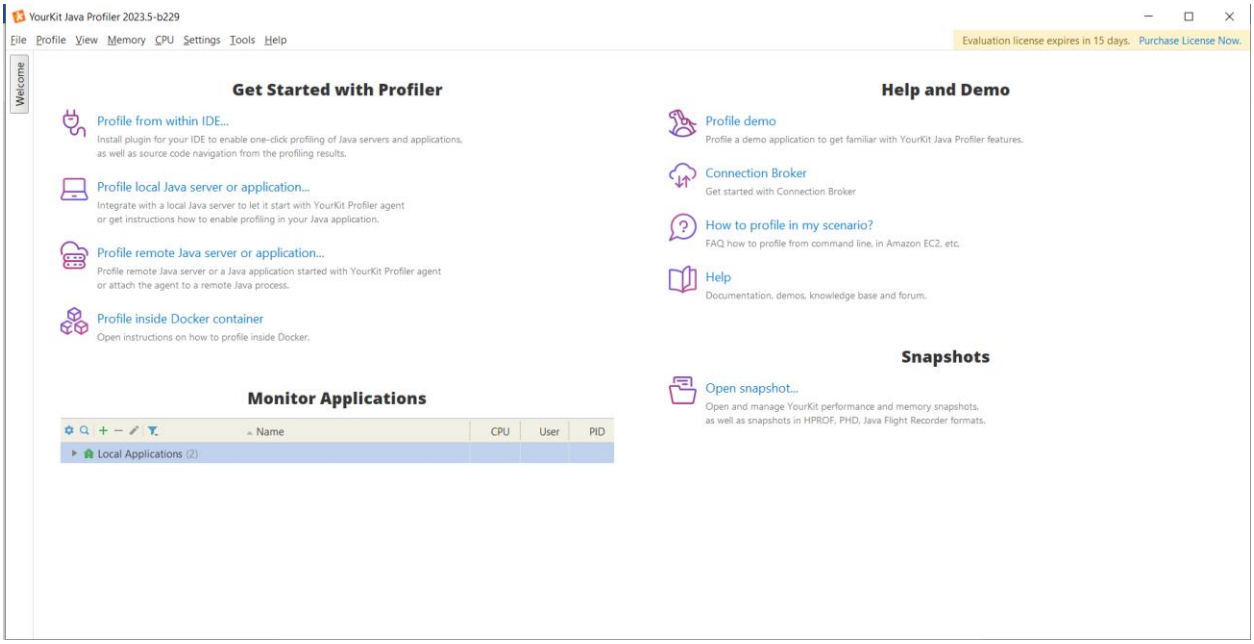
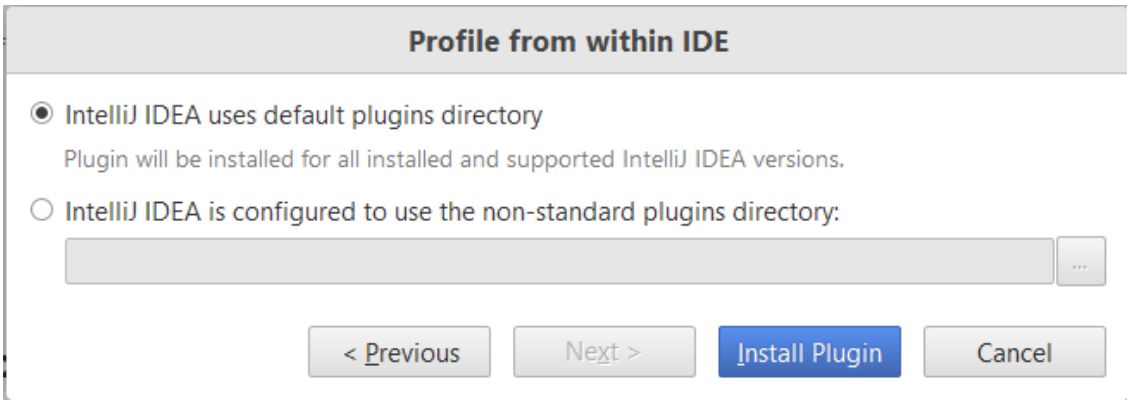
شرح آزمایش

