Comparing the Performance of Hybrid-1 & Fischer-Heun RMQ Structures

Problem definition:

In this project we compare the performance of my implementation of Hybrid-1 RMQ structure with Fischer-Heun RMQ structure. The criteria for comparing the performance of the above data-structures is the average time taken for Pre-processing and Querying both structure on 3 randomly generated datasets of size 128M, 256M and 512M values. Smaller time values represent a better performance and vice-versa.

Algorithm:

The Fischer-Heun RMQ algorithm ensures a Pre-processing time complexity of O(n) and a Querying time complexity of O (1). It is able to achieve the above performance by implementing the below steps in an efficient manner. Start by partitioning the array A into blocks $B_1, \ldots, B_{n/s}$ of sizes:=log n/4. Define two arrays A′ and B of size n/s=4n/log n, where A′[i] stores the minimum of block B_i, and B[i] stores the position of this minimum in A. Now A′ is pre-processed using the algorithm from Sect. 2.1, occupying O ((4n/log n) * log (4n/log n)) = O(n) space. Then precompute the answers to all possible queries on arrays of sizes and store the results in a table P. According to Lemma 1, this table occupies O((4^((log n)/4))(log n/4)^1/2)=O(n) space. Finally, compute the type of each block in A and store these values in array T [1,4n/log n]. As this is not as obvious as in Sect. 2.2, it is explained in detail in the following subsection. A query RMQ (i, j) is now answered exactly as explained in the last paragraph of Sect. 2.2, namely by comparing at most three minima, depending on the blocks where i and j occur. Again, the time for answering a query is constant, leading to the 〈O (n), O (1)〉 time bounds stated before.

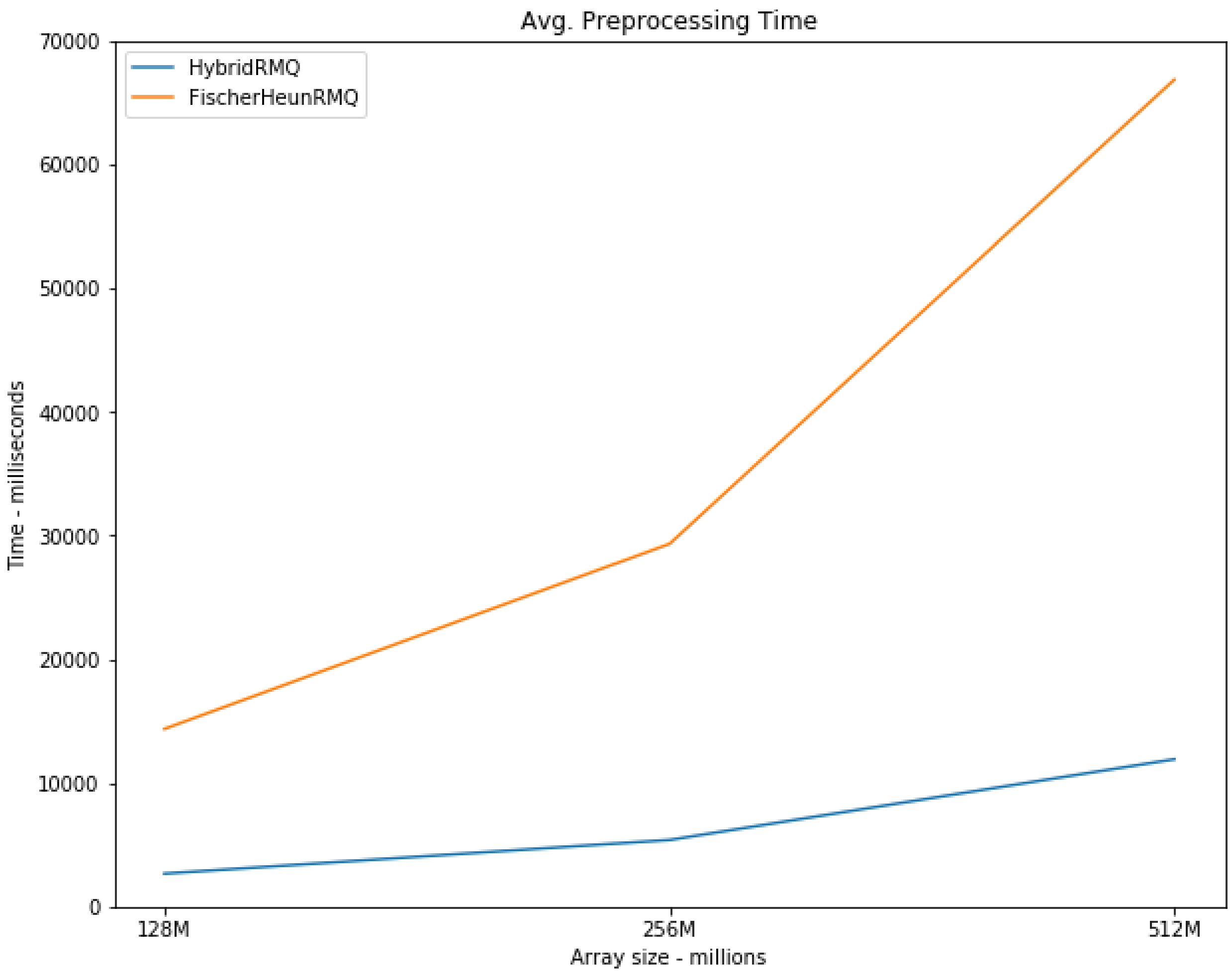
Problem Setup:

