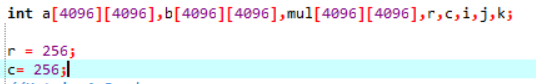
**Question 1-)**

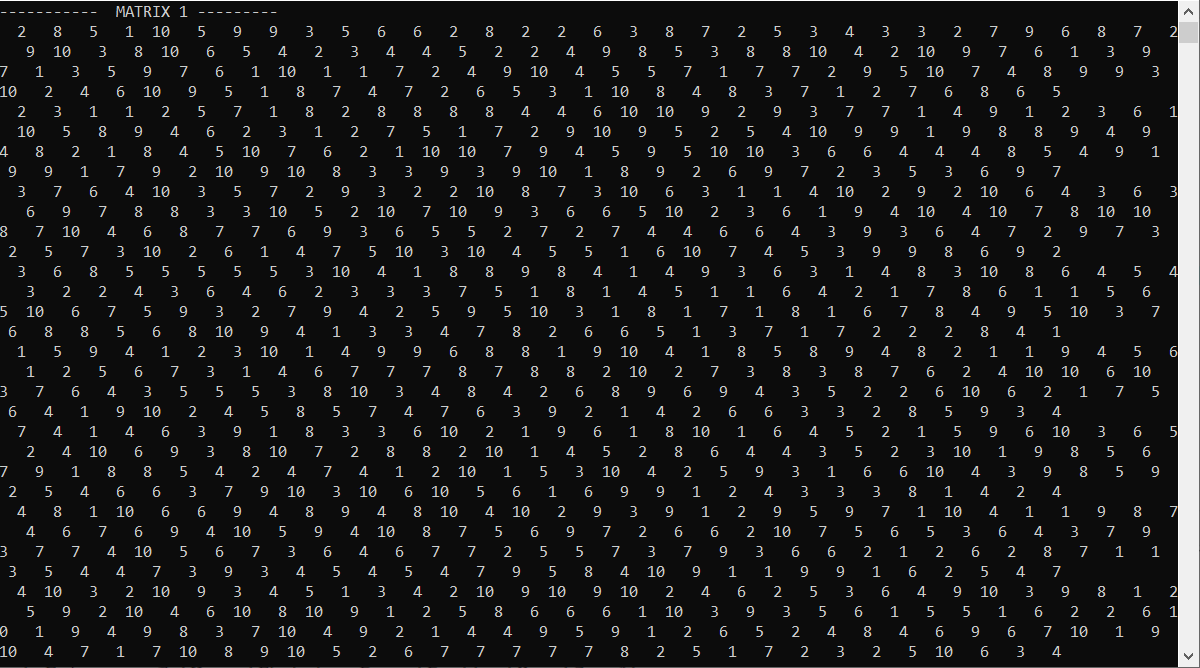
**Code :**

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  int main()  {  int a[4096][4096],b[4096][4096],mul[4096][4096],r,c,i,j,k;  r = 128;  c= 128;  //Matrix 1 Random  for(i=0;i<r;i++)  {  for(j=0;j<c;j++)  {  a[i][j]=rand() % 5 + 1;  }  }  char buffer[10];  char filename[] = "a";  sprintf(buffer, "%s%d.txt",filename,i);  FILE \*dosya1 = fopen(buffer, "w");  // Matrix 1 Printing  printf("----------- MATRIX 1 --------- \n");  int f,g=0;  for(f=0;f<i;f++){  for(g=0;g<j;g++){  printf(" %2d ",a[f][g]);  fprintf(dosya1, "%2d ",a[f][g]);  }  fprintf(dosya1,"\n");  printf("\n");  }  fclose(dosya1);    //Matrix 2 Random  for(i=0;i<r;i++)  {  for(j=0;j<c;j++)  {  b[i][j] = rand() % 5 + 1;  }  }  char buffer2[10];  char filename2[] = "b";  sprintf(buffer2, "%s%d.txt",filename2,j);  FILE \*dosya2 = fopen(buffer2, "w");  printf("\n -------------- MATRIX 2----------------\n");  int t,y=0;  for(t=0;t<i;t++){  for(y=0;y<j;y++){  printf(" %2d ",b[t][y]);  fprintf(dosya2, " %d ",b[t][y]);  }  fprintf(dosya2,"\n");  printf("\n");  }  fclose(dosya2);    printf(" \n\n-----------SERIAL MULTIPLACTION MATRIX ------------- \n");  for(i=0;i<r;i++)  {  for(j=0;j<c;j++)  {  mul[i][j]=0;  for(k=0;k<c;k++)  {  mul[i][j]+=a[i][k]\*b[k][j];  }  }  }  char buffer3[10];  char filename3[] = "result";  sprintf(buffer3, "%s%d.txt",filename3,j);  FILE \*result = fopen(buffer3, "w");  //for printing result  for(i=0;i<r;i++)  {  for(j=0;j<c;j++)  {  printf("%d\t",mul[i][j]);  fprintf(result, " %d ",mul[i][j]);  }  fprintf(result,"\n");  printf("\n");  }  return 0;  } |

