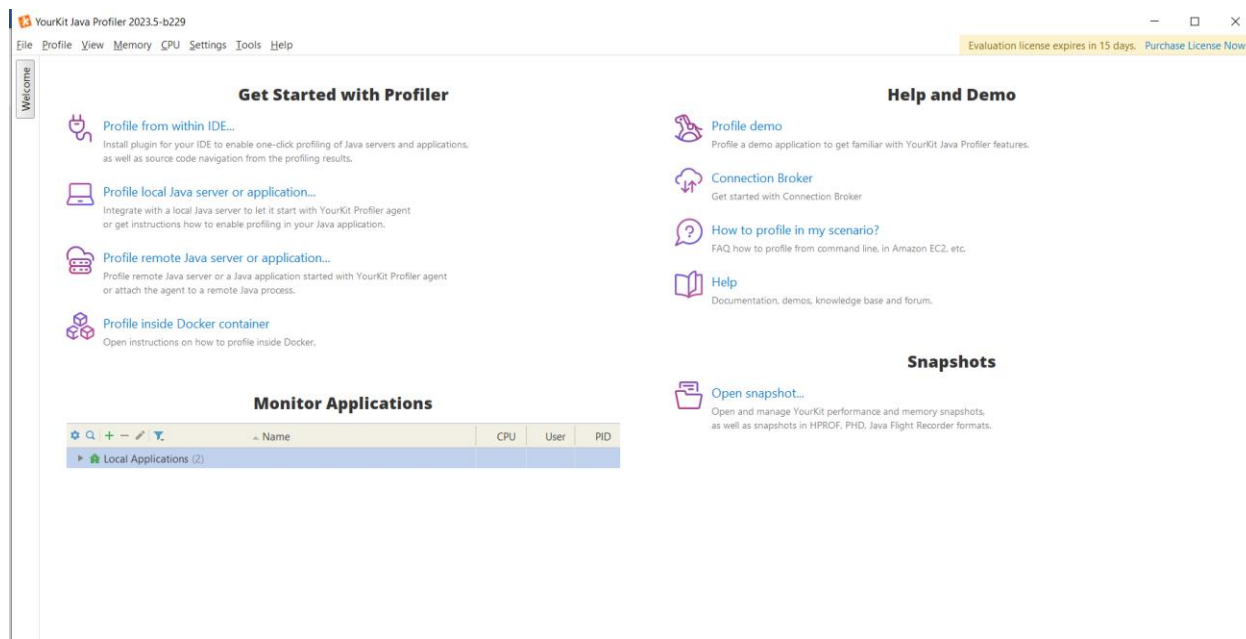


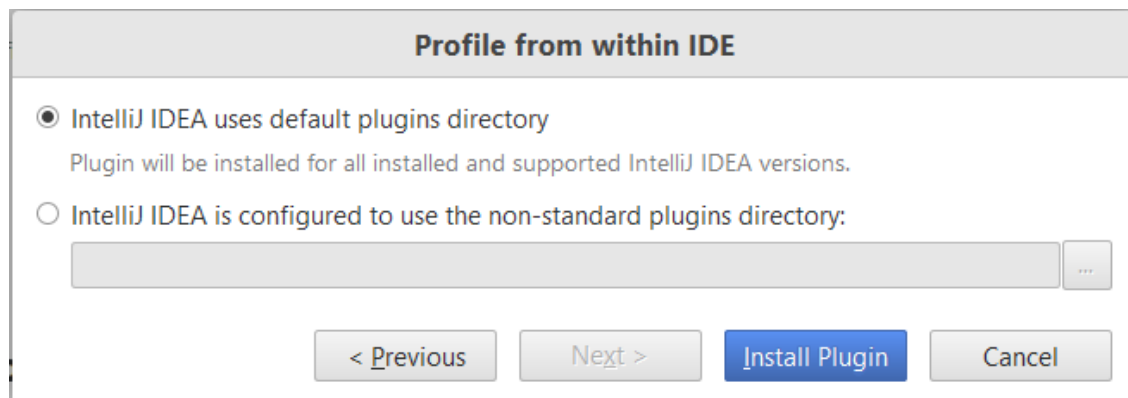
شرح آزمایش

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