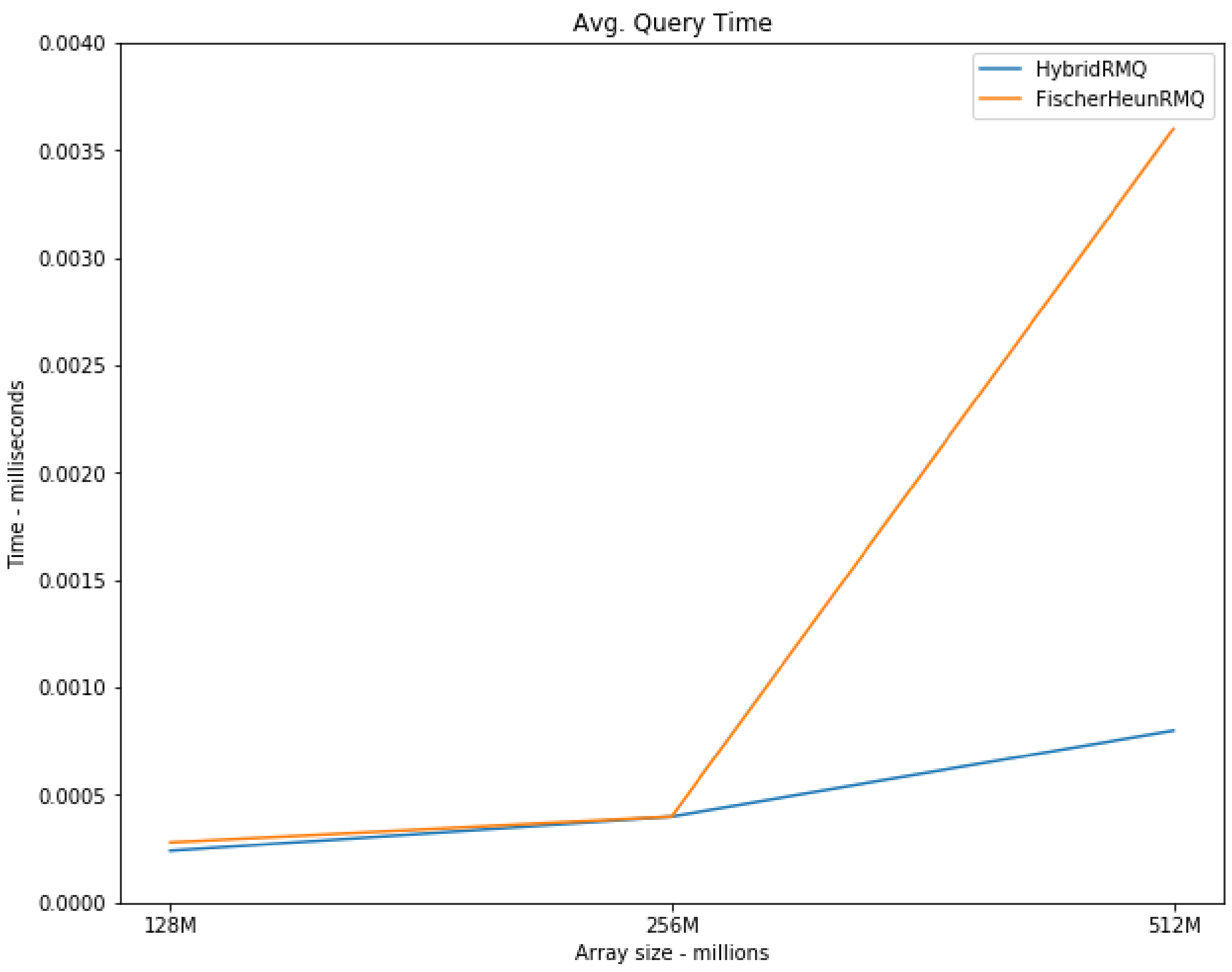
I have randomly generated 3 input files containing 128 million, 256 million & 512 million integers respectively, out of a range of 1 – 1 million. The algorithm is developed by testing on these input files. Since the size of the input files was too large to upload with the project, I have created another method for generating sequential integers for each test case. For making the below report I have created 3 test cases with the above-mentioned sizes. Then the two RMQ structure are pre-processed by passing the input data as an argument. The time taken for pre-processing both structures is noted. The above step is repeated 10 times for each test case and values are average to arrive at the average pre-processing time for both structures on the 3 test files.

