# Project 4: Suffix arrays and BW-match

## Status of work

We have implemented the binary search and the burrows wheeler based search algorithms. Everything is working as expected.

## Description of implementation

The programs are fragmented into two parts.

I. The first part preprocesses the input (genome). For binary search, this implies creating the suffix array which is done in a naive manner with pythons built-in sort-function. For Burrows wheeler based search, preprocessing implies calculating the suffix array (shared code with binary search), computing the c-table, bwt-string and the o-table. When all preprocessing steps have been completed, the collection of data structures used in each program which resides in a python object, is serialized by the built-in python module ‘pickle’ and saved to disk.

II. The second parts of the programs, reads the serialized objects from disk and rebuilds python objects that contain all data structures and functions needed to perform searching for a read that can be specified without recalculating the preprocessed data.

In order to save on memory usage when computing the suffix array, we tried to implement a pointer in python. Instead of creating a list of all the suffix-strings, we created a list, containing memoryview objects to the part of the genome that the suffix consists of. This greatly reduces the relative memory usage for large inputs, as the maximum size of a suffix is now the size of a memoryview object, instead of a potentially long string. This implementation detail is important, because our 8GB machine would run out of memory in a matter of minutes without it.

## Insight during implementation

One should not make a function $L(\alpha w)$ and evaluate that recursively. Because then the search is done one can save computational time by saving the last result and using that to evaluate the new $L(\beta\alpha w)$.

Python when using reverse on a list does not actually reverse the list but gives an iterator that runs in reverse.

## How to run the program

**search_bs.py**

```
python3 search_bs.py -p data/seqs.fasta && \
python3 search_bs.py -i data/reads.fastq
```

**search_bw.py**
```
python3 search_bw.py -p data/seqs.fasta && \
python3 search_bw.py -i data/reads.fastq
```

**Common for both programs:**

- The -p option tells the program that it should preprocess the sequences in the supplied .fasta-file, and save the data to disk. This data is saved in `preprocessed_sequences_bs.pickle` and `preprocessed_sequences_bw.pickle` respectively.
- The -i option tells the program that it should import earlier preprocessed sequences and map the reads from the supplied .fastq-file.

## Description of the machine

The running time analysis was done on a laptop running mac os 10.12, with a fifth generation intel-i5 (two physical cores), 2.40GHz CPU with 8 GB 1600 MHz DDR3 RAM.

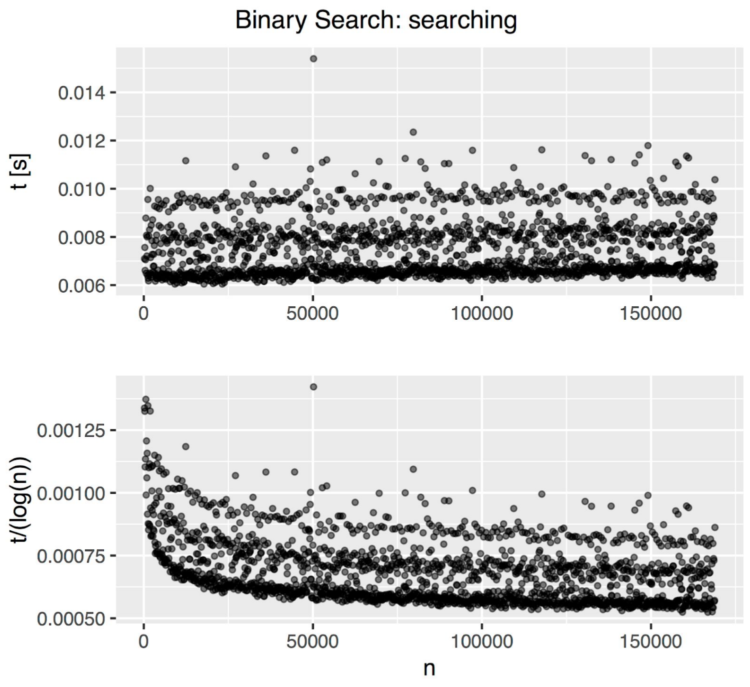
## Correctness of program

We have tested that the program works with “mississippi” and checked against the output given in the exercise.
We also used a set of strings to test edge cases.

## Description of tests

We tested the average case by using a real life representative sequence: Enterobacteria phage T4, complete genome from NCBI. The genome has a length of 168K basepairs.
For both programs, search_bs.py and search_bw.py, we tested the search time by searching for a ‘read’ which is an incrementally larger part of the whole genome. Such that when n = 1, the read is the first character of the genome, and when n = 168K, the read is equal to the complete genome.

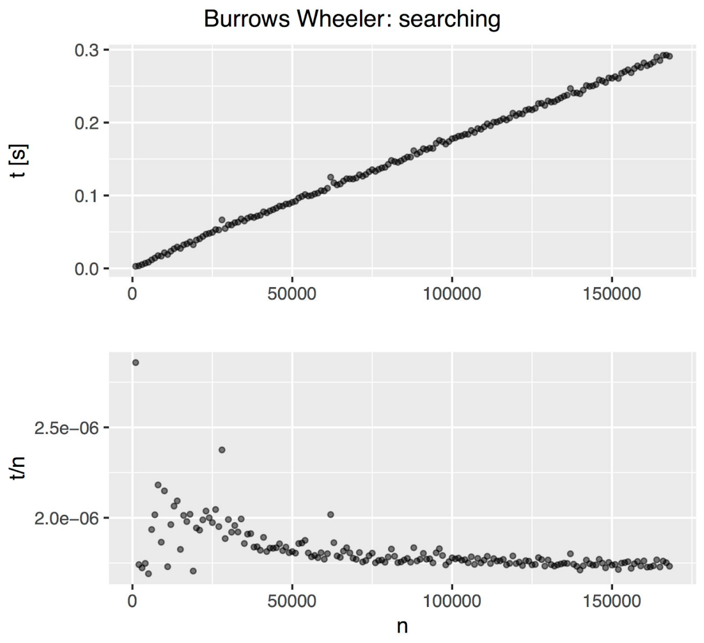
### Binary search



Searching for read with size n.
Enterobacteria phage T4 genome.

Binary search has a theoretical running time of *O(n log(|genome|))*, where n is the size of the read. From our test, i looks like our algorithm could be performing better than that. This might be because the genome used is not a worst case input.

**Burrows wheeler transform based search**



Searching for read with size n.
Enterobacteria phage T4 genome.

The theoretical search time for BW based search is *o(n)*, where *n* is the size of the read. From our test, it looks like it could be performing better than that. This could be because the genome used is not a worst case input.

## Conclusion

We have successfully implemented binary search and burrows wheeler based search as command line programs in python. Further analysis should include a discussion of whether the test-inputs are worst case or not.