```
/// CSCI 176 Program 3
2
 3
    /// Kenneth Willeford
4
    111
5
    ///
           This program performs parallel matrix multiplication on matrices
    ///
6
           of the form specified in prog3.pdf.
7
    ///
           Once compiled the program is ran as so...
8
               <exe name> <L value> <m value> <n value> <number of thread>
    ///
9
   ///
               ie our required test cases...
10
   ///
               g++ mmultiply.cpp && ./a.out 1003 2000 3000 1 > output1.txt
               g++ mmultiply.cpp && ./a.out 1003 2000 3000 2 > output2.txt
11
    ///
12
   ///
               g++ mmultiply.cpp && ./a.out 1003 2000 3000 4 > output4.txt
13
   ///
               g++ mmultiply.cpp && ./a.out 1003 2000 3000 8 > output8.txt
14 ///
               The relevant output for these test cases will be provided separately.
15
    ///
           Additional Notes-
16
    ///
               I thought I would also turn this program into a practice in
17
   ///
               cache optimization. I create a 'matrix' class that can be built
18
   ///
               'rotated', which allows for row-major access on column elements.
19
   ///
               This is useful for optimizing the layout of data for the B operand.
20
    21
   #include<iostream>
                          // cout
22 #include<cstdlib>
                          // atoi
23 #include<vector>
                          // Dynamic Arrays (I prefer not to use the manual approach if I
    don't have to.)
24
   #include<ctime>
                          // For Benchmarking
25 #include<pthread.h>
                         // pthread t, mthread mutex t, lock, unlock, create, join
26
   #include <sstream>
                          // for capturing output to send to the guarded cout
27
    using namespace std;
28
    //-----
29
   // Object Definitions / Implementation
30
   //-----
31
    class matrix{
        // This bool is set when the matrix is created. It marks whether or not the matrix
32
        was created while 'rotated.'
33
        // A rotated matrix's its column vectors are in row major.
34
       bool rotated:
        // The actual matrix as a 2D Dynamic Array
35
       vector<vector<long long int> > m;
36
37
        // The Number of Rows/Columns
38
       long long int r,c;
39
        public:
40
        // Empty Constructor
41
       matrix(){}
42
        // long long intiializes Matrix with its Dimensions.
43
       matrix(long long int rows, long long int columns, bool rot = false){
44
           rotated = rot;
45
           if(rotated)
               m = vector<vector<long long int> >(columns, vector<long long int>(rows,0));
46
47
48
               m = vector<vector<long long int> >(rows, vector<long long int>(columns, 0));
49
           r = rows;
50
           c = columns;
51
52
        // Provides the number of rows within the matrix.
53
        long long int rows() { return r; }
54
        // Provides the number of columns within the matrix.
55
        long long int columns() { return c; }
56
        // Retrieves an individual value in the matrix by row and column.
        long long int get(long long int row, long long int column){
57
58
           if(rotated)
59
               return m[column][row];
60
           else
61
               return m[row][column];
62
        }
```

```
63
        // Assigns an inidivdual value to the matrix by row and column.
        void assign(long long int row, long long int column, long long int val){
 64
 65
           if(rotated)
 66
              m[column][row] = val;
 67
           else
 68
              m[row][column] = val;
 69
        }
 70
    //-----
 71
 72
    // Global Values
 73
    74
    namespace GLOBAL {
75
        // Corresponds to the L,m,n values in the prog3.pdf specifications
76
        long long int L,m,n;
 77
        // remainder rows that will need to be added in during cyclical assignment.
 78
        long long int remainderRows = 0;
 79
        // The number of threads
80
        long long int num threads = 1;
 81
        // The the input matrices and the output matrix.
 82
        matrix A,B,C;
 83
        // The execution time of a benchmark in seconds.
 84
        double executionTime;
 85
    };
    86
 87
    // Function Definitions
    //----
 88
89
    // Performs a cout command on a string guarded by a mutex.