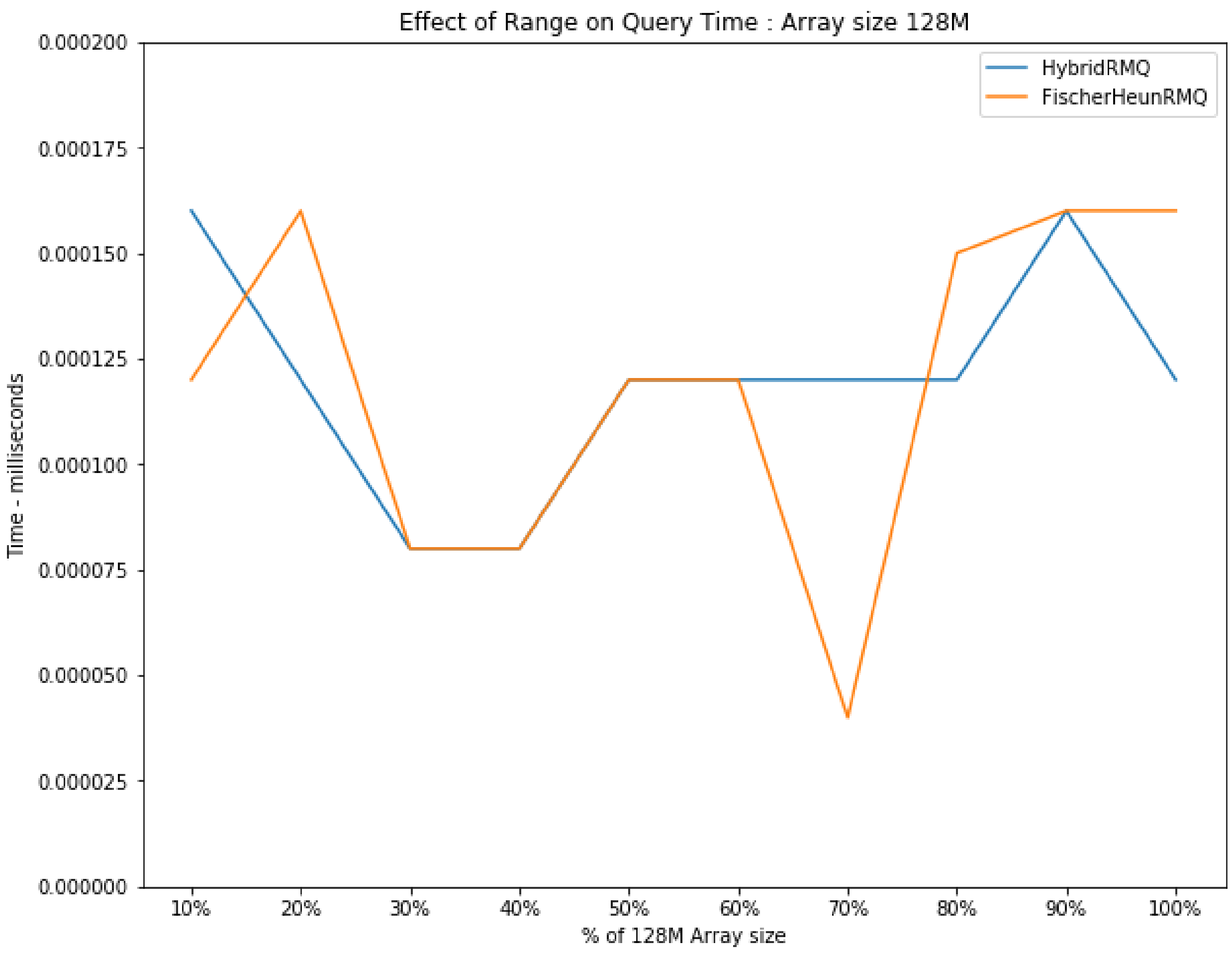
For calculating the average querying time, I query the above structures on 10000 different randomly generated ranges and average the individual query times for each test file.

