```
(base) DN0a2450ac:starter-code yizhouqian$ ./main_q2_part6
tests_q2.cpp:92:Test1   TEST PASSED.
tests_q2.cpp:109:Test2  TEST PASSED.
tests_q2.cpp:121:Test3  TEST PASSED.
tests_q2.cpp:140:Test4  TEST PASSED.
tests_q2.cpp:156:Test5  TEST PASSED.
Serial Radix Sort: PASS
Parallel Radix Sort: PASS
stl: 0.301696
serial radix: 0.071935
parallel radix: 0.065288
Threads Blocks / Timing
         1       2       4       8       12      16      24      32      40      48
   1    0.078   0.077   0.078   0.082   0.092   0.089   0.093   0.098   0.109   0.120
   2    0.078   0.050   0.046   0.052   0.060   0.063   0.060   0.061   0.067   0.074
   4    0.076   0.058   0.046   0.052   0.060   0.059   0.060   0.066   0.076   0.077
   8    0.078   0.046   0.049   0.058   0.056   0.065   0.066   0.067   0.069   0.079
  12    0.083   0.046   0.049   0.051   0.060   0.066   0.064   0.071   0.076   0.079
  16    0.083   0.060   0.048   0.053   0.055   0.058   0.061   0.068   0.081   0.079
  24    0.081   0.046   0.053   0.050   0.059   0.068   0.064   0.065   0.070   0.080
  32    0.089   0.049   0.050   0.058   0.058   0.057   0.065   0.067   0.075   0.078
  40    0.086   0.049   0.053   0.061   0.055   0.058   0.061   0.068   0.070   0.078
  48    0.087   0.051   0.053   0.055   0.060   0.069   0.068   0.073   0.070   0.075
Benchmark runs: PASS
```

For the last problem, we print the timing for different combination of number of threads/number of blocks. As we can see here, the minimal appears when we have 8 threads/2 blocks or 12 threads/2 blocks. However, we notice here the general trend is that the timing goes to almost minimal (around 0.05s) near the center of the table and increases as it goes to the margin of the table. There are several reasons for this.

First, when there are too few blocks, the code cannot use all the threads efficiently. Since there are not many tasks, most of them will always be idle. When there are too many blocks, the final step where we populate output from BlockExScan becomes the bottleneck of the runtime, which makes having more threads no longer meaningful. When we have too few threads, the number of blocks does not matter. Since there are too few threads to assign tasks to, we have limited parallelization and increasing the number of blocks cannot resolve this problem. Finally, when we have too many threads, if the number of blocks is less than the number of threads, then the parallelization is limited by the number of blocks. However, if the number of blocks is more than the number of threads, then we will run into the same problem of the runtime bottleneck as in the case of too many blocks. Having too many threads will also makes the overhead of parallelization significant, thus increasing the overall runtime. Hence, the optimal choice of number of blocks and number of threads will be somewhere near the center of the table, which is near 24 threads/8 blocks in this case.