 90
    void cout semaphore(string);
    // Master function for the matrix multiplication. Statically generates the needed
 91
    threads and executes the matrix multiplication.
 92
    void*matrixMultiplicationMaster(void*);
    // Gets four command line arguments. L,m,n,num threads
 93
 94
    void getCommandLineArguments(char* argv[]);
 95
    // Utilizes L,m,n and num threads to initialize the matrices and any needed metadata.
96
    void initializeMatrices();
97
    // Performs the dot product between a row in A and a column in B.
98
    long long int dotProduct(long long int rowA, long long int colB);
    // Slave function for matrix multiplication. Performs part of the multiplication.
99
100
    void* matrixMultiplication(void*);
    // Takes in a generic function and performs benchmarking on it, stores the result
101
    globally.
102
    void benchmark(void*(*)(void*),void*arg);
103
    // Prints the final output.
104
    void printResults();
    //-----
105
    // Main
106
    107
108
    int main(int argc, char* argv[]){
109
        getCommandLineArguments(argv);
110
        initializeMatrices();
111
        benchmark(matrixMultiplicationMaster, NULL);
112
        printResults();
113
    //----
114
115
    // Function Implementations
    116
117
    void* matrixMultiplication(void* arg) {
118
        // Get Current Thread ID
119
        long long int my rank = (long long int)arg;
120
        // Output Data to screen
121
       ostringstream ss;
        ss << "Thread " << my rank << ": " << my rank << " ~ " << GLOBAL::C.rows() << ",
122
        step " << GLOBAL::num threads << endl;</pre>
123
        cout semaphore(ss.str());
```

```
124
          // If there are 'remainderRows' remaining...
125
          if(my rank < GLOBAL::remainderRows) {</pre>
126
              // Then go ahead and account for extra rows...
127
              for(long long int i = my rank; i < GLOBAL::C.rows(); i+=GLOBAL::num threads)</pre>
128
                  for(long long int j = 0; j < GLOBAL::C.columns(); j++)</pre>
129
                       GLOBAL::C.assign(i,j,dotProduct(i,j));
          // Otherwise...
130
131
          } else {
132
              // Don't accout for extra rows. (don't hit the same rows twice.)
133
              for(long long int i = my rank; i < GLOBAL::C.rows() - GLOBAL::remainderRows;</pre>
              i+=GLOBAL::num threads)
134
                  for(long long int j = 0; j < GLOBAL::C.columns(); j++)</pre>
135
                      GLOBAL::C.assign(i,j,dotProduct(i,j));
136
137
          // So the compiler doesn't return a warning.
138
          return NULL;
139
      }
140
141
      long long int dotProduct(long long int rowA, long long int colB) {
142
          long long int sum = 0;
143
          long long int range = GLOBAL::m;
144
          for (long long int i = 0; i < range; i++)
145
              // Perform Dot Product to Build up sum
146
              sum += GLOBAL::A.get(rowA,i) * GLOBAL::B.get(i,colB);
147
          return sum;
148
      }
149
150
     void getCommandLineArguments(char* argv[]) {
151
          GLOBAL::L = atoi(argv[1]);
152
          GLOBAL::m = atoi(argv[2]);
153
          GLOBAL::n = atoi(argv[3]);
154
          // If the number of threads specified are less than 1(or not specified at all)
          default to 1.
