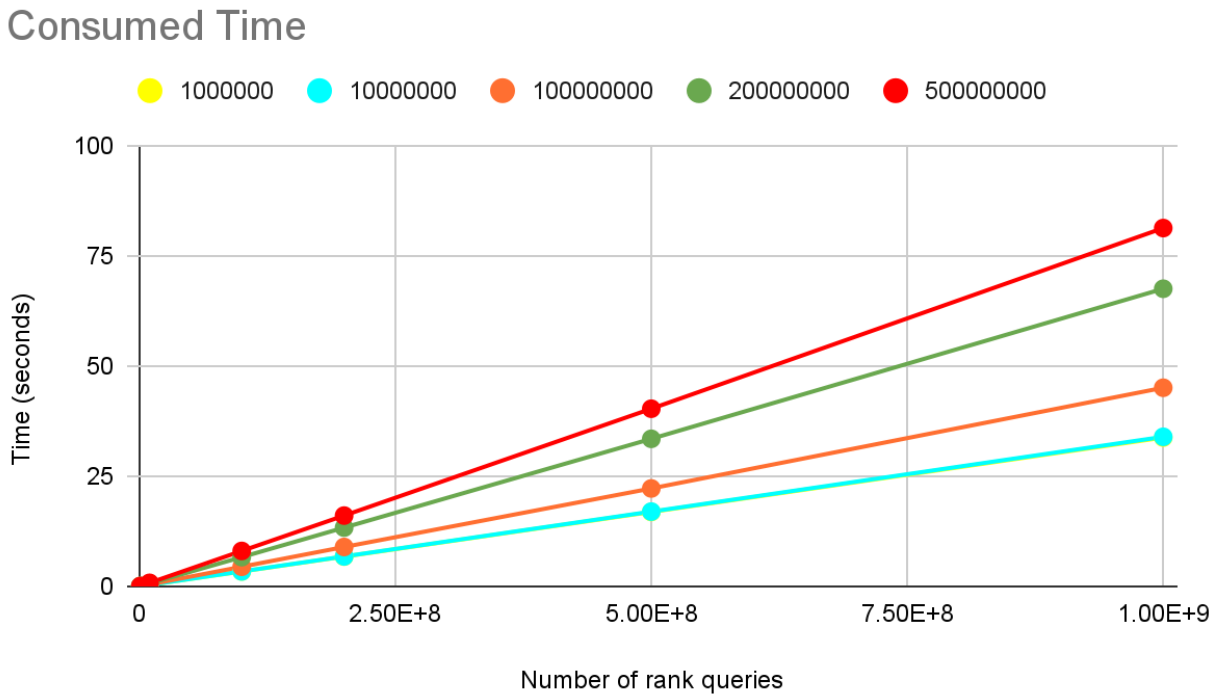
Github repository: https://github.com/PJeBeK/genomics-hw1

