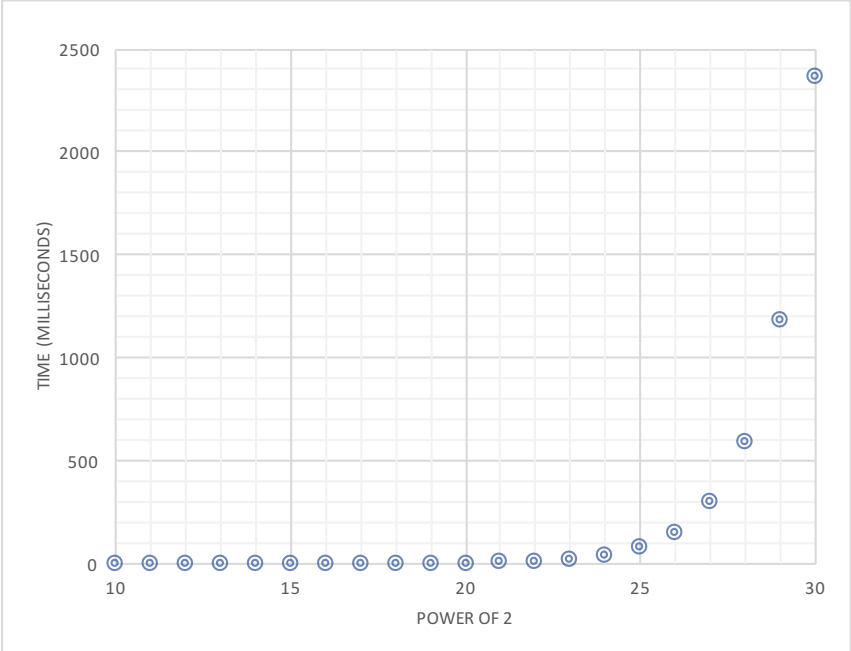
1c.



3f.

```
29  void mmul3(const double* A, const double* B_T, double* C, const std::size_t n)
30  {
31      int len = (int)n;
32      for (int A_row_iter = 0; A_row_iter < len; A_row_iter++) {
33          for (int B_col_iter = 0; B_col_iter < len; B_col_iter++) {
34              C[A_row_iter*n + B_col_iter] = 0;
35              for (int k = 0; k < len; k++) {
36                  C[A_row_iter*n + B_col_iter] += A[A_row_iter*n + k] * B_T[B_col_iter*n + k];
37              }
38          }
39      }
40  }
41
42  void mmul4(const double* A_T, const double* B, double* C, const std::size_t n)
43  {
44      int len = (int)n;
45      for (int A_row_iter = 0; A_row_iter < len; A_row_iter++) {
46          for (int B_col_iter = 0; B_col_iter < len; B_col_iter++) {
47              C[A_row_iter*n + B_col_iter] = 0;
48              for (int k = 0; k < len; k++) {
49                  C[A_row_iter*n + B_col_iter] += A_T[A_row_iter + k*n] * B[B_col_iter + k*n];
50              }
51          }
52      }
53  }
54
```

Since there is cache in hardware, we could utilize cache to speed up program execution time by either utilizing spatial locality or temporal locality. mmul3 is such a function that uses spatial locality in its most inner loop (line 36) which do a stride 1 increment when k increases by 1. On the contrary, mmul4 do a stride n increment when k increases by 1 (line 49), so mmul4 is less cache-friendly. As a result, mmul3 is much quicker than mmul4.