155
          GLOBAL::num threads = atoi(argv[4]) < 1 ? 1:atoi(argv[4]);
156
          // Print to Screen
157
          cout << "L=" << GLOBAL::L << ",m=" << GLOBAL::m << ",n=" << GLOBAL::n << endl;</pre>
158
      }
159
160
     void initializeMatrices(){
161
          // Initialize Matrix A according to prog3.pdf Specifications
162
          GLOBAL::A = matrix(GLOBAL::L,GLOBAL::m);
163
          for(long long int i = 0; i < GLOBAL::A.rows(); i++)</pre>
164
              for(long long int j = 0; j < GLOBAL::A.columns(); j++)</pre>
165
                  GLOBAL::A.assign(i,j,i+j+1);
166
          // Initialize Matrix B according to prog3.pdf Specifications
167
          GLOBAL::B = matrix(GLOBAL::m,GLOBAL::n,true);
168
          for(long long int i = 0; i < GLOBAL::B.rows(); i++)</pre>
169
              for(long long int j = 0; j < GLOBAL::B.columns(); j++)</pre>
170
                  GLOBAL::B.assign(i,j,i+j);
171
          // Initialize Matrix C dimensions according to prog3.pdf Specifications
172
          GLOBAL::C = matrix(GLOBAL::L,GLOBAL::n);
173
          // Determine the remainder rows. Guarding this statement is proabbly unnecessary.
174
          if(GLOBAL::num threads > 0)
175
              GLOBAL::remainderRows = GLOBAL::C.rows() % GLOBAL::num threads;
176
      }
177
178
     void benchmark(void*(*f)(void*),void*arg){
179
          // Begin Timing
180
          clock t t = clock();
          // Run Function
181
182
          f(arg);
183
          // Get Timing
184
          GLOBAL::executionTime = ((float)(clock()-t))/CLOCKS PER SEC;
185
      }
```

```
186
187
     void*matrixMultiplicationMaster(void*) {
188
          // Build up the requested number of threads and launch them with their IDs.
189
          vector<pthread t> threads = vector<pthread t>(GLOBAL::num threads);
          for(long long int i = 0; i < GLOBAL::num threads; i++)</pre>
190
191
               pthread create(&threads[i], NULL, matrixMultiplication, (void*)i);
          // Wait for each thread to finish before continuing.
192
193
          for(long long int i = 0; i < GLOBAL::num threads; i++)</pre>
194
              pthread join(threads[i], NULL);
195
     }
196
197
     void printResults(){
198
          // Print the related sub-matrix
199
          cout << "===C:first 20*first 10===" << endl;</pre>
          for (long long int i = 0; i < 20 && i < GLOBAL::C.rows(); <math>i++) {
200
201
               for (long long int j = 0; j < 10 && j < GLOBAL::C.columns(); <math>j++) {
202
                   cout << GLOBAL::C.get(i,j) << " ";</pre>
203
              }
204
              cout << endl;</pre>
205
          }
206
          // Print the related sub-matrix
207
          cout << "===C:last 20*last 10===" << endl;
          for (long long int i = GLOBAL::C.rows() - 20; i > 0 && i < GLOBAL::C.rows(); i++) {
208
209
               for (long long int j = GLOBAL::C.columns() - 10; j > 0 && j <
              GLOBAL::C.columns(); j++){
210
                   cout << GLOBAL::C.get(i,j) << " ";</pre>
211
              }
212
              cout << endl;</pre>
213
          }
214
          // Print benchmark information
215
          cout << "Time taken (sec) = " << GLOBAL::executionTime << endl;</pre>
216
      }
217
     void cout semaphore(string s){
218
219
          // Shared lock between function calls. Protects cout.
