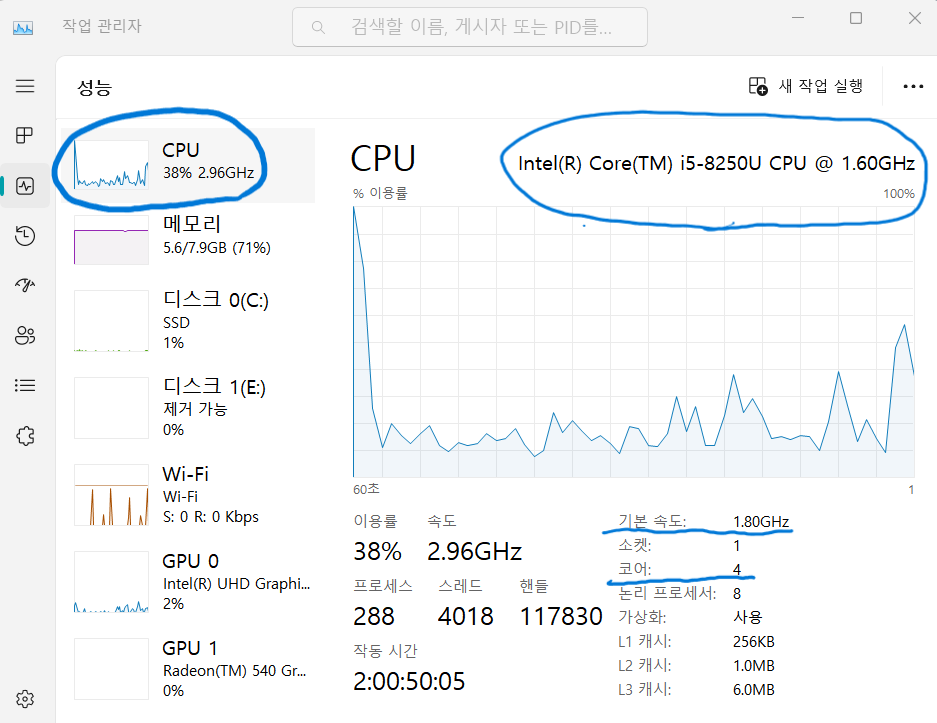
|  |  |
| --- | --- |
|  |  |
| Problem1 Report |  |
|  |  |
|  | Student NO: 20183784Student Name: 노현진 |
|  |  |