ابتدا نرم افزار Yourkit مخصوص زبان جاوا را دانلود و نصب می‌کنیم و طبق گفته کلاس، Evaluation ۱۵ روزه را فعال می‌کنیم:

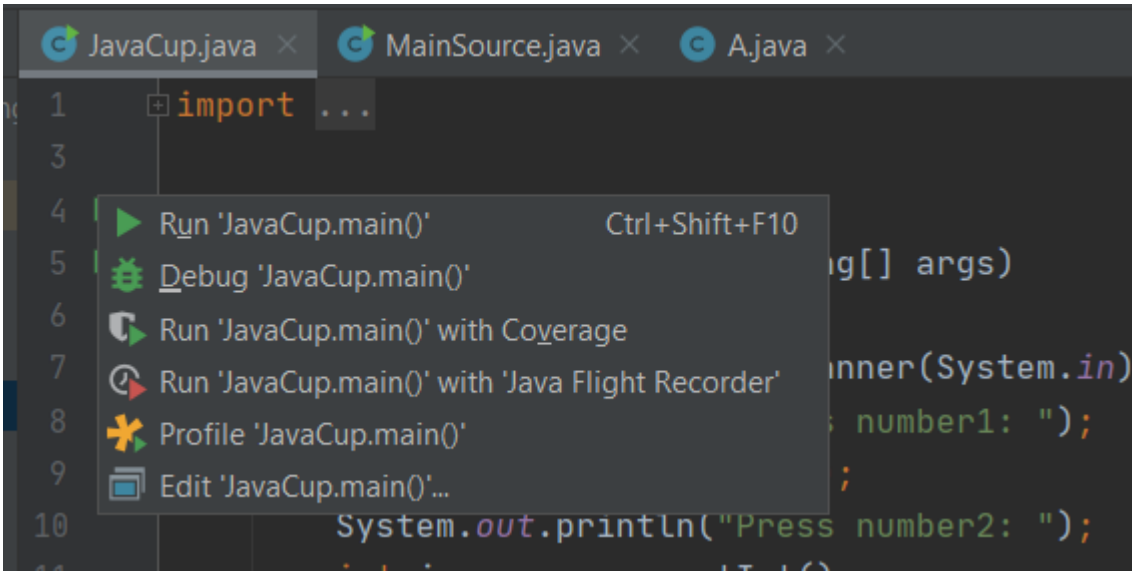


سپس روی گزینه Profile from within IDE کلیک می‌کنیم تا پلاگین Yourkit روی Intellij نصب شود.