220
          static pthread mutex t lock = PTHREAD MUTEX INITIALIZER;
221
          pthread mutex lock(&lock);
          cout << s << endl;</pre>
222
223
          pthread mutex unlock(&lock);
224
      }
```

L=1003, m=2000, n=3000Thread_0: 0 ~ 1003, step 1

```
===C:first_20*first_10===
2666666000 2668667000 2670668000 2672669000 2674670000 2676671000 2678672000
2680673000 2682674000 2684675000
2668665000 2670668000 2672671000 2674674000 2676677000 2678680000 2680683000
2682686000 2684689000 2686692000
2670664000 2672669000 2674674000 2676679000 2678684000 2680689000 2682694000
2684699000 2686704000 2688709000
2672663000 2674670000 2676677000
                                    2678684000 2680691000 2682698000 2684705000
2686712000 2688719000 2690726000
2674662000 2676671000 2678680000 2680689000 2682698000 2684707000 2686716000
2688725000 2690734000 2692743000
2676661000 2678672000 2680683000
                                   2682694000 2684705000 2686716000 2688727000
2690738000 2692749000 2694760000
2678660000 2680673000 2682686000 2684699000 2686712000 2688725000 2690738000 2692751000 2694764000 2696777000
2680659000 2682674000 2684689000
                                    2686704000 2688719000 2690734000 2692749000
2694764000 2696779000 2698794000
2682658000 2684675000 2686692000 2688709000 2690726000 2692743000 2694760000
2696777000 2698794000 2700811000
2684657000 2686676000 2688695000 2690714000 2692733000 2694752000 2696771000
2698790000 2700809000 2702828000
2686656000 2688677000 2690698000 2692719000 2694740000 2696761000 2698782000
2700803000 2702824000 2704845000
2688655000 2690678000 2692701000 2694724000 2696747000 2698770000 2700793000
2702816000 2704839000 2706862000
2690654000 2692679000 2694704000 2696729000 2698754000 2700779000 2702804000
2704829000 2706854000 2708879000
2692653000 2694680000 2696707000 2698734000 2700761000 2702788000 2704815000
2706842000 2708869000 2710896000
2694652000 2696681000 2698710000 2700739000 2702768000 2704797000 2706826000
2708855000 2710884000 2712913000
2696651000 2698682000 2700713000 2702744000 2704775000 2706806000 2708837000
2710868000 2712899000 2714930000
2698650000 2700683000 2702716000 2704749000 2706782000 2708815000 2710848000
2712881000 2714914000 2716947000
2700649000 2702684000 2704719000 2706754000 2708789000 2710824000 2712859000 2714894000 2716929000 2718964000 2702648000 2704685000 2706722000 2708759000 2710796000 2712833000 2714870000
2716907000 2718944000 2720981000
2704647000 2706686000 2708725000 2710764000 2712803000 2714842000 2716881000
2718920000 2720959000 2722998000
===C:last_20*last_10===
16493013000 16496980000 16500947000 16504914000 16508881000 16512848000 16516815000
16520782000 16524749000 16528716000
16500992000 16504961000 16508930000
                                       16512899000 16516868000 16520837000 16524806000
16528775000 16532744000 16536713000
16508971000 16512942000 16516913000 16520884000 16524855000 16528826000 16532797000
16536768000 16540739000 16544710000
16516950000 16520923000 16524896000 16528869000 16532842000 16536815000 16540788000
16544761000 16548734000 16552707000
16524929000 16528904000 16532879000 16536854000 16540829000 16544804000 16548779000
16552754000 16556729000 16560704000
16532908000 16536885000 16540862000
                                       16544839000 16548816000 16552793000 16556770000
16560747000 16564724000 16568701000
16540887000 16544866000 16548845000 16552824000 16556803000 16560782000 16564761000
16568740000 16572719000 16576698000
16548866000 16552847000 16556828000 16560809000 16564790000 16568771000 16572752000
16576733000 16580714000 16584695000 16556845000 16560828000 16564811000 16568794000 16572777000 16576760000 16580743000 16584726000 16588709000 16592692000