**[Environment]**

* **CPU**

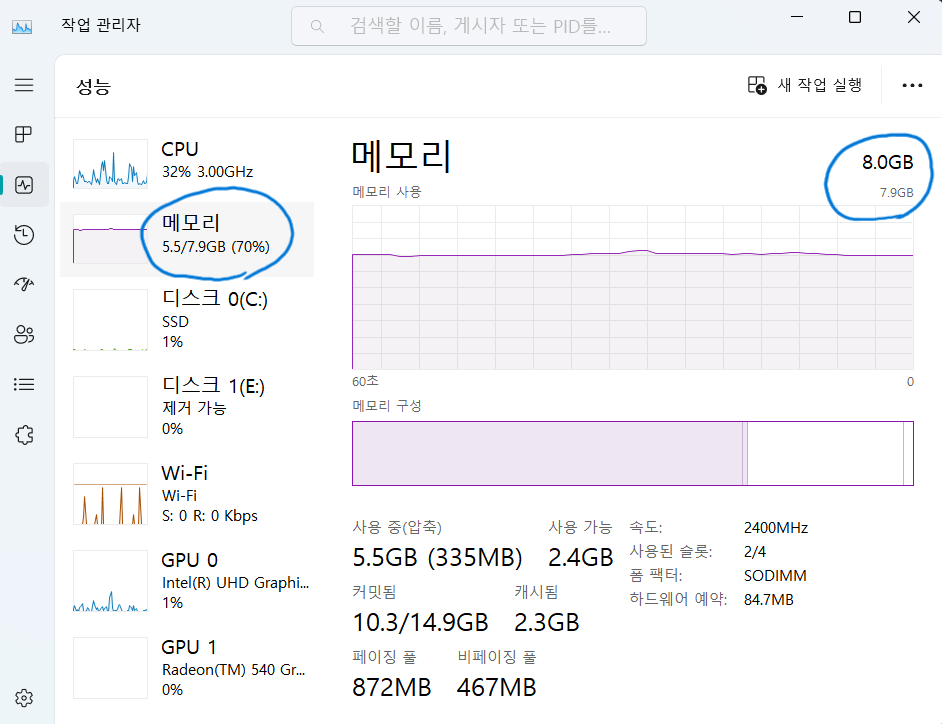


**CPU type: Intel® Core™ i5-8250U CPU**

**Clock Speed: 1.80GHz**

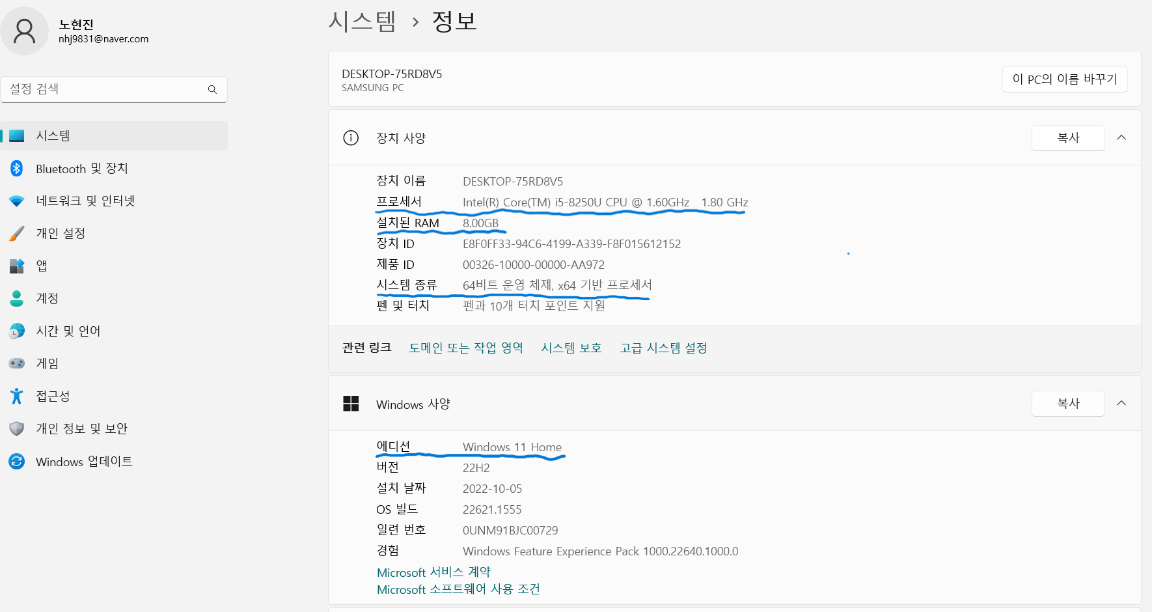
**Number of cores: 4**

* **Memory**



**Memory size: 8.0GB**

* **OS**



**OS type: Windows 11**

**[Source Code]**

* **pc\_static\_block.java**

package com.hyunjin.study.problem1;  
  
*/\*  
 This program should print the following values:  
 (1) execution time of each thread  
 (2) program execution time  
 (3) the number of 'prime numbers'  
\*/*public final class pc\_static\_block {  
 private static int *NUM\_THREADS* = 1; *// default number of threads: 1* private static int *NUM\_END* = 200000; *// default input: 200000* public static void main(String[] args) {  
 if (args.length == 2) {  
 *NUM\_THREADS* = Integer.*parseInt*(args[0]); *// Possible number of threads: 1, 2, 4, 6, 8, 10, 12, 14, 16, 32  
 NUM\_END* = Integer.*parseInt*(args[1]);  
 }  
  
 BlockThread[] blockThreads = new BlockThread[*NUM\_THREADS*];  
 int[] results = new int[*NUM\_THREADS*];  
  
 int workUnit = *NUM\_END* / *NUM\_THREADS*;  
 for (int i = 1; i <= *NUM\_THREADS*; i++) {  
 int startNum = workUnit \* (i - 1) + 1;  
 int endNum;  
 if (i == *NUM\_THREADS*) {  
 endNum = *NUM\_END*;  
 } else {  
 endNum = workUnit \* i;  
 }  
 blockThreads[i - 1] = new BlockThread(startNum, endNum, results, i - 1);  
 }  
  
 long startTime = System.*currentTimeMillis*();  
  
 for (BlockThread blockThread : blockThreads) {  
 blockThread.start();  
 }  
  
 try {  
 for (BlockThread blockThread : blockThreads) {  
 blockThread.join();  
 }  
 } catch (Exception ignored) {}  
  
 int answer = 0;  
 for (int result : results) {  
 answer += result;  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
 System.*out*.println("\nProgram Execution Time: " + timeDiff + "ms");  
 System.*out*.println("1 ~ " + *NUM\_END* + ", the number of 'prime numbers': " + answer);  
 }  
}  
  
final class BlockThread extends Thread {  
 private final int startNum;  
 private final int endNum;  
 private final int[] results;  
 private final int index;  
  
 public BlockThread(int startNum, int endNum, int[] results, int index) {  
 this.startNum = startNum;  
 this.endNum = endNum;  
 this.results = results;  
 this.index = index;  
 }  
  
 @Override  
 public void run() {  
 long startTime = System.*currentTimeMillis*();  
  
 for (int i = startNum; i <= endNum; i++) {  
 if (isPrime(i)) {  
 results[index] += 1;  
 }  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
  
 System.*out*.println(this.getName() + " ==> " + startNum + " ~ " + endNum + ", Execution Time: " + timeDiff + "ms");  
 }  
  
 private boolean isPrime(int x) {  
 if (x <= 1) {  
 return false;  
 }  
 for (int i = 2; i < x; i++) {  
 if (x % i == 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

* **pc\_static\_cyclic.java**

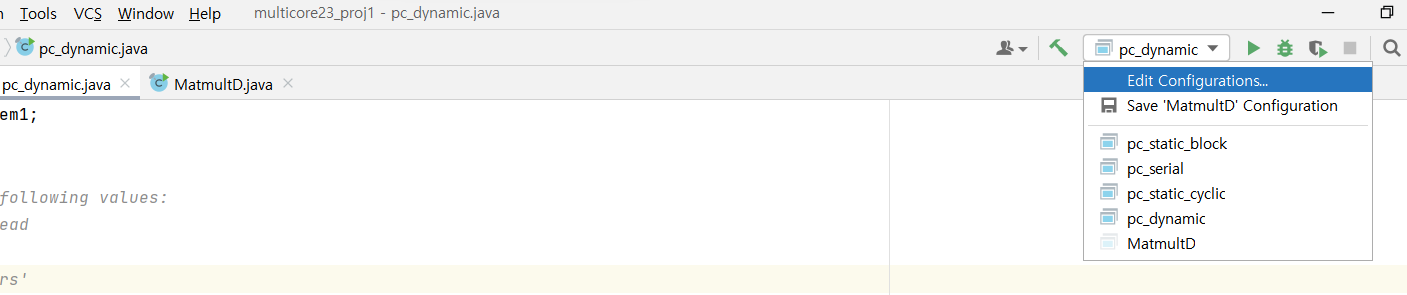
package com.hyunjin.study.problem1;  
  
*/\*  
 This program should print the following values:  
 (1) execution time of each thread  
 (2) program execution time  
 (3) the number of 'prime numbers'  
\*/*public final class pc\_static\_cyclic {  
 private static int *NUM\_THREADS* = 1; *// default number of threads* private static int *NUM\_END* = 200000; *// default input* public static void main(String[] args) {  
 if (args.length == 2) {  
 *NUM\_THREADS* = Integer.*parseInt*(args[0]); *// Possible number of threads: 1, 2, 4, 6, 8, 10, 12, 14, 16, 32  
 NUM\_END* = Integer.*parseInt*(args[1]);  
 }  
  