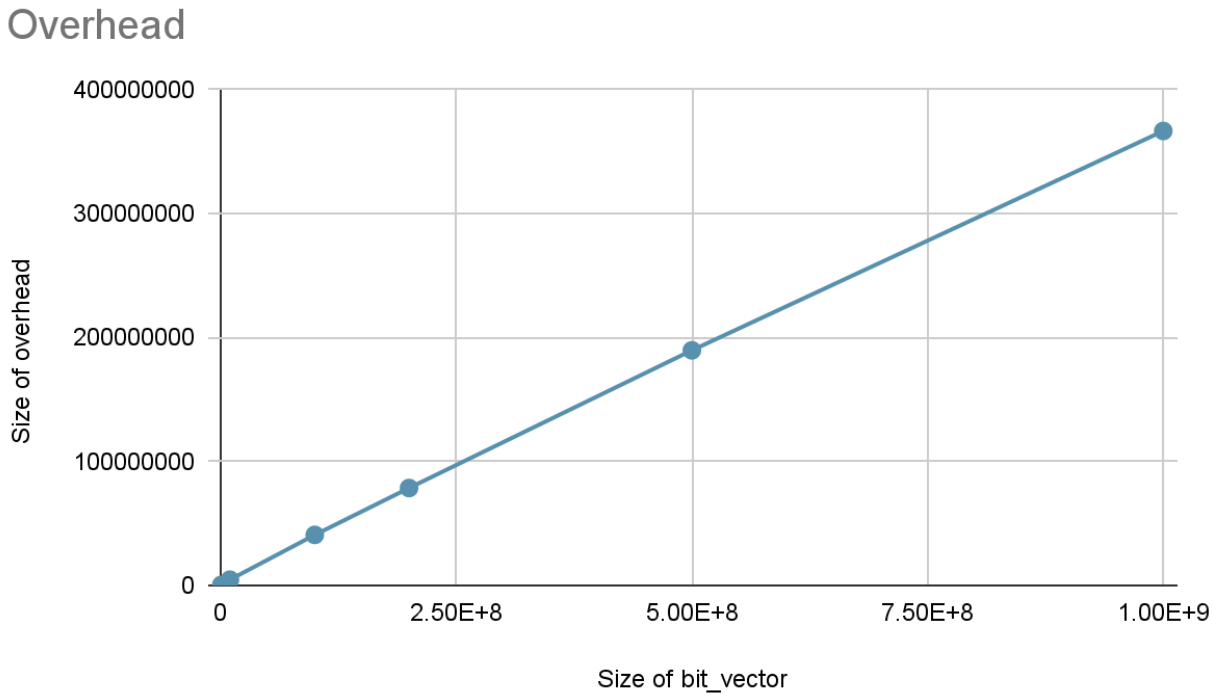
1. bit-vector rank

I have a *rank_support* structure which takes a bit_vector and answers to query *rank1(i)*. In this structure, we have an init*()* function which fills the *rs* and *rb* bit_vectors. First it split the input into some superblocks and then split each superblock into some smaller blocks. Then, for each superblock, *rs* will keep the number of 1s before this superblock, and for each block, *rb* will keep the number of 1s before this block inside its superblock. At the end, for answering *rank1(i)* queries, we just need to find the superblock and the block of index i and sum their *rs* and *rb* and add the number of 1s in its block up to i.

The most difficult part of this problem was using bit_vector. I did not find any well-written documentation for using sdsl::bit_vector and it was really hard to work its instances.



Each line shows the run time of my code on different N. For example the orange line shows that when N=10^8, how much time is consumed for different numbers of queries.