```

			output1.txt				
16564824000	16568809000	16572794000			16584749000	16588734000	
16592719000	16596704000	16600689000					
		16580777000	16584764000	16588751000	16592738000	16596725000	
		16608686000	16502740000	16506730000	16600737000	16604716000	
16580782000	16584771000 16612694000	16588760000 16616683000	16592749000	16596/38000	16600727000	16604716000	
		16596743000	16600734000	16604725000	16608716000	16612707000	
	16620689000	16624680000	10000734000	10004723000	10000710000	10012707000	
	16600733000	16604726000	16608719000	16612712000	16616705000	16620698000	
	16628684000	16632677000					
	16608714000	16612709000	16616704000	16620699000	16624694000	16628689000	
		16640674000	16624600000	1662060600	16633603000	1662660000	
		16620692000	16624689000	16628686000	16632683000	16636680000	
	16644674000 16624676000	16648671000 16628675000	16622674000	16626672000	16640672000	16644671000	
		16656668000	10032074000	10030073000	10040072000	10044071000	
16628656000		16636658000	16640659000	16644660000	16648661000	16652662000	
16656663000		16664665000					
16636635000	16640638000	16644641000	16648644000	16652647000	16656650000	16660653000	
16664656000	16668659000	16672662000					
16644614000	16648619000	16652624000	16656629000	16660634000	16664639000	16668644000	
16672649000		16680659000					
Time taken (sec) = 196.517							

output2.txt

L=1003,m=2000,n=3000 Thread_0: 0 \sim 1003, step 2 Thread_1: 1 \sim 1003, step 2

C.finct	20%finc+ 10	^				
===C:first_		2670668000	2672669000	2674670000	2676671000	2678672000
2680673000			2072003000	207 107 0000	2070071000	2070072000
2668665000	2670668000	2672671000	2674674000	2676677000	2678680000	2680683000
2682686000						
		2674674000	2676679000	2678684000	2680689000	2682694000
2684699000			2679694000	2690601000	2692609000	2684705000
2686712000		2676677000	2070004000	2000091000	2002090000	2684703000
		2678680000	2680689000	2682698000	2684707000	2686716000
2688725000						
		2680683000	2682694000	2684705000	2686716000	2688727000
2690738000			262462222	2505=4200	2600=25000	20072000
26/8660000	26806/3000	2682686000	2684699000	2686/12000	2688725000	2690738000
2692751000		2684689000	2696704000	2600710000	2600724000	2602740000
2694764000			2000704000	2000/19000	2090734000	2092749000
		2686692000	2688709000	2690726000	2692743000	2694760000
2696777000						
2684657000	2686676000	2688695000	2690714000	2692733000	2694752000	2696771000
2698790000						
		2690698000	2692719000	2694740000	2696761000	2698782000
2700803000			2004724000	2000747000	2609770000	2700702000
2702816000		2692701000	2694724000	2696747000	2698770000	2700793000
		2694704000	2696729000	2698754000	2700779000	2702804000
2704829000			2030723000	2030734000	2700773000	2702004000
		2696707000	2698734000	2700761000	2702788000	2704815000
2706842000						
		2698710000	2700739000	2702768000	2704797000	2706826000
2708855000	2710884000	2712913000	2702744000	2704775000	2706006000	2700027000
		2700713000	2702744000	2/04//5000	2706806000	2708837000
2710868000	2712899000	2714930000	2704740000	2706782000	2709915000	2710848000
2712881000			2704743000	2700762000	2700013000	2710040000
		2704719000	2706754000	2708789000	2710824000	2712859000
2714894000						
2702648000	2704685000	2706722000	2708759000	2710796000	2712833000	2714870000
2716907000	2718944000	2720981000				
		2708725000	2710764000	2712803000	2714842000	2716881000
2718920000						
===C:last_2			000 1650401	4000 165088	21000 165129	348000 16516815000
		00 165287160		+000 1030000	31000 103120	348000 10310813000
				9000 1651680	58000 165208	337000 16524806000
		00 165367130				
				4000 165248	55000 165288	326000 16532797000
		00 165447100				
				9000 1653284	12000 165368	315000 16540788000