 CyclicThread[] cyclicThreads = new CyclicThread[*NUM\_THREADS*];  
 int[] results = new int[*NUM\_THREADS*];  
  
 int workUnit = *NUM\_THREADS* \* 10;  
 int endNum = *NUM\_END*;  
 for (int i = 1; i <= *NUM\_THREADS*; i++) {  
 int startNum = 10 \* (i - 1) + 1;  
 cyclicThreads[i - 1] = new CyclicThread(startNum, endNum, workUnit, results, i - 1);  
 }  
  
 long startTime = System.*currentTimeMillis*();  
  
 for (CyclicThread cyclicThread : cyclicThreads) {  
 cyclicThread.start();  
 }  
  
 try {  
 for (CyclicThread cyclicThread : cyclicThreads) {  
 cyclicThread.join();  
 }  
 } catch (Exception ignored) {}  
  
 int answer = 0;  
 for (int result : results) {  
 answer += result;  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
 System.*out*.println("\nProgram Execution Time: " + timeDiff + "ms");  
 System.*out*.println("1 ~ " + *NUM\_END* + ", the number of 'prime numbers': " + answer);  
 }  
}  
  
final class CyclicThread extends Thread {  
 private final int startNum;  
 private final int endNum;  
 private final int workUnit;  
 private final int[] results;  
 private final int index;  
  
 public CyclicThread(int startNum, int endNum, int workUnit, int[] results, int index) {  
 this.startNum = startNum;  
 this.endNum = endNum;  
 this.workUnit = workUnit;  
 this.results = results;  
 this.index = index;  
 }  
  
 @Override  
 public void run() {  
 long startTime = System.*currentTimeMillis*();  
  
 for (int i = startNum; i <= endNum; i += workUnit) {  
 for (int j = 0; j < 10; j++) {  
 if (isPrime(i + j)) {  
 results[index] += 1;  
 }  
 }  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
 System.*out*.println(this.getName() + " ==> Execution Time: " + timeDiff + "ms");  
 }  
  
 private boolean isPrime(int x) {  
 if (x <= 1) {  
 return false;  
 }  
 for (int i = 2; i < x; i++) {  
 if (x % i == 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

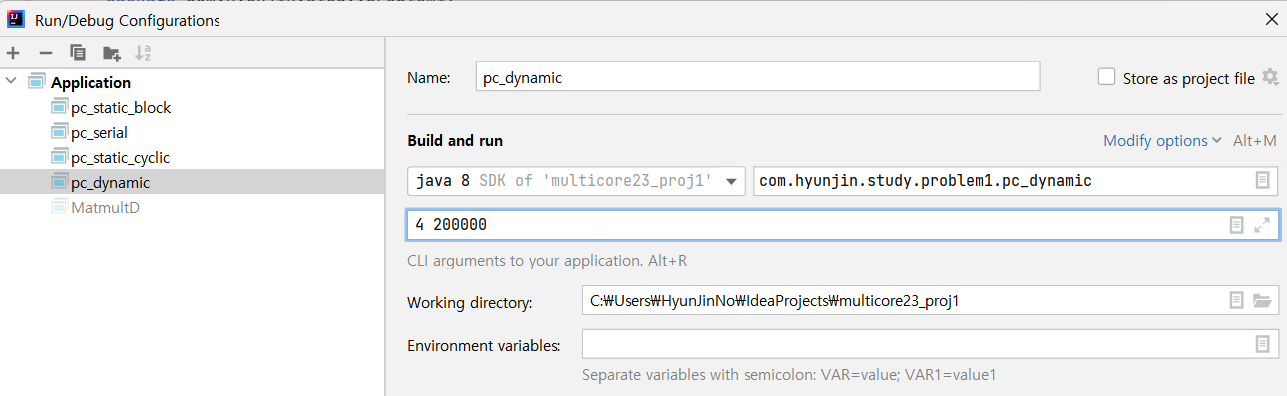
* **pc\_dynamic.java**

package com.hyunjin.study.problem1;  
  
*/\*  
 This program should print the following values:  
 (1) execution time of each thread  
 (2) program execution time  
 (3) the number of 'prime numbers'  
\*/*public final class pc\_dynamic {  
 private static int *NUM\_THREADS* = 1; *// default number of threads* private static int *NUM\_END* = 200000; *// default input* private static int *NUM\_START* = 1;  
  
 public static void main(String[] args) {  
 if (args.length == 2) {  
 *NUM\_THREADS* = Integer.*parseInt*(args[0]); *// Possible number of threads: 1, 2, 4, 6, 8, 10, 12, 14, 16, 32  
 NUM\_END* = Integer.*parseInt*(args[1]);  
 }  
 DynamicThread[] dynamicThreads = new DynamicThread[*NUM\_THREADS*];  
 int[] results = new int[*NUM\_THREADS*];  
  
 for (int i = 1; i <= *NUM\_THREADS*; i++) {  
 dynamicThreads[i - 1] = new DynamicThread(results, i - 1);  
 }  
  
 long startTime = System.*currentTimeMillis*();  
  
 for (DynamicThread dynamicThread : dynamicThreads) {  
 dynamicThread.start();  
 }  
  
 try {  
 for (DynamicThread dynamicThread : dynamicThreads) {  
 dynamicThread.join();  
 }  
 } catch (Exception ignored) {}  
  
 int answer = 0;  
 for (int result : results) {  
 answer += result;  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
 System.*out*.println("\nProgram Execution Time: " + timeDiff + "ms");  
 System.*out*.println("1 ~ " + *NUM\_END* + ", the number of 'prime numbers': " + answer);  
 }  
  
 public synchronized static int getWork() {  
 if (*NUM\_START* >= *NUM\_END*) {  
 return -1;  
 } else {  
 *NUM\_START* += 10;  
 return (*NUM\_START* - 10);  
 }  
 }  
}  
  
final class DynamicThread extends Thread {  
 private final int[] results;  
 private final int index;  
  
 public DynamicThread(int[] results, int index) {  
 this.results = results;  
 this.index = index;  
 }  
  