**Output:**

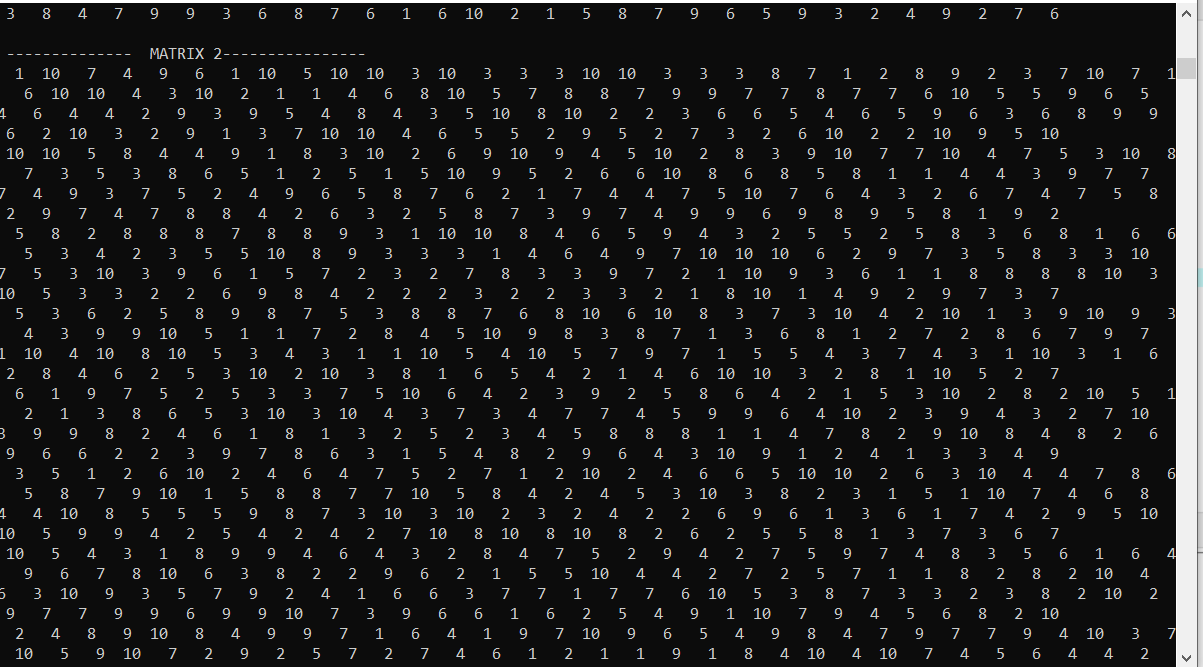
****

**Matrix 1**

****

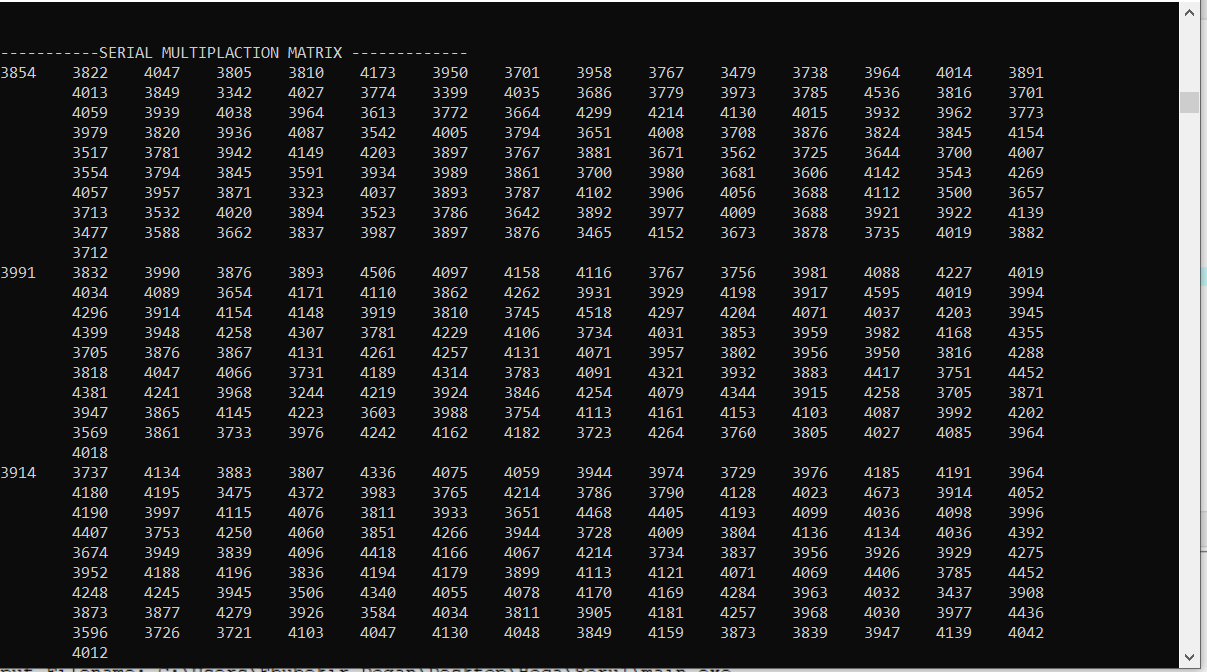
**a short section from matrix 1**

**Matrix 2**

****

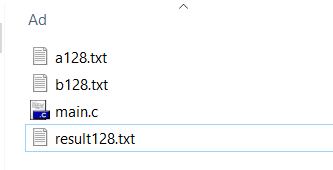
**a short section from matrix 2**

**Result Matrix:**

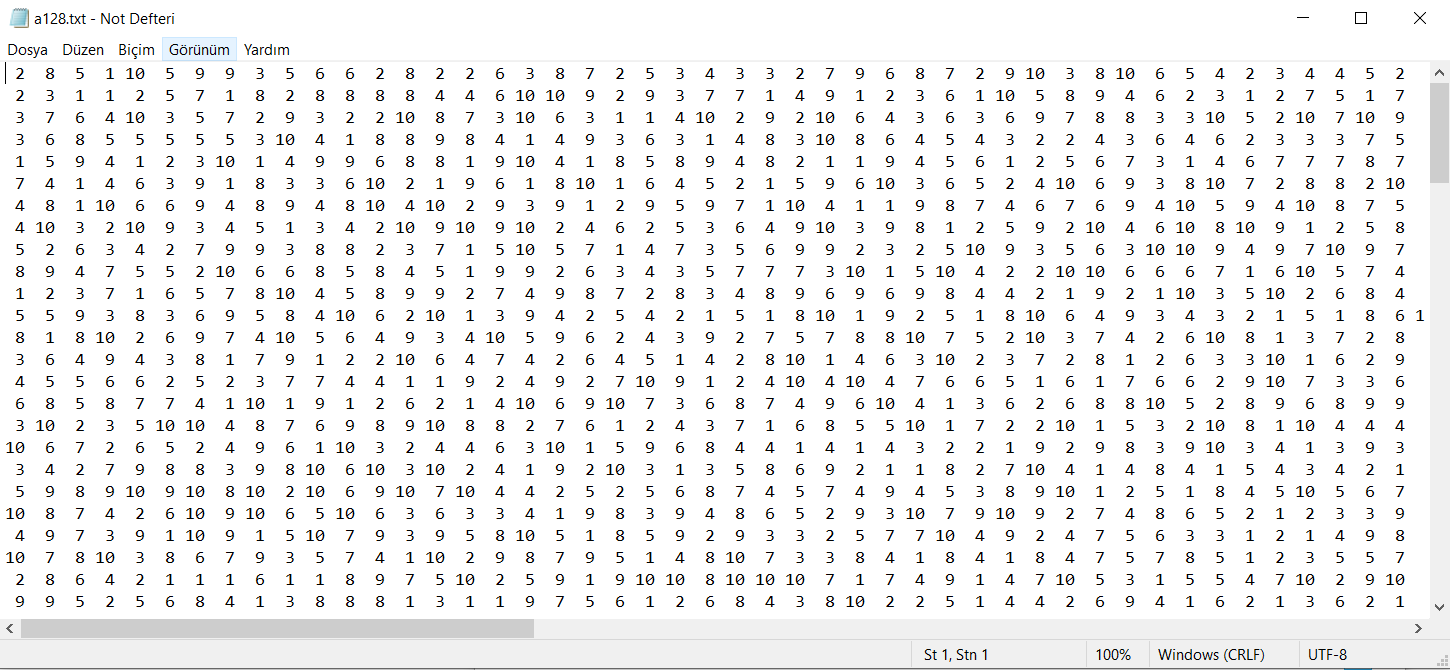
****

**a short section from result matrix**

**File Structure:**

****

**a128.txt file**

****

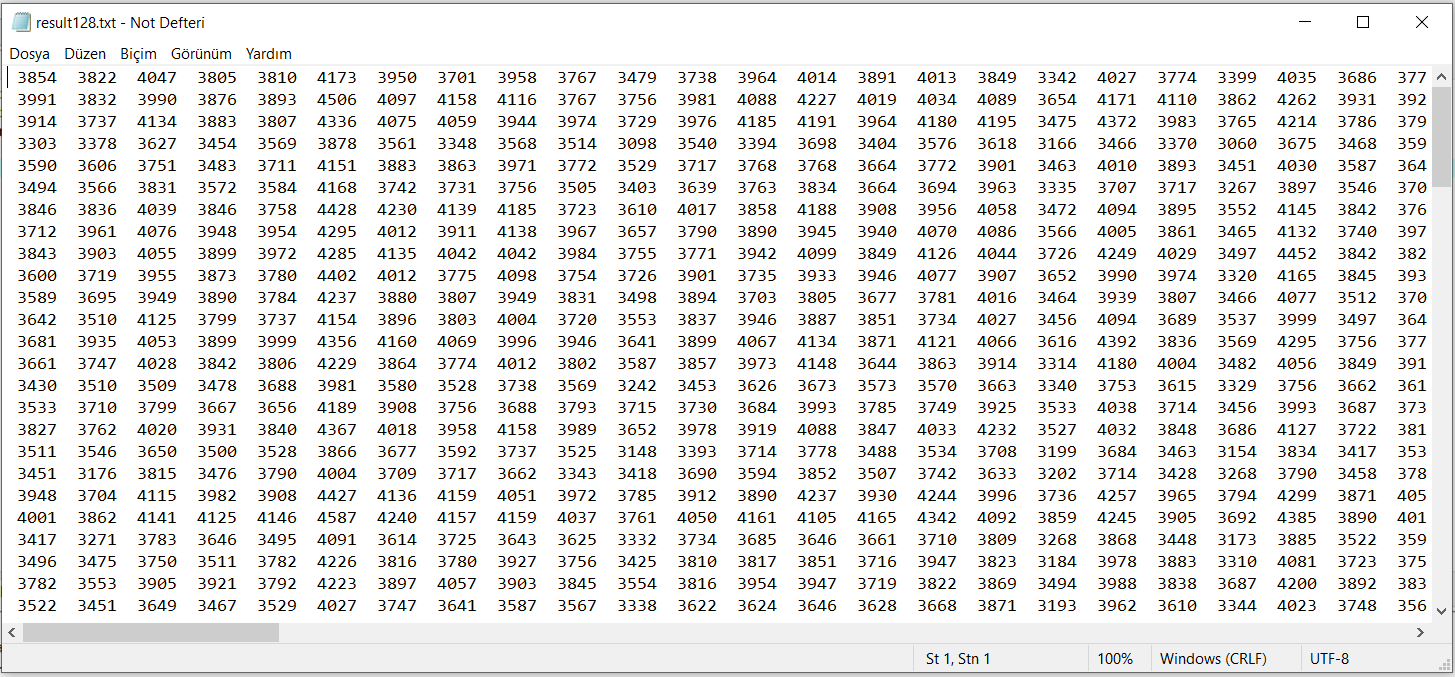
**a short section from a128.txt**

**b128.txt file**

****

**a short section from b128.txt**

**Result128.txt file**

****

**a short section from result128.txt**

**Question 2)**

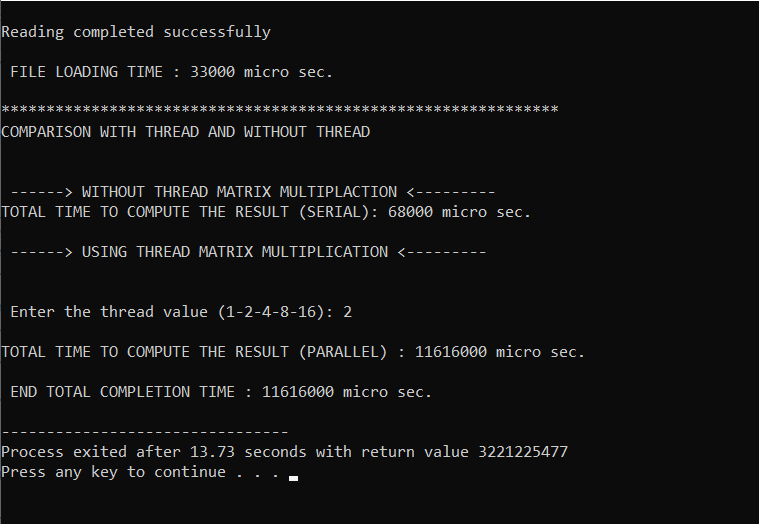
**Code :**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <pthread.h>**  **#include <time.h>**  **typedef struct MATRIX**  **{**  **int \*\*data;**  **int rows;**  **int columns;**  **double elapsedTime;**  **} MATRIX;**  **typedef struct THREAD\_DATAS**  **{**  **MATRIX M,N;**  **MATRIX \*result;// = (MATRIX \*) malloc(sizeof(MATRIX));**  **int row;**  **int column;**  **} THREAD\_DATAS;**  **MATRIX\* read\_matrices();**  **int\*\* fillMatrix(FILE \*f, int rows, int columns);**  **//INPUT OUTPUT METHOD**  **void write\_output(MATRIX \*matrices);**  **void print\_matrix(MATRIX matrix, char name);**  **void serial\_matrix\_multiplication(MATRIX M, MATRIX N, MATRIX \*C);**  **// Element by element threading**  **void element\_by\_element(MATRIX M, MATRIX N, MATRIX \*C);**  **void\* evaluate\_element\_thread(void \*ptr);**  **// Element evaluation**  **void calculate\_element(MATRIX M, MATRIX N, MATRIX\* C, int row, int column);**    **MATRIX\* read\_matrices()**  **{**  **MATRIX \*matrices = (MATRIX \*) malloc(2 \* sizeof(MATRIX));**    **FILE \*file = fopen("b256.txt", "r");**  **if(file != NULL)**  **{**  **fscanf(file, "%d %d\n", &matrices[0].rows, &matrices[0].columns);**  **// FILL MATRICE**  **matrices[0].data = fillMatrix(file, matrices[0].rows, matrices[0].columns);**  **fscanf(file, "%d %d\n", &matrices[1].rows, &matrices[1].columns);**  **matrices[1].data = fillMatrix(file, matrices[1].rows, matrices[1].columns);**    **fclose(file);**  **return matrices;**  **printf("Dosya Var reis");**  **}**  **else**  **{**  **printf("FILE NOT FOUND ! PLEASE TRY AGAIN ");**  **exit(0);**  **}**    **}**  **int\*\* fillMatrix(FILE \*file, int rows, int columns)**  **{**  **int \*\*matrix = (int\*\*) malloc(rows \* sizeof(int \*));**  **int \*newRow;**  **int i,j;**  **for (i = 0; i < rows; i++)**  **{**  **newRow = (int\*) malloc(columns \* sizeof(int));**  **for(j = 0; j < (columns-1) ; j++)**  **{**  **fscanf(file, "%d ", &newRow[j]);**  **}**  **fscanf(file, "%d\n", &newRow[j]);**  **matrix[i] = newRow;**  **}**  **return matrix;**  **}**  **void write\_output(MATRIX \*matrices)**  **{**  **FILE \*f = fopen("resultparallel.txt", "w");**  **int count, row, column;**  **for (count = 0; count < 3; count++)**  **{**  **fprintf(f, "%d\n", (matrices[count].rows \* matrices[count].columns));**  **for (row = 0; row < matrices[count].rows; row++)**  **{**  **for (column = 0; column < matrices[count].columns; column++)**  **fprintf(f, "%d ", matrices[count].data[row][column]);**  **fprintf(f, "\n");**  **}**  **fprintf(f, "Elapsed time: %.0lf micro sec.\n", matrices[count].elapsedTime);**  **}**  **fclose(f);**  **}**  **void print\_matrix(MATRIX matrix, char name)**  **{**  **int rows = matrix.rows;**  **int columns = matrix.columns;**  **printf("matrix %c (size: %d x %d)\n", name, rows, columns);**  **int i, j;**  **for (i = 0; i < rows; i++)**  **{**  **for (j = 0; j < columns; j++)**  **printf("%d\t", matrix.data[i][j]);**  **printf("\n");**  **}**  **return;**  **}**  **void serial\_matrix\_multiplication(MATRIX M, MATRIX N, MATRIX \*C)**  **{**  **(\*C).rows = M.rows;**  **(\*C).columns = N.columns;**  **(\*C).data = (int\*\*) malloc(M.rows \* sizeof(int\*));**  **int x, y;**  **for (x = 0; x < M.rows; x++)**  **{**  **(\*C).data[x] = (int\*) malloc(N.columns \* sizeof(int));**  **for (y = 0; y < N.columns; y++)**  **{**  **calculate\_element(M, N, C, x, y);**  **}**  **}**  **return;**  **}**  **void element\_by\_element(MATRIX M, MATRIX N, MATRIX \*C)**  **{**  **(\*C).rows = M.rows;**  **(\*C).columns = N.columns;**  **(\*C).data = (int\*\*) malloc(M.rows \* sizeof(int\*));**  **int x, y;**  **int z=0;**  **printf("\n Enter the thread value (1-2-4-8-16): ");**  **scanf("%d",&z);**    **int threads\_number = M.rows \* N.columns;**  **pthread\_t threads[threads\_number];**  **for (x = 0; x < M.rows; x++)**  **{**  **(\*C).data[x] = (int\*) malloc(N.columns \* sizeof(int));**  **for (y = 0; y < N.columns; y++)**  **{**    **// create & initialize thread struct**  **THREAD\_DATAS \*args = (THREAD\_DATAS\*) malloc(sizeof(THREAD\_DATAS));**  **(\*args).M = M; (\*args).N = N; (\*args).result = C; (\*args).row = x; (\*args).column = y;**  **// create thread**  **pthread\_create(&threads[z], NULL, evaluate\_element\_thread, (void\*) args);**  **z++;**  **}**  **}**  **for(z = 0; z < threads\_number; z++)**  **{**  **pthread\_join(threads[z], NULL);**  **}**  **return;**  **}**  **void\* evaluate\_element\_thread(void \*ptr)**  **{**  **THREAD\_DATAS data = \*((THREAD\_DATAS \*) ptr);**    **(\*data.result).data[data.row][data.column] = 0;**  **int i;**  **for (i = 0; i < data.M.columns; i++)**  **{**  **(\*(data.result)).data[data.row][data.column] += data.M.data[data.row][i] \* data.N.data[i][data.column];**  **}**  **return NULL;**  **}**  **// Element evaluation**  **void calculate\_element(MATRIX M, MATRIX N, MATRIX\* C, int row, int column)**  **{**  **(\*C).data[row][column] = 0;**  **int i;**  **for (i = 0; i < M.columns; i++)**  **{**  **(\*C).data[row][column] += M.data[row][i] \* N.data[i][column];**  **}**  **return;**  **}**  **int main()**  **{**  **clock\_t startTime, endTime,total\_completion ,end\_total\_completion\_time= 0;**  **double elapsed\_time = 0;**  **end\_total\_completion\_time= clock();**  **startTime = clock();**  **MATRIX \*matrices = read\_matrices();**  **if(matrices[0].columns != matrices[1].rows)**  **{**  **printf("OOPPSS!!! ROWS AND COLUMNS DO NOT MATCH WITH EACH OTHER\n");**  **exit(0);**  **}**    **printf("\nReading completed successfully\n");**  **endTime = clock();**    **elapsed\_time = 1000000.0 \* ((double) (endTime - startTime)) / CLOCKS\_PER\_SEC;**  **printf("\n FILE LOADING TIME : %.0lf micro sec.\n", elapsed\_time);**  **printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**  **// RESULTS MATRICES**  **MATRIX\* result = (MATRIX \*)malloc(3 \* sizeof(MATRIX));**  **startTime, endTime = 0;**  **elapsed\_time = 0;**    **// SERIAL MATRIX**  **printf("COMPARISON WITH THREAD AND WITHOUT THREAD \n\n");**  **printf("\n ------> WITHOUT THREAD MATRIX MULTIPLACTION <--------- \n");**    **startTime = clock();**  **serial\_matrix\_multiplication(matrices[0], matrices[1], &result[0]);**  **endTime = clock();**  **elapsed\_time = 1000000.0 \* ((double) (endTime - startTime)) / CLOCKS\_PER\_SEC;**  **result[0].elapsedTime = elapsed\_time;**      **printf("TOTAL TIME TO COMPUTE THE RESULT (SERIAL): %.0lf micro sec.\n", elapsed\_time);**  **// threading each for each element in the resulting matrix**    **printf("\n ------> USING THREAD MATRIX MULTIPLICATION <--------- \n\n");**  **startTime = clock();**  **element\_by\_element(matrices[0], matrices[1], &result[1]);**  **endTime = clock();**  **elapsed\_time = 1000000.0 \* ((double) (endTime - startTime)) / CLOCKS\_PER\_SEC;**  **result[1].elapsedTime = elapsed\_time;**    **printf("\nTOTAL TIME TO COMPUTE THE RESULT (PARALLEL) : %.0lf micro sec.\n", elapsed\_time);**    **end\_total\_completion\_time = clock();**  **elapsed\_time = 1000000.0 \* ((double) (endTime - startTime)) / CLOCKS\_PER\_SEC;**  **printf("\n END TOTAL COMPLETION TIME : %.0lf micro sec.\n", elapsed\_time);**    **write\_output(result);**    **return 0;**  **}** |