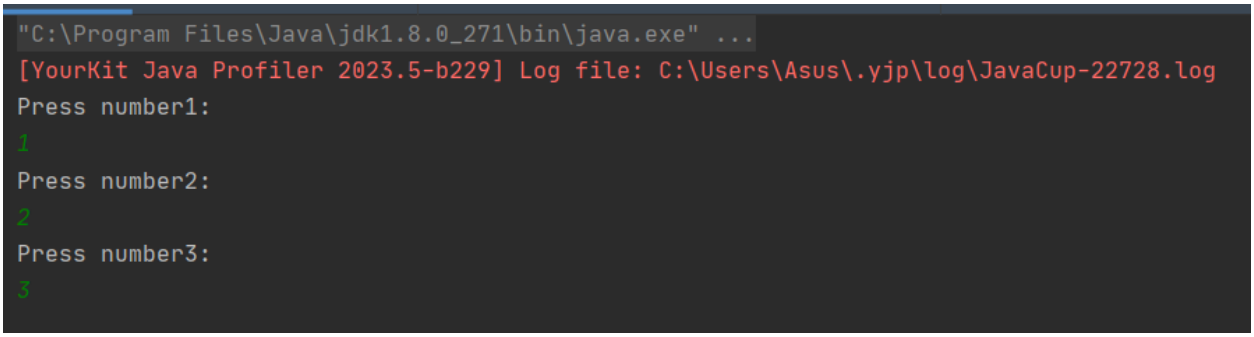


بخش اول

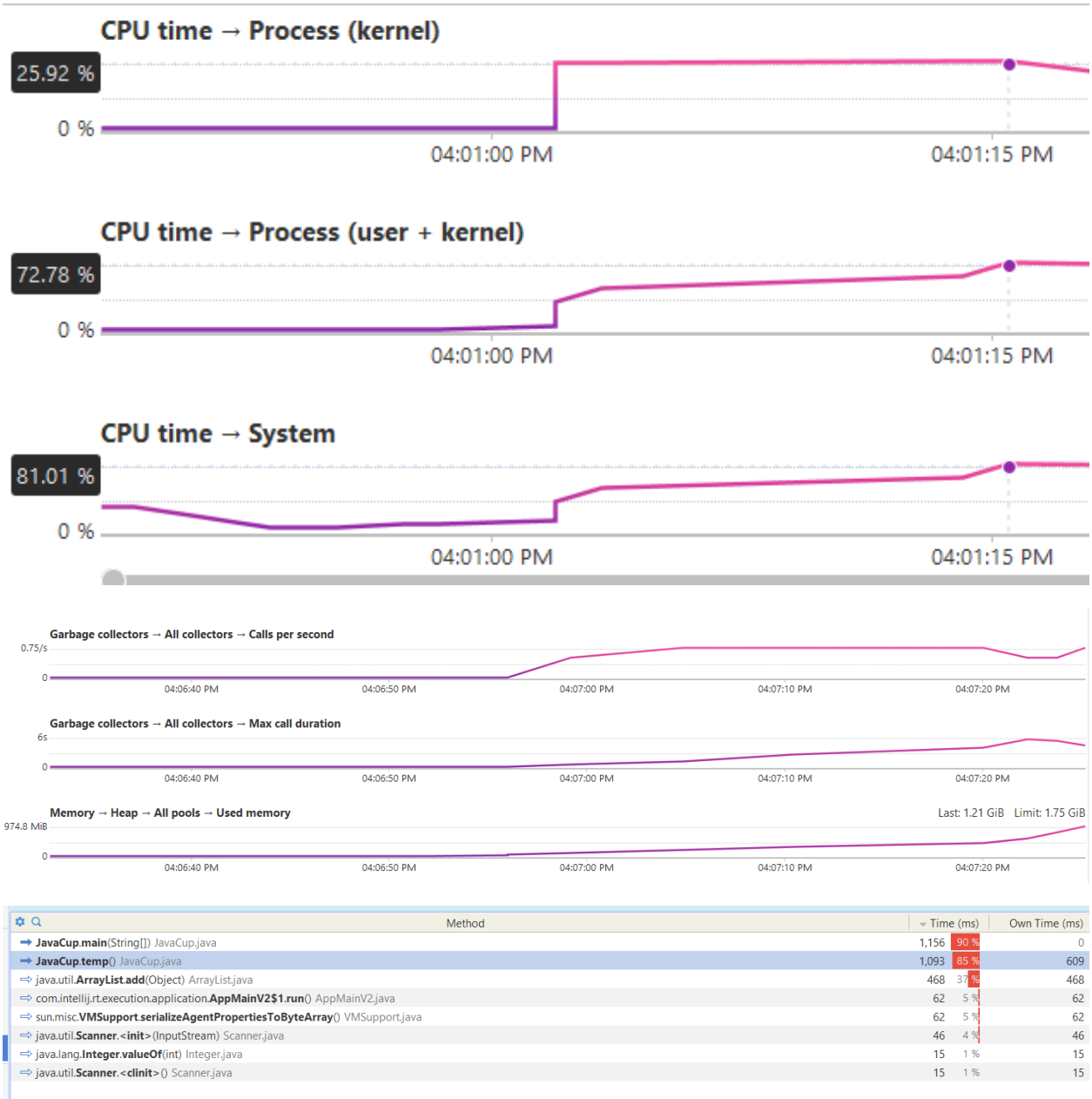
حال پروژه ذکر شده را در intellij باز می‌کنیم و عمل profiling را روی فایل JavaCup اجرا می‌کنیم:



ورودی‌های مورد نیاز را وارد می‌کنیم و منتظر اتمام عملیات اجرا و profiling می‌شویم:



بعد از اتمام اجرا، تب‌های مربوط به CPU و Memory و Method list را مشاهده می‌کنیم:



Method	Time (ms)	Own Time (ms)
JavaCup.main(String[]) JavaCup.java	1,156 90 %	0
JavaCup.temp() JavaCup.java	1,093 85 %	609
java.util.**ArrayList.add**(Object) ArrayList.java	468 37 %	468
com.intellij.rt.execution.application.**AppMainV2$1.run**() AppMainV2.java	62 5 %	62
sun.misc.**VMSupport.serializeAgentPropertiesToByteArray**() VMSupport.java	62 5 %	62
java.util.**Scanner.<init>**(InputStream) Scanner.java	46 4 %	46
java.lang.**Integer.valueOf**(int) Integer.java	15 1 %	15
java.util.**Scanner.<clinit>**() Scanner.java	15 1 %	15

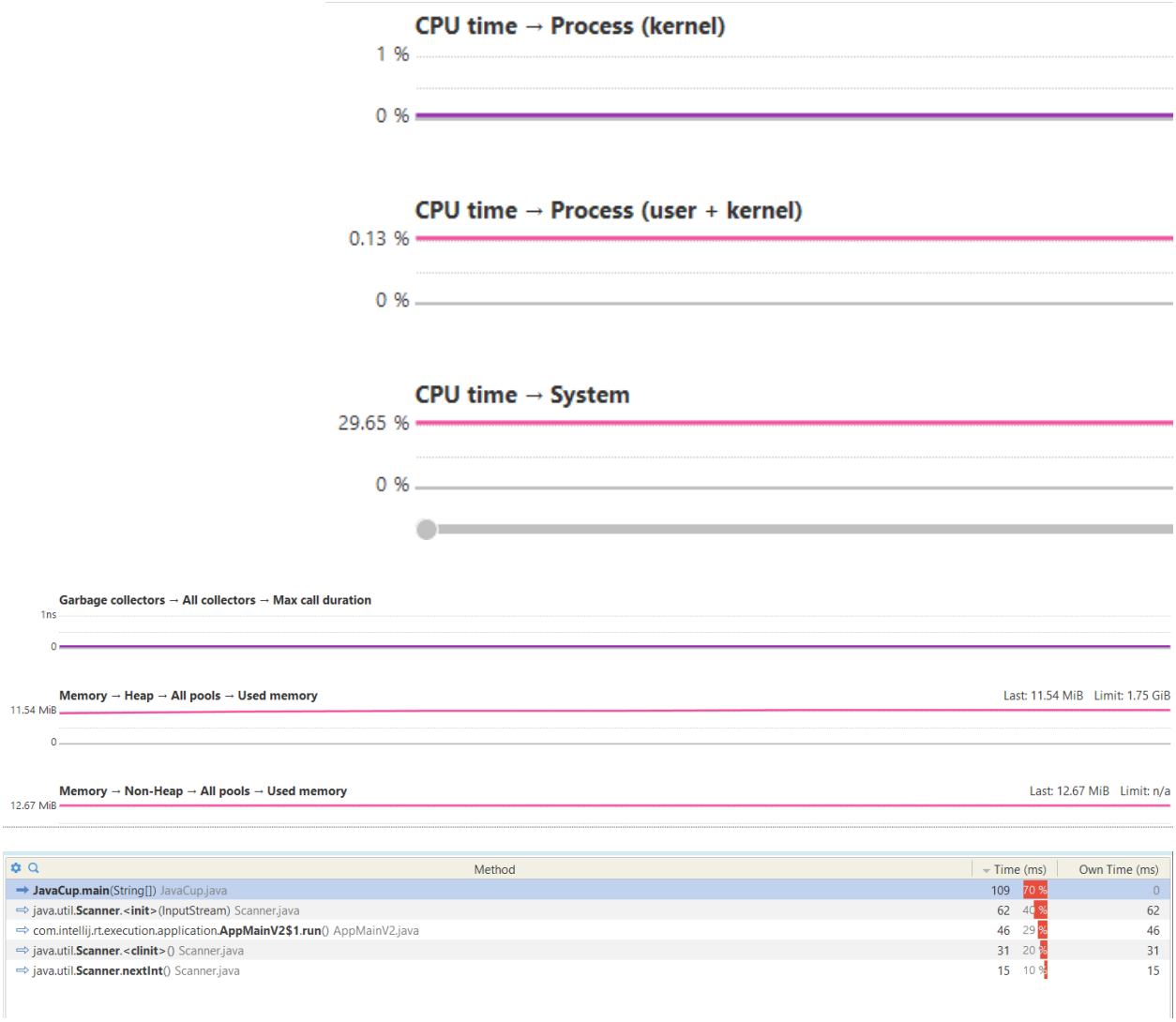
همانطور که مشخص است، در زمان اجرا، ۸۰ درصد پردازنده سیستم اشغال شده بود و Memory مربوط به Heap نیز به خاطر اینکه در حال ایجاد یک ArrayList بزرگ بودیم در حال پر شدن بود و تابعی که بیشترین مصرف را داشته، تابع temp بوده است.

```java
public static void temp() {
    ArrayList a = new ArrayList();
    for (int i = 0; i < 10000; i++)
    {
        for (int j = 0; j < 20000; j++) {
            a.add(i + j);
        }
    }
}
```

یک راه برای کم‌کردن زمان اجرا و مصرف منابع این تابع این است که می‌توانیم به جای ArrayList از Array استفاده کنیم؛ چون اندازه این آرایه ثابت است و قرار نیست تغییر کند. بعد از انجام این تغییر کد به شکل زیر در می‌آید:

```java
public static void temp() {
    int[] a = new int[10000 * 20000];
    for (int i = 0; i < 10000; i++)
    {
        for (int j = 0; j < 20000; j++) {
            a[20000*i + j] = i + j;
        }
    }
}
```

بعد از این تغییر، نتیجه profiling برنامه به این شکل خواهد بود:



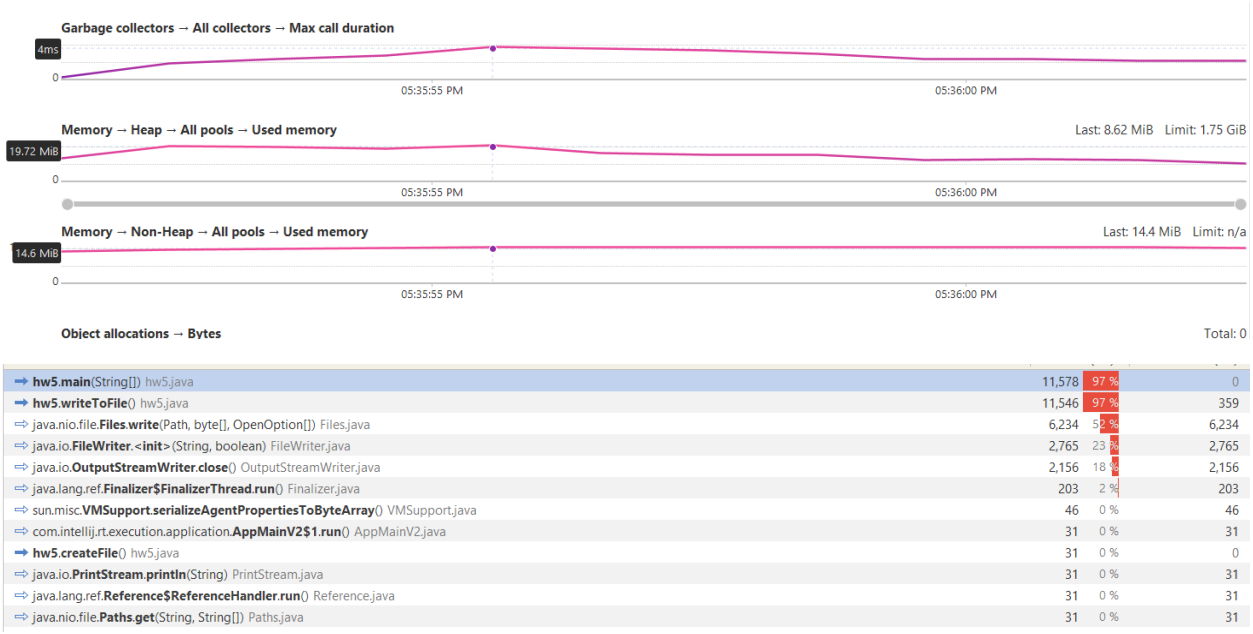
که بهبود برنامه را نشان می‌دهد.