 @Override  
 public void run() {  
 long startTime = System.*currentTimeMillis*();  
  
 while (true) {  
 int startNum = pc\_dynamic.*getWork*();  
 if (startNum == -1) {  
 break;  
 }  
 for (int i = startNum; i < startNum + 10; i++) {  
 if (isPrime(i)) {  
 results[index] += 1;  
 }  
 }  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
 System.*out*.println(this.getName() + " ==> Execution Time: " + timeDiff + "ms");  
 }  
  
 private boolean isPrime(int x) {  
 if (x <= 1) {  
 return false;  
 }  
 for (int i = 2; i < x; i++) {  
 if (x % i == 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

* **How to compile and execute**

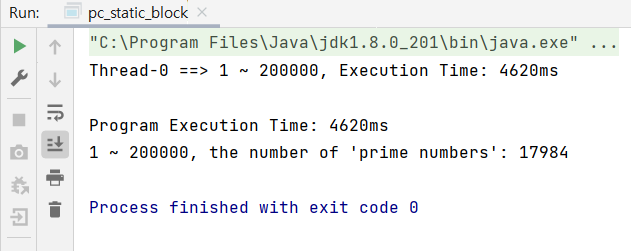
1. **Firstly, install Intellij IDEA.**
2. **After installation, open the submitted file.**
3. **Before execution, click “Edit Configurations..”**

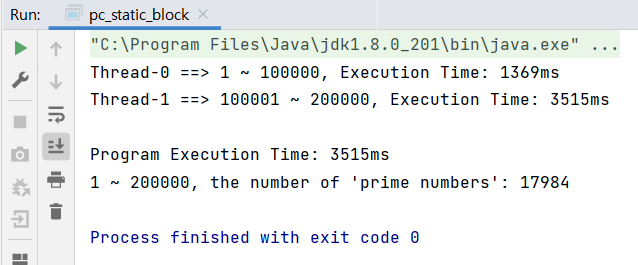
****

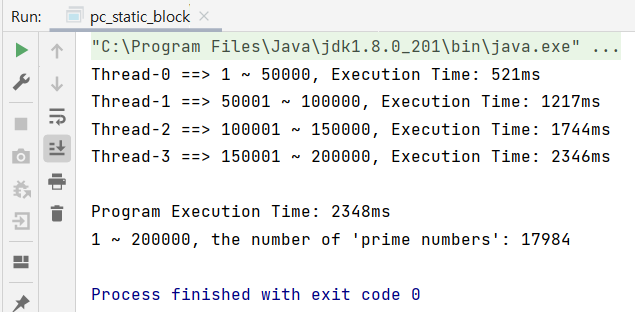
1. **Set the arguments. In this picture, “4” means NUM\_THREADS, and “200000” means NUM\_END.**
2. **Finally, run the source code.**

