Project 3: Genome Scale algorithms

Maria, Mateo and Jacob

2/20/2018

Instruction of the project: https://github.com/mailund/gsa-exercises/tree/master/Project04

Project 4 - Suffix arrays and BW-match

Program description

The programs are implemented in Python (2.7) and we had no unsolved issues implementing the algorithms. Execution of the programs on command line takes the form:

```
python search-bs.py [text-filename] [pattern]
python search-bw.py [text-filename] [pattern]
```

The program outputs positions where the second parameter (pattern) occurs in the text file. The positions are sorted by index of occurrence and if the pattern does not exist in the text *None* is returned.

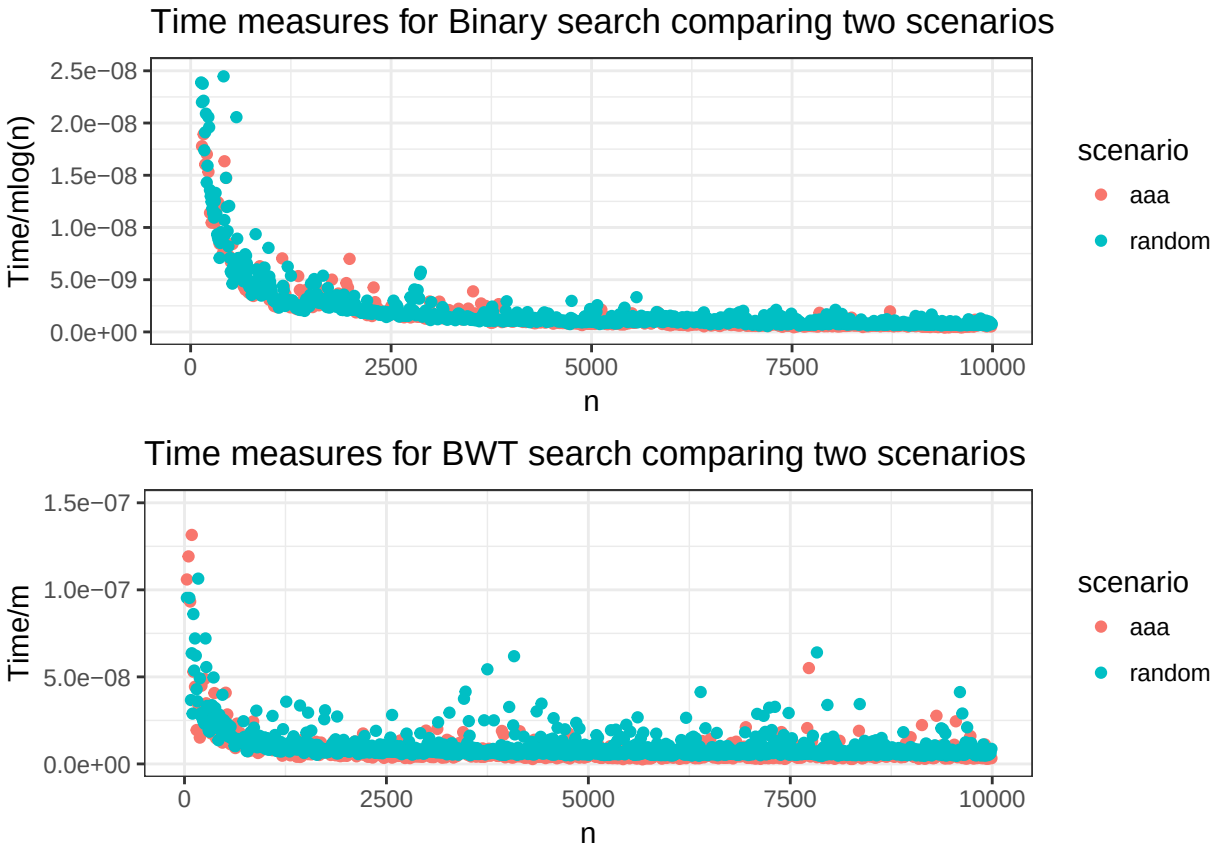
Problems

We did not have any problems regarding implementation except Radix sort. Since sorting suffixes using radix sort requires creating a 'bucket' for each letter in alphabet and taking ASCII values of letters we could end up with potential problems. Each letter in character corresponds to an integer value called ASCII value but special characters could not be found there, which can cause problems when trying to sort them using radix sort.