16544/61000	16548/3400	00 165527070)UU 100 1652695.	4000 16E400'	20000 16544	304000 16548779000
16552754000	1655672904	00 165607040)UU 100 10330634	4000 103406	29000 103440	304000 16346779000
				9000 165488	16000 16552	793000 16556770000
		00 165687010		200 200 100.		12300 10330,70000
16540887000	1654486600	00 165488450	000 1655282	4000 1655680	03000 165607	782000 16564761000
16568740000	1657271900	00 165766980	000			
				9000 1656479	90000 165687	771000 16572752000
16576733000	1658071400	00 165846950	000			

L=1003, m=2000, n=3000

	output3.txt			
16540887000 16544866000 165488		16556803000	16560782000	16564761000
16568740000 16572719000 165766				
16548866000 16552847000 165568		16564790000	16568771000	16572752000
16576733000 16580714000 165846 16556845000 16560828000 165648		16572777000	16576760000	16580743000
16584726000 16588709000 165926		10372777000	10370700000	10300743000
16564824000 16568809000 165727		16580764000	16584749000	16588734000
16592719000 16596704000 166006				
16572803000 16576790000 165807		16588751000	16592738000	16596725000
16600712000 16604699000 166086				
16580782000 16584771000 165887		16596738000	16600727000	16604716000
16608705000 16612694000 166166 16588761000 16592752000 165967		16604725000	16609716000	16612707000
16616698000 16620689000 166246	=	10004723000	10008/10000	10012/0/000
16596740000 16600733000 166047		16612712000	16616705000	16620698000
16624691000 16628684000 166326		10011,11000	10010, 03000	1002003000
16604719000 16608714000 166127	09000 16616704000	16620699000	16624694000	16628689000
16632684000 16636679000 166406				
16612698000 16616695000 166206		16628686000	16632683000	16636680000
16640677000 16644674000 166486		16626672000	16640672000	16644671000
16620677000 16624676000 166286 16648670000 16652669000 166566		16636673000	16640672000	16644671000
16628656000 16632657000 166366		16644660000	16648661000	16652662000
16656663000 16660664000 166646		10011000000	10010001000	10032002000
16636635000 16640638000 166446	41000 16648644000	16652647000	16656650000	16660653000
16664656000 16668659000 166726				
16644614000 16648619000 166526		16660634000	16664639000	16668644000
16672649000 16676654000 166806	59000			
Time taken (sec) = 92.135				

output4.txt

```
L=1003, m=2000, n=3000
Thread_0: 0 \sim 1003, step 8
Thread_1: 1 \sim 1003, step 8
Thread_2: 2 \sim 1003, step 8
Thread_3: 3 \sim 1003, step 8
Thread_4: 4 \sim 1003, step 8
Thread_5: 5 \sim 1003, step 8
Thread_6: 6 ~ 1003, step 8
Thread_7: 7 \sim 1003, step 8
===C:first_20*first_10=== 2666666000 2668667000 2670668000 2672669000 2674670000 2676671000 2678672000
2680673000 2682674000 2684675000
2668665000 2670668000 2672671000 2674674000 2676677000 2678680000 2680683000
2682686000 2684689000 2686692000
2670664000 2672669000 2674674000 2676679000 2678684000 2680689000 2682694000
2684699000 2686704000 2688709000
2672663000 2674670000 2676677000 2678684000 2680691000 2682698000 2684705000
2686712000 2688719000 2690726000
2674662000 2676671000 2678680000
                                   2680689000 2682698000 2684707000 2686716000
2688725000 2690734000 2692743000
2676661000 2678672000 2680683000 2682694000 2684705000 2686716000 2688727000
2690738000 2692749000 2694760000
2678660000 2680673000 2682686000 2684699000 2686712000 2688725000 2690738000
2692751000 2694764000 2696777000
                                   2686704000 2688719000 2690734000 2692749000
2680659000
           2682674000 2684689000
2694764000 2696779000 2698794000
2682658000 2684675000 2686692000 2688709000 2690726000 2692743000 2694760000
2696777000 2698794000 2700811000
2684657000 2686676000 2688695000 2690714000 2692733000 2694752000 2696771000
2698790000 2700809000 2702828000
2686656000 2688677000 2690698000 2692719000 2694740000 2696761000 2698782000