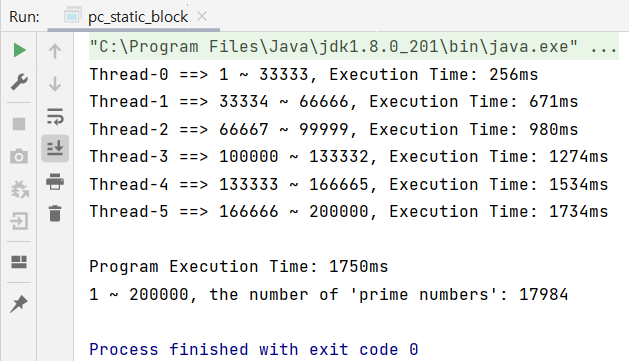
**[Results]**

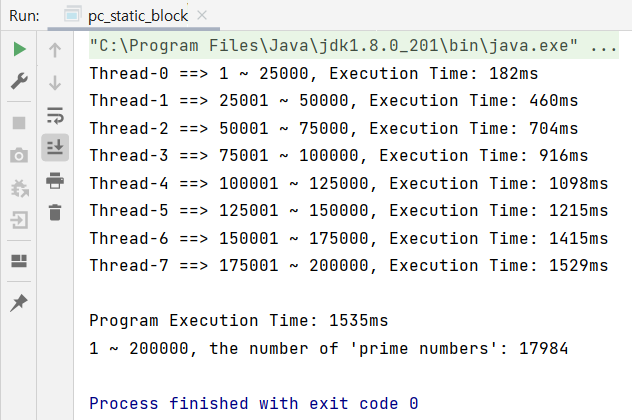
* **Static (Block)**

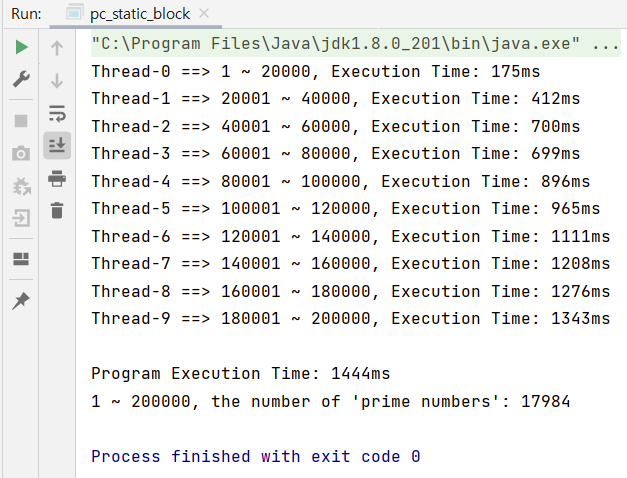


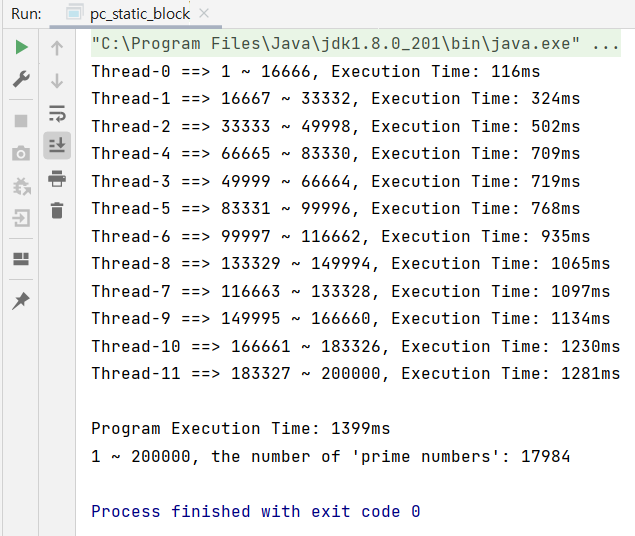


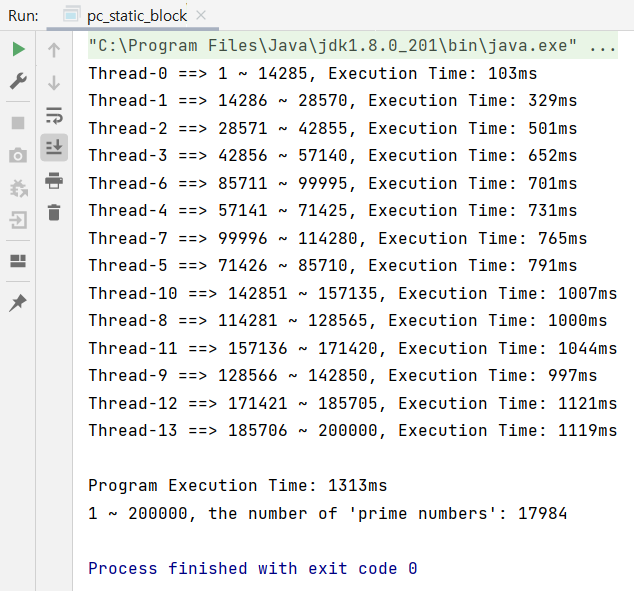


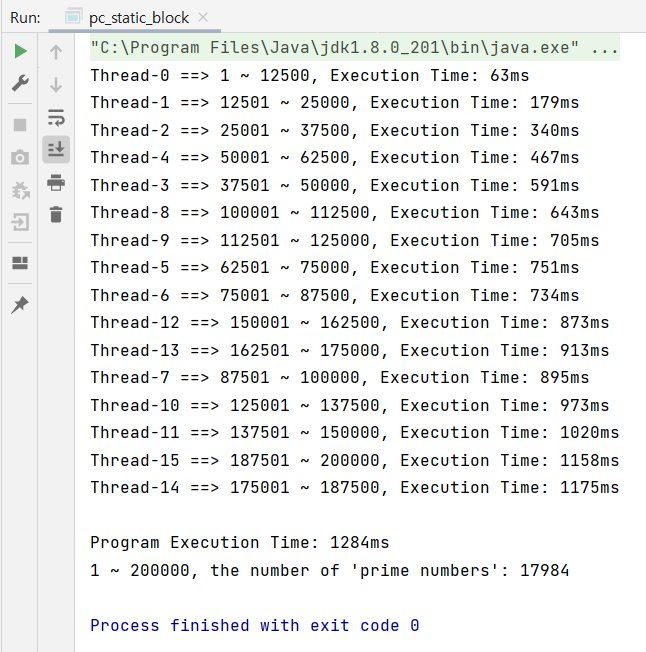


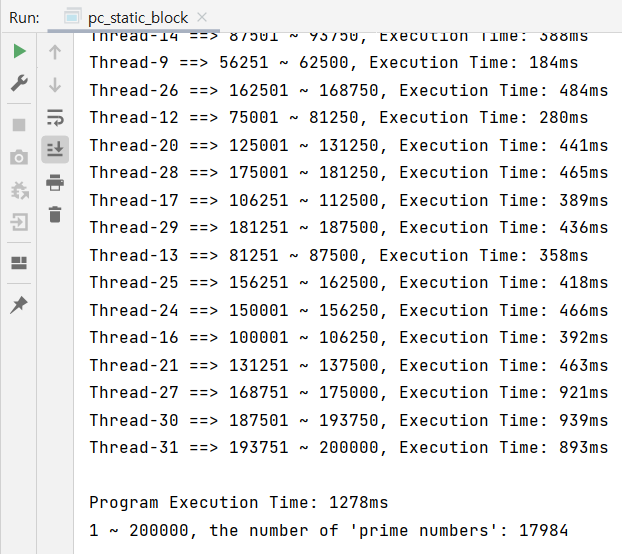




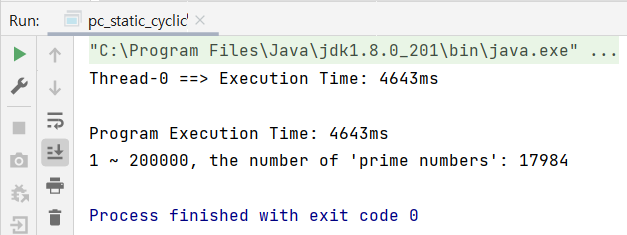


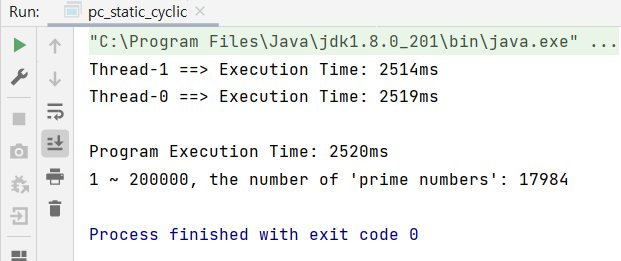


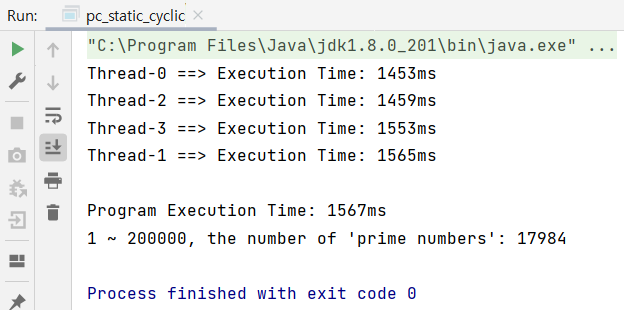


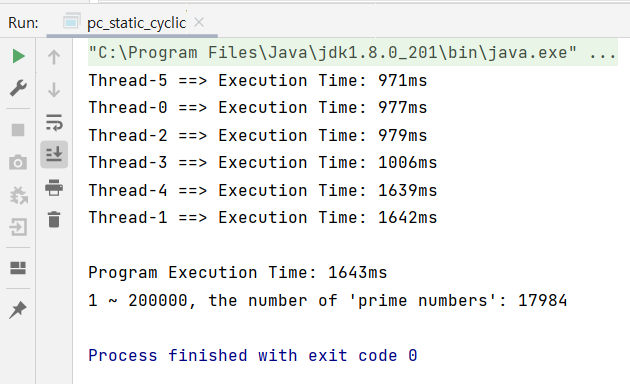


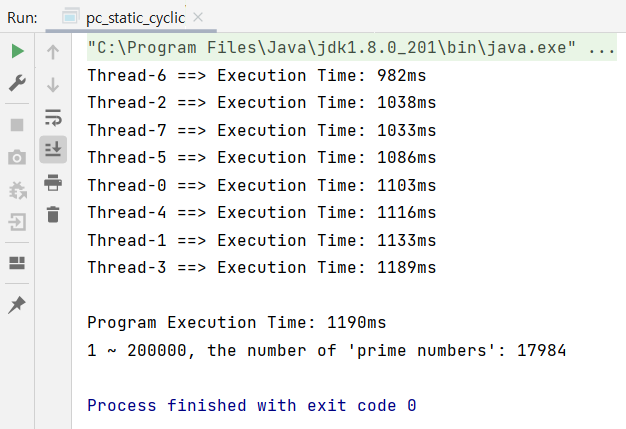
* **Static (Cyclic)**