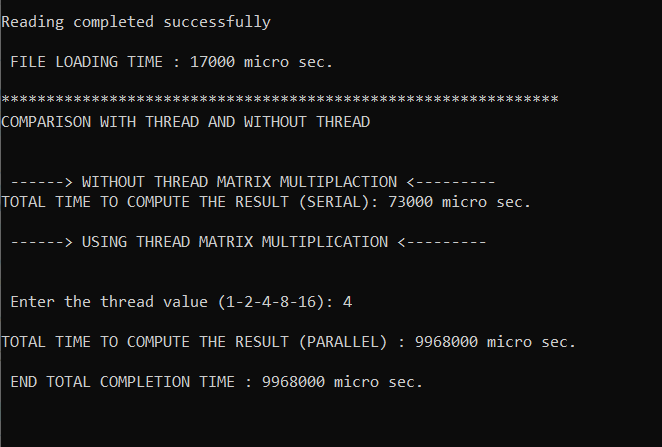
**Example Outputs ;**



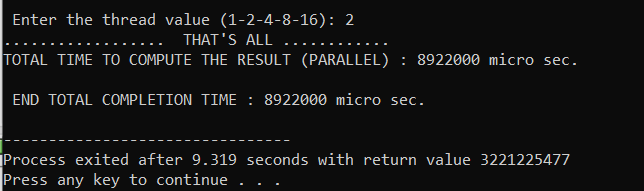
**256x256 -> 2 Thread;**

****

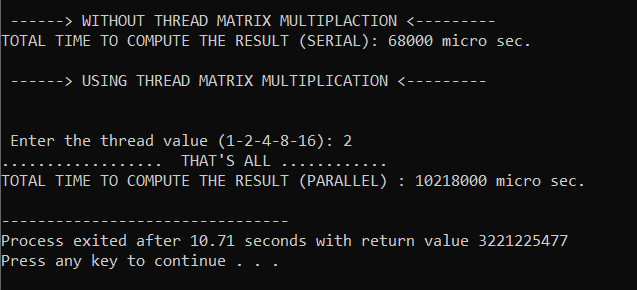
**256x256 -> 4 Thread;**

****

**Compute Time :**

****

* 1. **Total Time To Compute The Result**

****

**Question3 )**

Hardware Details :