2700803000 2702824000 2704845000
2688655000 2690678000 2692701000 2694724000 2696747000 2698770000 2700793000
2702816000 2704839000 2706862000
2690654000 2692679000 2694704000 2696729000 2698754000 2700779000 2702804000
2704829000 2706854000 2708879000
2692653000 2694680000 2696707000 2698734000 2700761000 2702788000 2704815000
2706842000 2708869000 2710896000
2694652000 2696681000 2698710000
                                   2700739000 2702768000 2704797000 2706826000
2708855000 2710884000
                        2712913000
2696651000 2698682000 2700713000 2702744000 2704775000 2706806000 2708837000
2710868000 2712899000 2714930000
2698650000 2700683000 2702716000 2704749000 2706782000 2708815000 2710848000
2712881000 2714914000 2716947000
2700649000 2702684000 2704719000 2706754000 2708789000 2710824000 2712859000 2714894000 2716929000 2718964000
2702648000 2704685000 2706722000 2708759000 2710796000 2712833000 2714870000
2716907000 2718944000 2720981000
2704647000 2706686000 2708725000 2710764000 2712803000 2714842000 2716881000
2718920000 2720959000 2722998000
===C:last_20*last_10===
164930130\overline{0}0 16496\overline{9}80000 16500947000 16504914000 16508881000 16512848000 16516815000
16520782000 16524749000 16528716000 16500992000 16504961000 16508930000 16512899000 16516868000 16520837000 16524806000 16528775000 16532744000 16536713000
```

	output4.txt						
16508971000 16512942000 16516913000		16524855000	16528826000	16532797000			
16536768000 16540739000 16544710000							
	16528869000	16532842000	16536815000	16540788000			
16544761000 16548734000 16552707000							
	16536854000	16540829000	16544804000	16548779000			
16552754000 16556729000 16560704000 16532908000 16536885000 16540862000	16544920000	16540016000	16552793000	16556770000			
16560747000 16564724000 16568701000	10344039000	10346610000	10332793000	10330770000			
	16552824000	16556803000	16560782000	16564761000			
16568740000 16572719000 16576698000	1033202 1000	10330003000	10300702000	10301701000			
16548866000 16552847000 16556828000	16560809000	16564790000	16568771000	16572752000			
16576733000 16580714000 16584695000							
	16568794000	16572777000	16576760000	16580743000			
16584726000 16588709000 16592692000	16576770000	16500764000	16504740000	16500724000			
16564824000 16568809000 16572794000 16592719000 16596704000 16600689000	165/6//9000	16580/64000	16584749000	16588734000			
	16594764000	16500751000	16592738000	16506725000			
16600712000 16604699000 16608686000	10304704000	10300731000	10392738000	10390723000			
	16592749000	16596738000	16600727000	16604716000			
16608705000 16612694000 16616683000							
16588761000 16592752000 16596743000	16600734000	16604725000	16608716000	16612707000			
16616698000 16620689000 16624680000							
	16608719000	16612712000	16616705000	16620698000			
16624691000 16628684000 16632677000	16616704000	1662060000	16624604000	1662060000			
	16616704000	16620699000	16624694000	16628689000			
16632684000 16636679000 16640674000 16612698000 16616695000 16620692000	16624689000	16628686000	16632683000	16636680000			
16640677000 16644674000 16648671000	10024009000	1002000000	10032063000	10030000000			
	16632674000	16636673000	16640672000	16644671000			
16648670000 16652669000 16656668000	1003207 1000	200500,5000	100 1007 2000	20011072000			
16628656000 16632657000 16636658000	16640659000	16644660000	16648661000	16652662000			
16656663000 16660664000 16664665000							
	16648644000	16652647000	16656650000	16660653000			
16664656000 16668659000 16672662000	16656620622	16660634600	16664630600	16660644000			
	10020073000	10000034000	16664639000	10008044000			
16672649000 16676654000 16680659000 Time taken (sec) = 99.401							
Time taken (Sec) = 33.401							