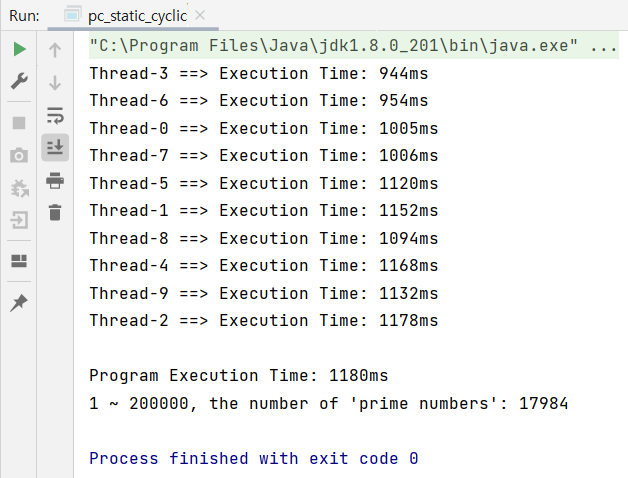


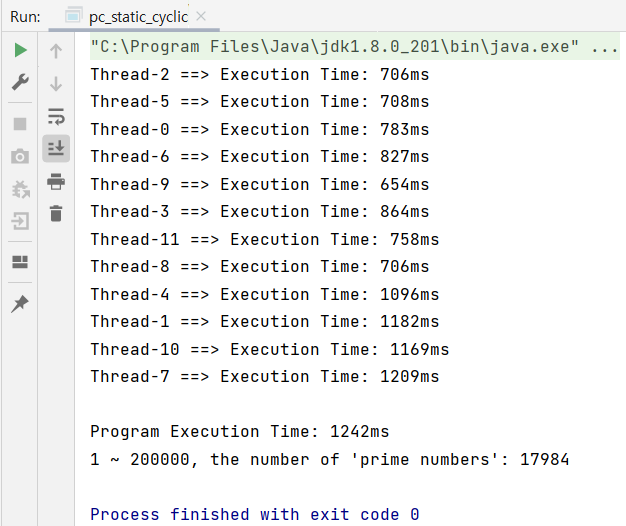


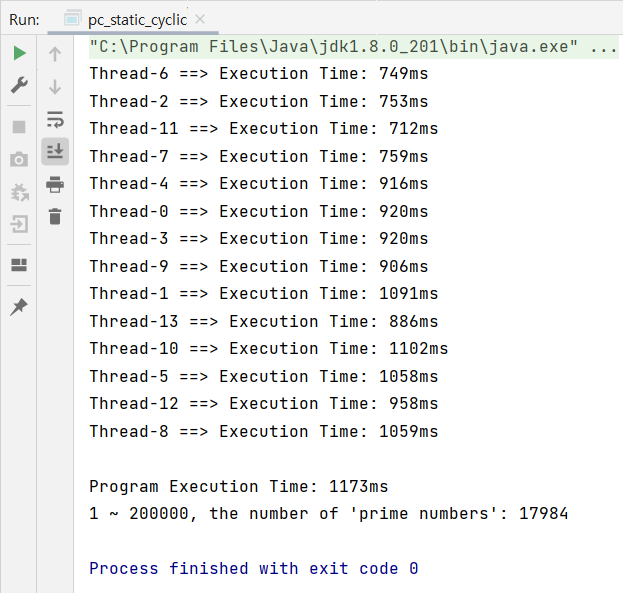


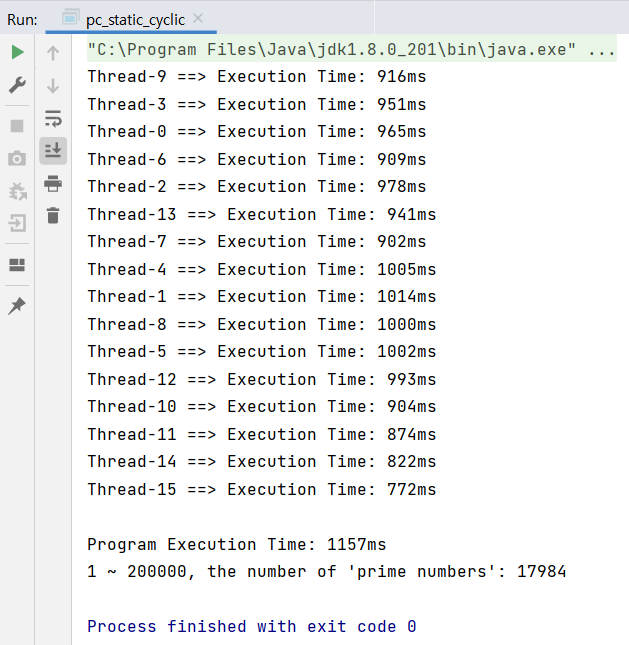


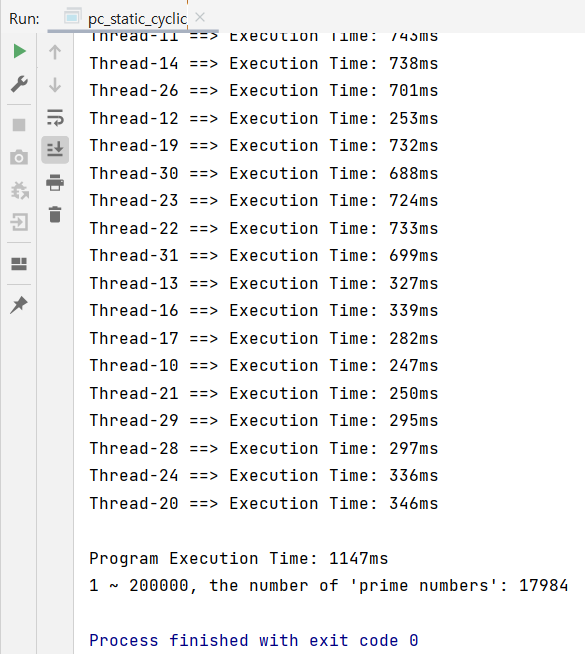




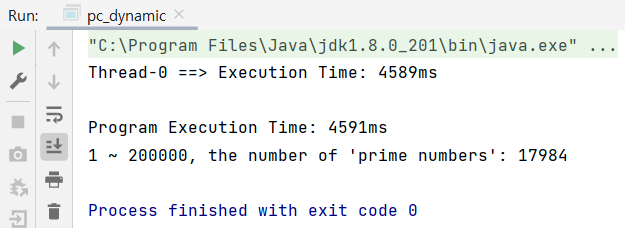


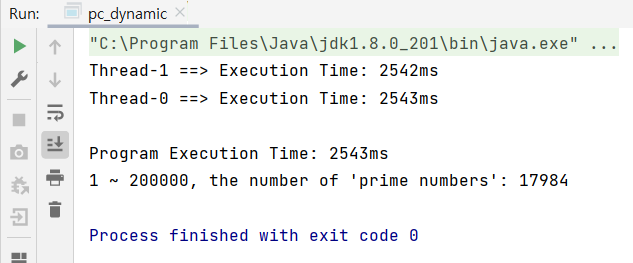


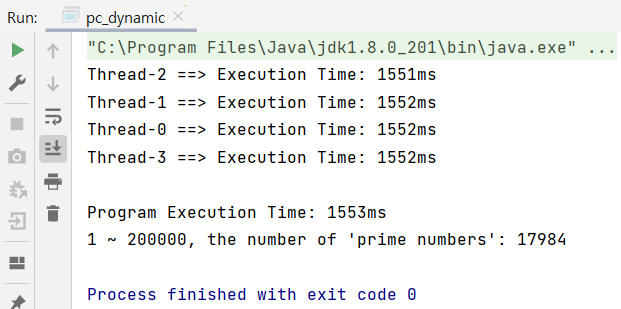


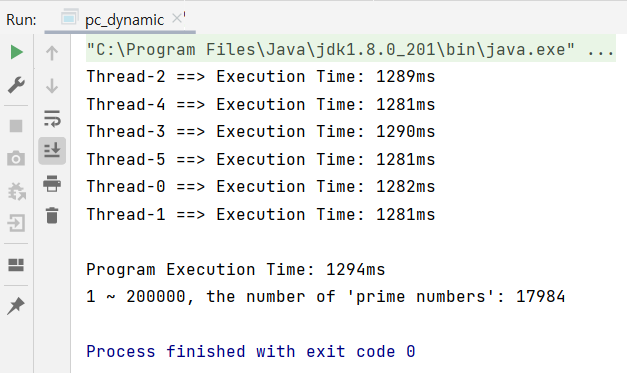


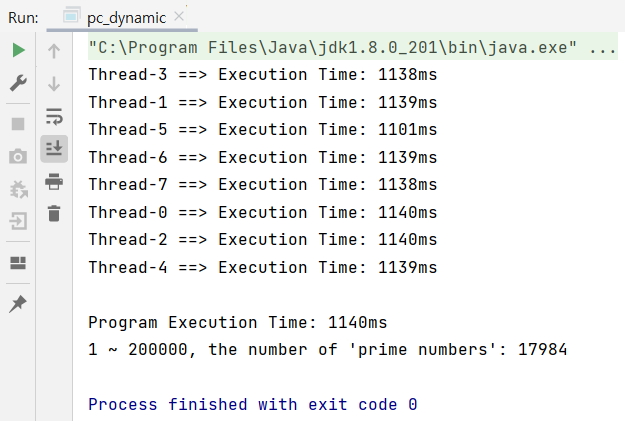
* **Dynamic**

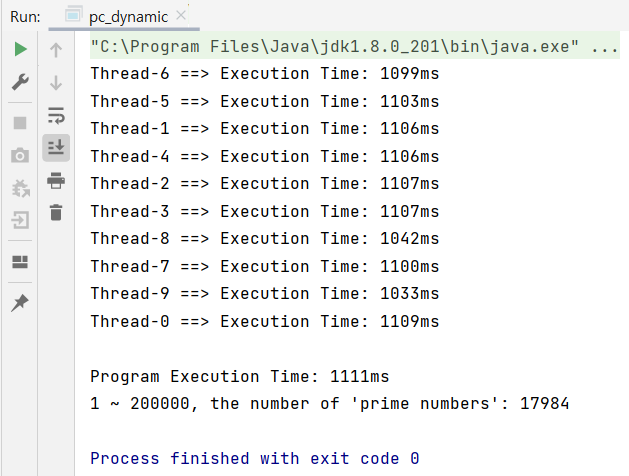


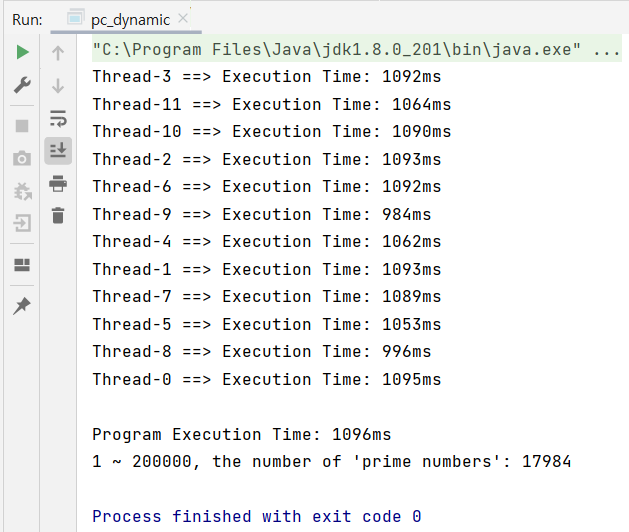


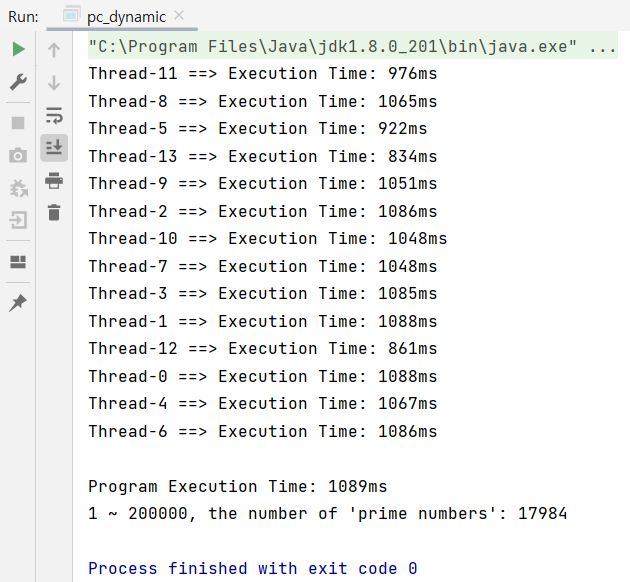


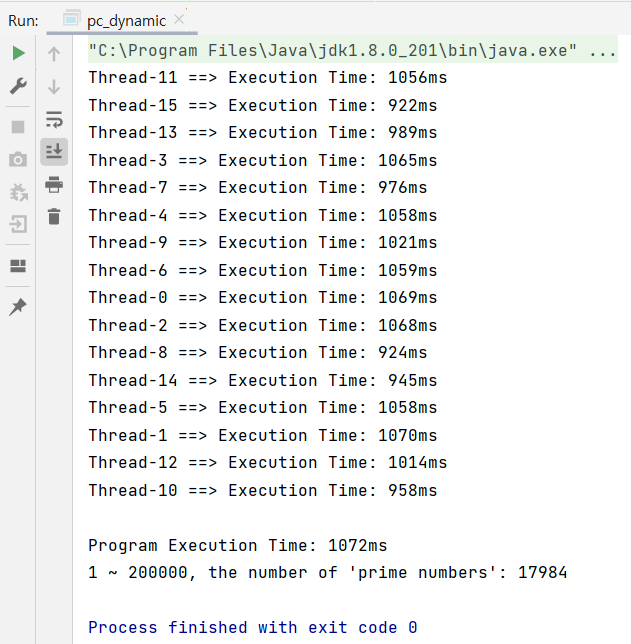