**RAM:**

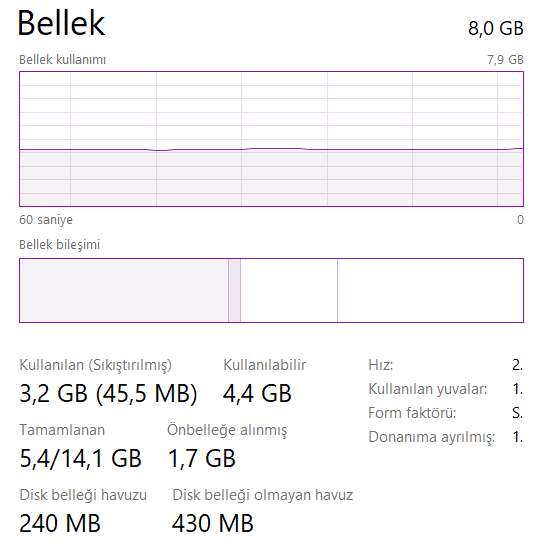
Total Physical Memory: 8.063 MB

Available Physical Memory: 3.377 MB

Virtual Memory: Max Size: 14.463 MB

Virtual Memory: Available: 7.514 MB

Virtual Memory: In Use: 6.949 MB

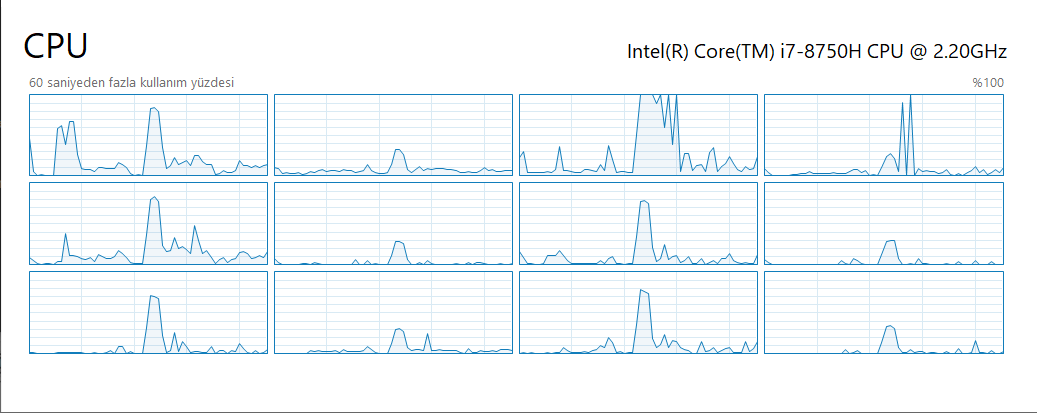


**Disk Details :**

SSD 450 GB

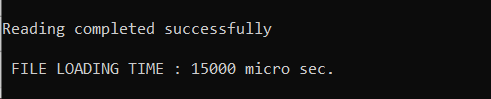
**CPU:**

Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz

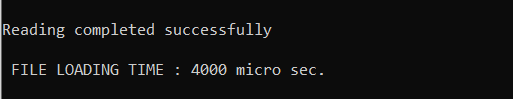
****

**File Loading Time :**

128 x 128

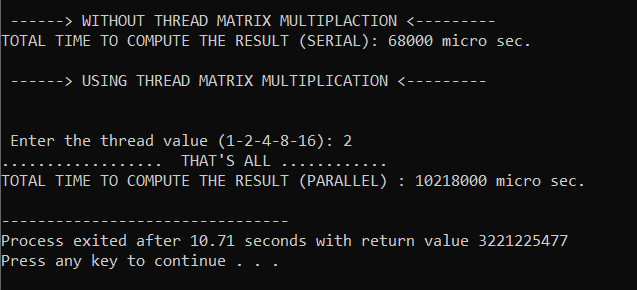


256 x 256

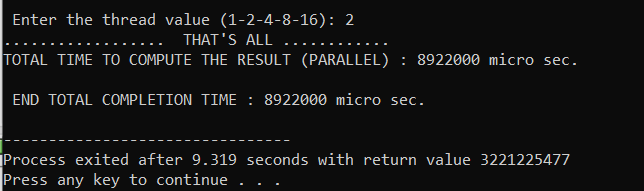


Compute Time :

128 x 128

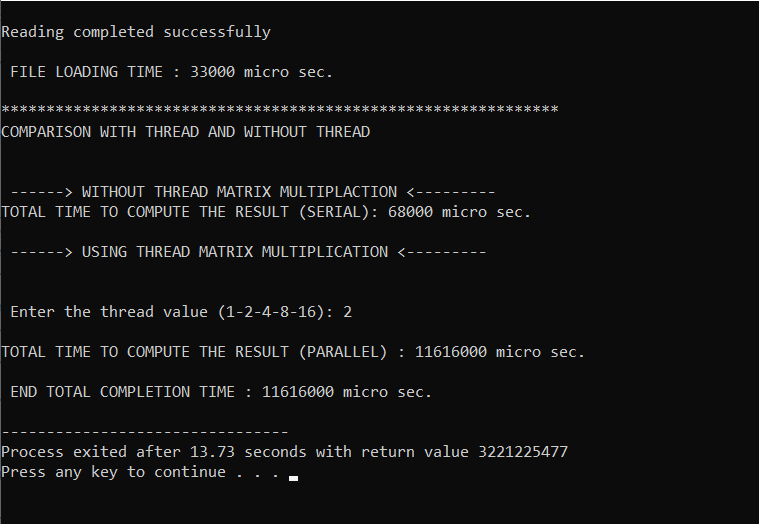


Total completion time to do the job ;

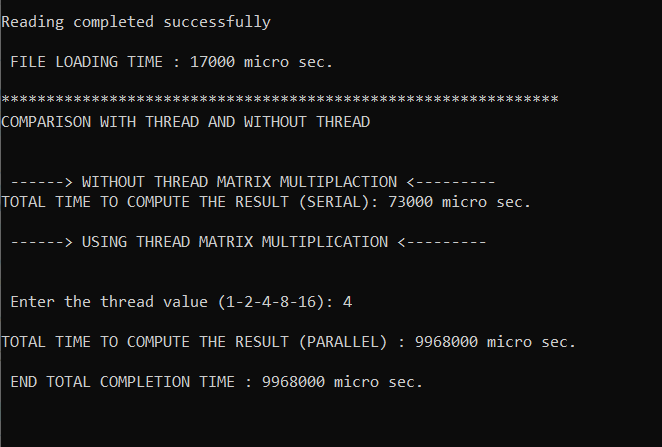


Comparison Result:

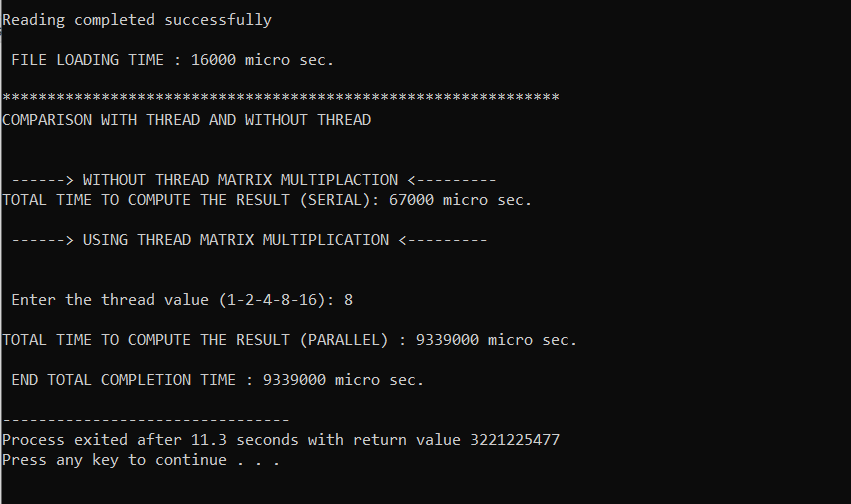
**256 x 256 -> 2 Thread:**



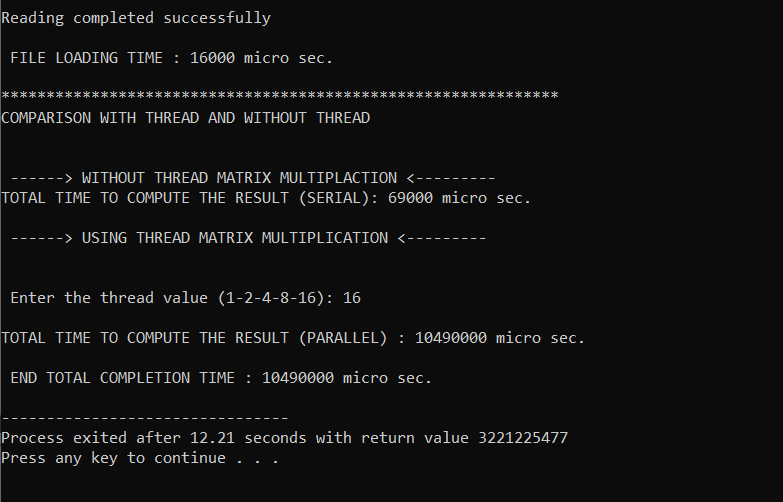
256x256 -> 4 Thread;



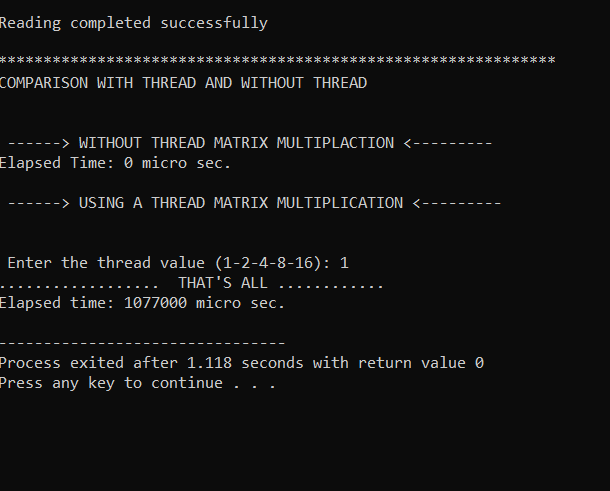
256x256 -> 8 Thread;



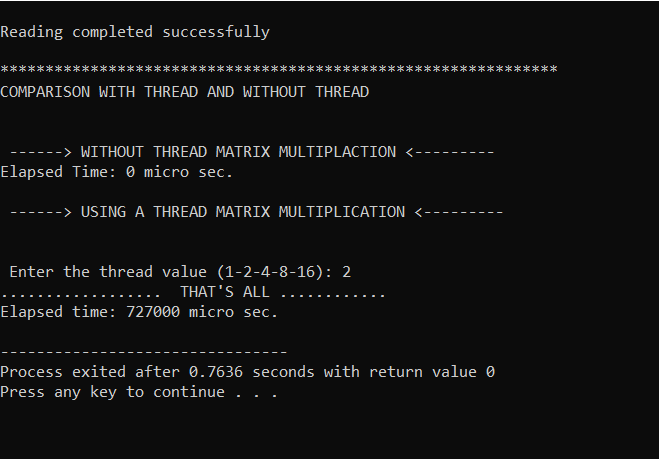
256 x 256 -> 16 Thread;



**1024x1024 -> 1 Thread;**



**1024x1024 -> 2 Thread;**



**1024x1024 -> 4 Thread;**