بخش دوم

در این بخش یک قطعه کد در فایل hw5.java می‌نویسیم که وظیفه نوشتن به ترتیب اعداد ۱ تا ۱۰۰۰۰۰ را در فایلی به نام result.txt دارد. قطعه کد در ابتدا به این شکل است:

```java
public class hw5 {
    public static void main(String[] args) {
        createFile();
        writeToFile();
    }

    private static void writeToFile() {
        for (int i = 0; i < 100000; i++) {
            try {
                FileWriter myWriter = new FileWriter( fileName: "result.txt",  append: true);
                Files.write(Paths.get( first: "result.txt"), (i + "\n").getBytes(), StandardOpenOption.APPEND);
                myWriter.close();
            } catch (IOException e) {
                System.out.println("An error occurred.");
                e.printStackTrace();
            }
        }
        System.out.println("Successfully wrote to the file.");
    }

    private static void createFile() {
        try {
            File myObj = new File( pathname: "result.txt");
            if (myObj.createNewFile()) {
                System.out.println("File created: " + myObj.getName());
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```

حال عمل profiling را روی این کد انجام می‌دهیم:



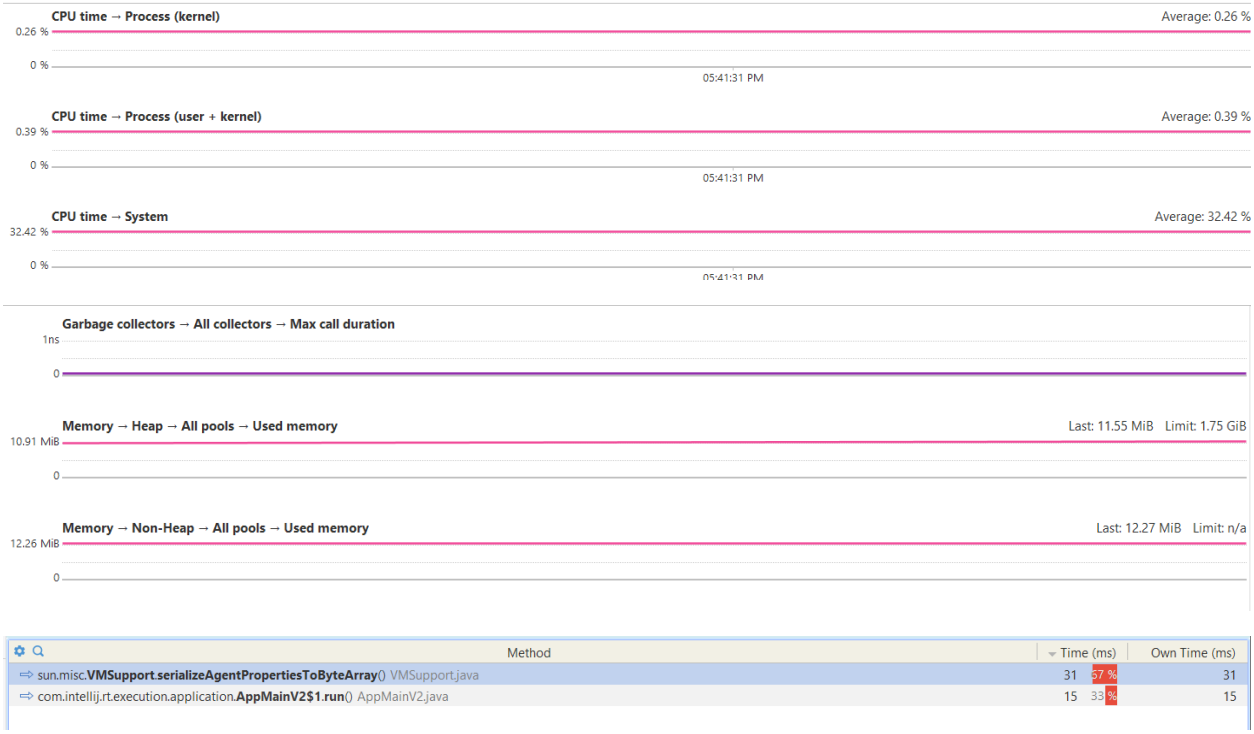
همانطور که مشاهده می‌شود، اجرای برنامه حدودا ۱۰ ثانیه طول کشیده است و حین اجرا، ۲۰ مگابایت حافظه و ۱۲ درصد پردازنده درگیر شده است. همچنین تابع writeToFile بیشترین مصرف منابع را به خود اختصاص داده است؛ بنابراین باید این تابع را بهبود دهیم. حال برای بهبود عملکرد برنامه، قطعه کد را به شکل زیر تغییر می‌دهیم:

```java
public static void main(String[] args) {
    createFile();
    writeToFile2();
}

private static void writeToFile2() {
    try {
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < 100000; i++) {
            s.append(i).append('\n');
        }
        FileWriter myWriter = new FileWriter( fileName: "result.txt");
        myWriter.write(s.toString());
        myWriter.close();
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
    System.out.println("Successfully wrote to the file.");
}

private static void createFile() {
    try {
        File myObj = new File( pathname: "result.txt");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
```

تغییرات اعمال شده به این شکل است: به جای اینکه هر بار در حلقه فایل را باز کنیم و عدد را بنویسیم و فایل را ببندیم، رشته مورد نظر را با یک StringBuilder می‌سازیم و در آخر فقط یک بار فایل را باز می‌کنیم و کل

رشته را در آن می‌نویسیم و فایل را می‌بندیم. چون عمل باز کردن فایل فرایند زمان‌بری است و تغییر یک StringBuilder با منابع و زمان بسیار کمتری انجام می‌گیرد.

حال عمل profiling را روی قطعه کد جدید انجام می‌دهیم:



همانطور که در تصاویر مشخص است، اختلاف میزان مصرف دو برنامه خیلی زیاد است و برنامه بهبود زیادی پیدا کرده است و زمان اجرای آن از ۱۰ ثانیه به ۳۱ میلی ثانیه کاهش یافته است.