Evaluation

We measured time for both, search using Burrows-Wheeler transformation and binary search and following plots are showing the results. Two scenarios were tested:

1. Patterns matching the text.
2. Random text and random pattern.



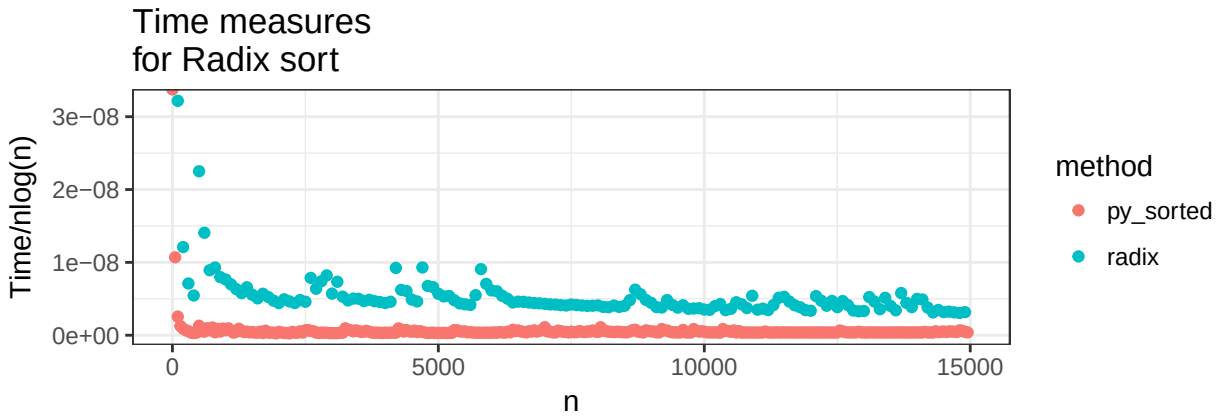
It is possible to observe that both implementations follow the expected time complexity.

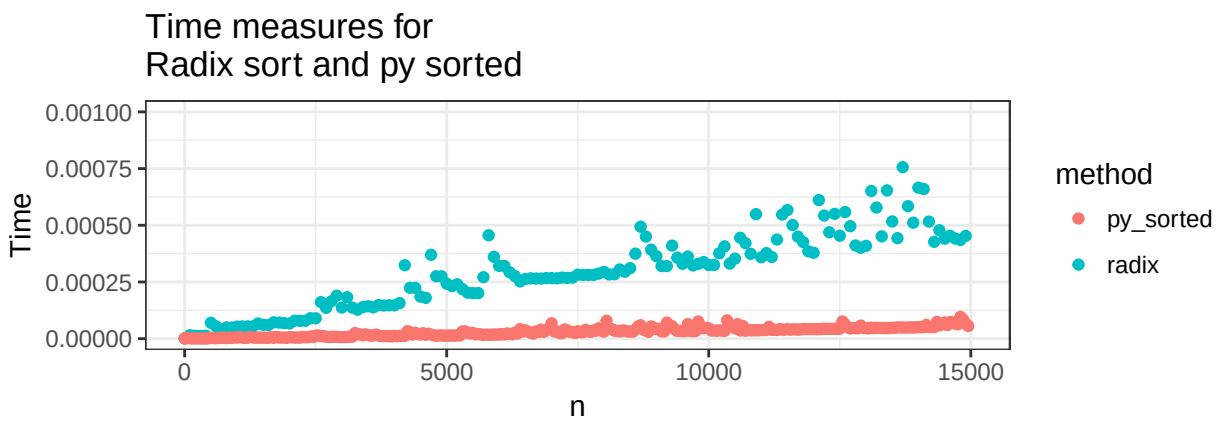
Test of correctness

We used the test data provided in folder test_data to test the correctness of the program. The outputs are the same as given by previous projects.

Time complexity of sorting (python sort and radix sort)

Implementation of radix sort uses bucket for each character in alphabet and takes ASCII values in order to arrange them. Using worst case scenario, text which contains only a's and patter which contains also all a's radix sort does not perform really good compared to built in python function (sorted()). Measured time in practice with worst case scenario showed that time complexity is nlog(n). On the other hand, built in function uses Timsort which is exceptionally adaptive merge sort, which is really fast in practice but in theory the same as merge sort. Time complexity of algorithm is nLog(n) (https://www.quora.com/What-is-the-time-complexity-of-the-Python-built-in-sorted-function).


Time measures
for Radix sort
Time/nlog(n)
3e−08
2e−08
1e−08
0e+00
0
5000
10000
15000
n
method
py_sorted
radix


Time measures for
Radix sort and py sorted
Time
0.00100
0.00075
0.00050
0.00025
0.00000
0
5000
10000
15000
n
method
py_sorted
radix