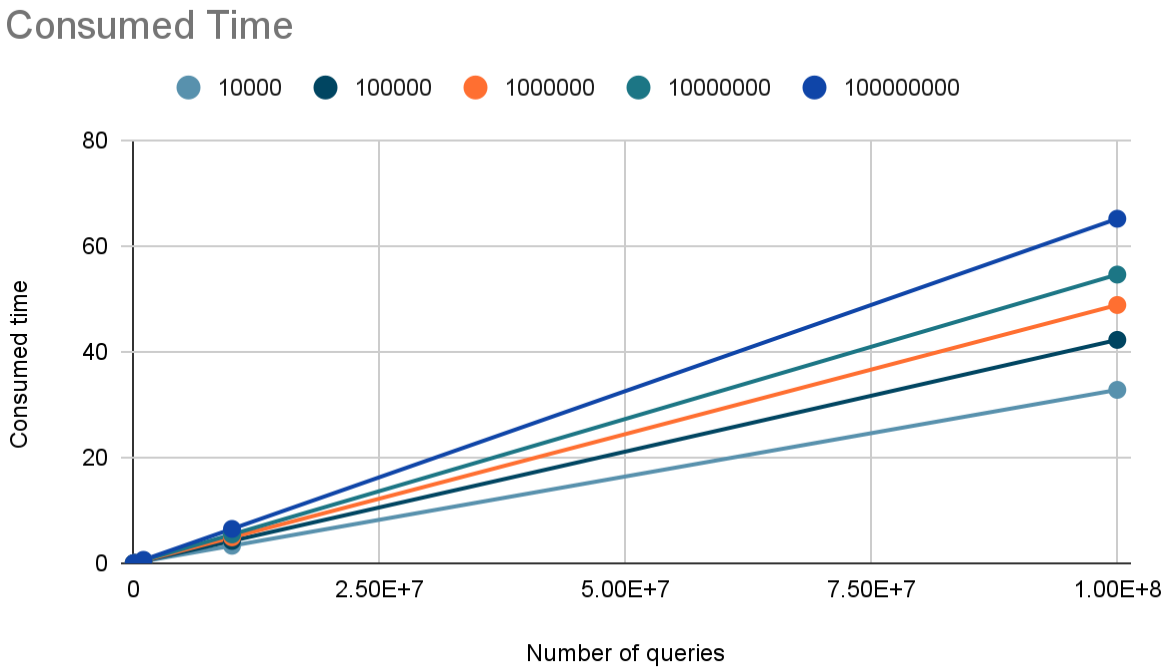


This plot shows that the extra space we use for *rs* and *rb* is less than the size of input and it is matching the expected theoretical bounds

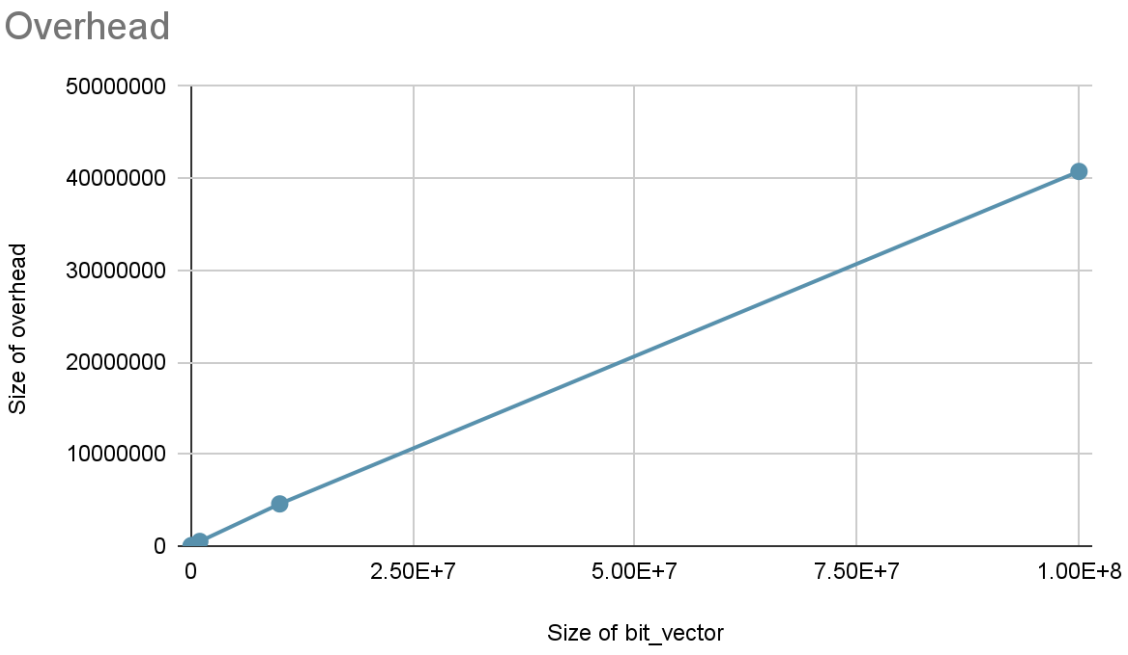
2. bit-vector select

For this part, we use the *rank_support* code from the previous section to construct our *select_support* structure. After we create a *rank_support* for the given bit_vector, for each query of *select1(i)*, we use binary search to find the first index which its rank is equal to *i*.

This task was not hard. Most of this task was implemented in the previous tasks and I just needed to add a binary search which is a few lines.



Each line shows the run time of my code on different N. For example the orange line shows that when N=10^6, how much time is consumed for different numbers of queries.



This plot shows that the extra space we use for *rs* and *rb* is less than the size of input and it is matching the expected theoretical bounds

3. Implementing a sparse array using your bitvector rank and select

For this part, I use *rank_support* in my *sparse_array* to answer queries. In *README* file, you can see the documentation of my code.
(sorry I don't have enough time to make the plots)