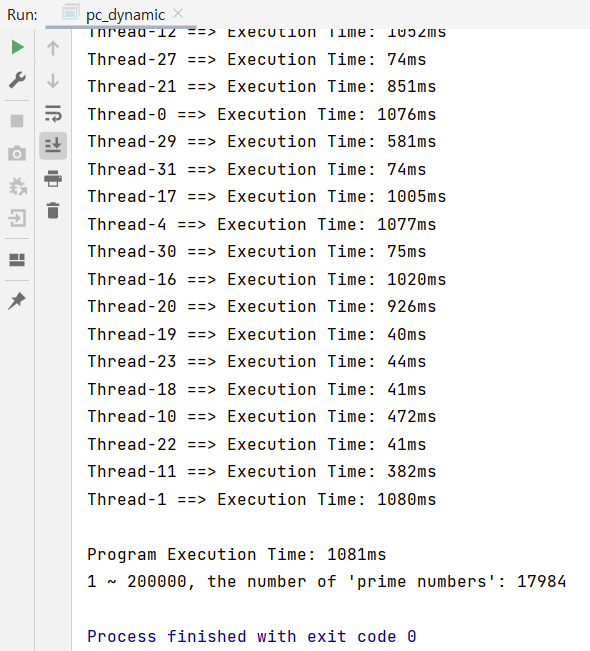












* **Execution Time**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exec time | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| Static (Block) | 4620ms | 3515ms | 2348ms | 1750ms | 1535ms | 1444ms | 1399ms | 1313ms | 1284ms | 1278ms |
| Static (Cyclic)  [task size: 10 numbers] | 4643ms | 2520ms | 1567ms | 1643ms | 1190ms | 1180ms | 1242ms | 1173ms | 1157ms | 1147ms |
| Dynamic  [task size: 10 numbers] | 4591ms | 2543ms | 1553ms | 1294ms | 1140ms | 1111ms | 1096ms | 1089ms | 1072ms | 1081ms |

* **Performance**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Performance  (1/exec time) | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| Static (Block) | 2.16e-4 | 2.84e-4 | 4.26e-4 | 5.71e-4 | 6.51e-4 | 6.93e-4 | 7.15e-4 | 7.62e-4 | 7.79e-4 | 7.82e-4 |
| Static (Cyclic)  [task size: 10 numbers] | 2.15e-4 | 3.97e-4 | 6.38e-4 | 6.09e-4 | 8.40e-4 | 8.47e-4 | 8.05e-4 | 8.53e-4 | 8.64e-4 | 8.72e-4 |
| Dynamic  [task size: 10 numbers] | 2.18e-4 | 3.93e-4 | 6.44e-4 | 7.73e-4 | 8.77e-4 | 9.00e-4 | 9.12e-4 | 9.18e-4 | 9.33e-4 | 9.25e-4 |

**[Explanation/Analysis on the Results]**

In the three