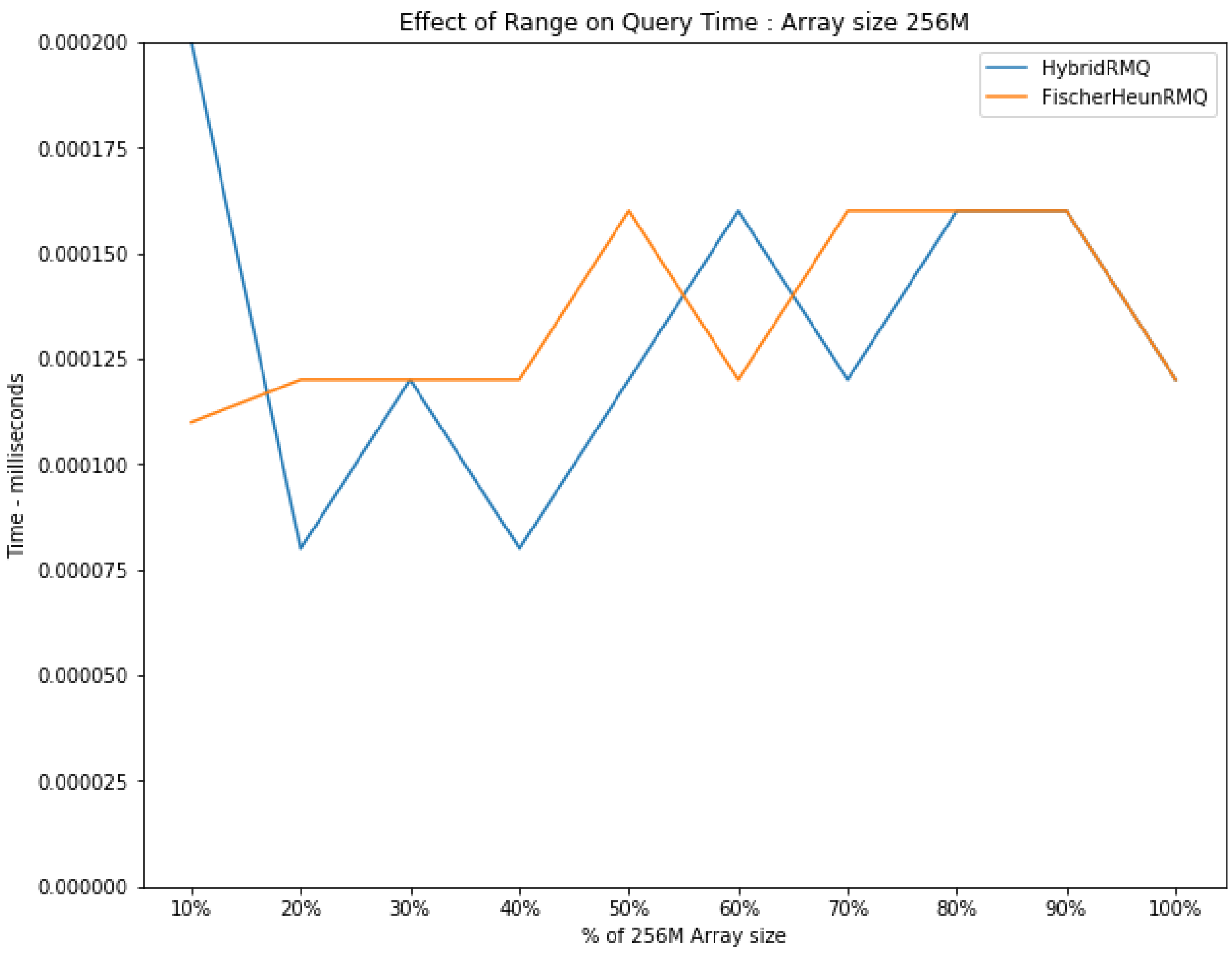
For evaluating the effect of range-size on the performance of both structure I test the querying time for each input file on different ranges of length 10%, 20%, .., 100% of the total array size and store the time taken. This step is performed 100000 times to arrive at the average query time for each range, respectively.

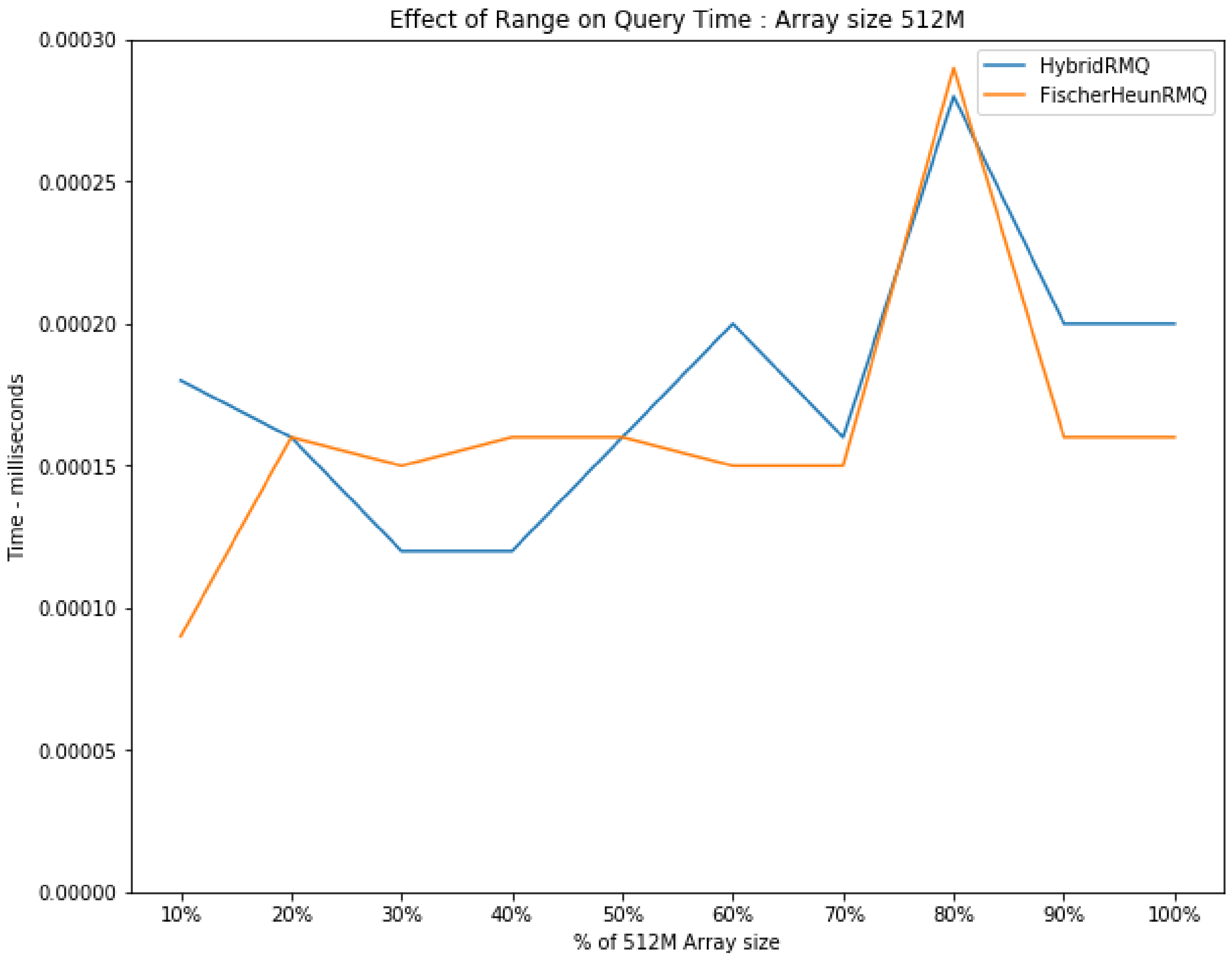
Experimental Evaluation:




Avg. Query Time
HybridRMQ
FischerHeunRMQ
Time - milliseconds
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
128M
256M
512M
Array size - millions


Effect of Range on Query Time : Array size 128M
HybridRMQ
FischerHeunRMQ
Time - milliseconds
% of 128M Array size
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%


Effect of Range on Query Time : Array size 256M
HybridRMQ
FischerHeunRMQ
Time - milliseconds
% of 256M Array size
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%


Effect of Range on Query Time : Array size 512M
HybridRMQ
FischerHeunRMQ
Time - milliseconds
% of 512M Array size
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0.00030
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

Summary:

FischerHeunRMQ has the same or better time complexity for both pre-processing and querying times as compared to HybridRMQ but this times complexity hides some pretty large constants. So even though it is a much better algorithm theoretically, but due to complex processing, HybridRMQ performs better in real life.

The pre-processing time of FischerHeunRMQ is much higher than HybridRMQ. This is to be expected as it does full pre-processing for BlockRMQ where as HybridRMQ does no pre-processing for BlockRMQ. The pre-processing time of FischerHeunRMQ increases almost linearly with the increase in input size.

The average query time of both algorithms is very similar, except for the last test case. This is a surprise as the query time of FischerHeunRMQ is O(1) where as that of HybridRMQ is O(log n), so the former should be faster. But this can be attributed to multiple memory lookups required for querying in FischerHeunRMQ, which add up to the total query time. The average query time of both algorithms also increases with increase in input size.

Finally, by looking at the last 2 tables we can clearly see the effect of range on HybridRMQ and FischerHeunRMQ. Neither of the algorithms are affected by querying on small or large ranges. Most of the average query time is scattered around a median value with minor deviations for individual inputs. So, we can say that the size of the query range doesn't affect the query time.