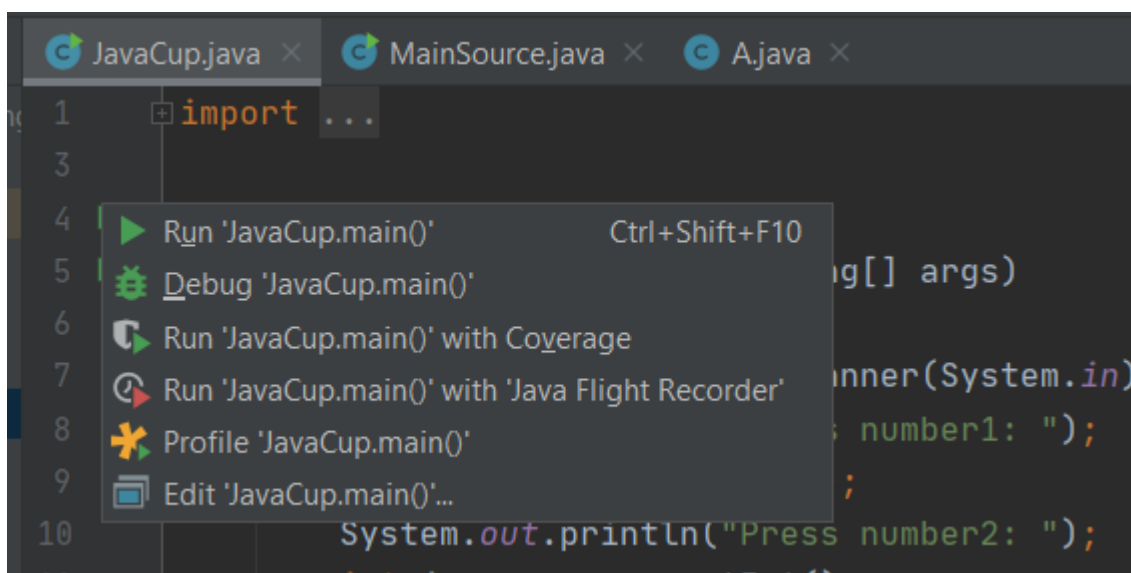
سپس روی گزینه Profile from within IDE کلیک می کنیم تا پلاگین Yourkit روی IntelliJ نصب شود.



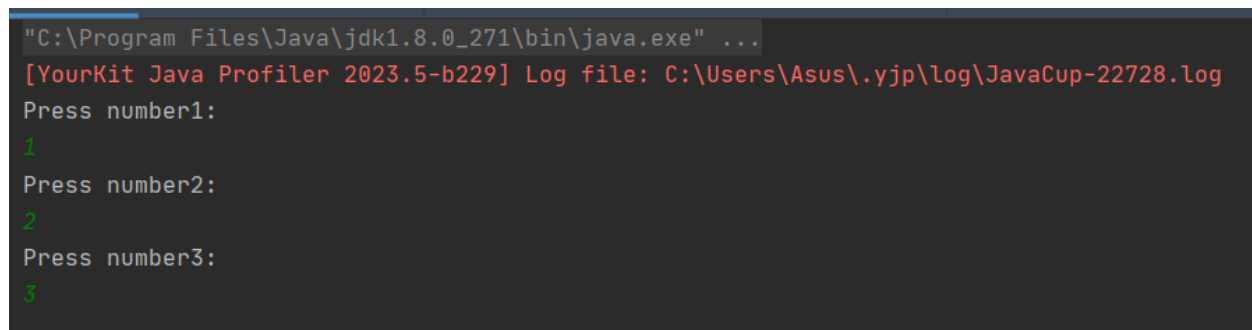


بخش اول

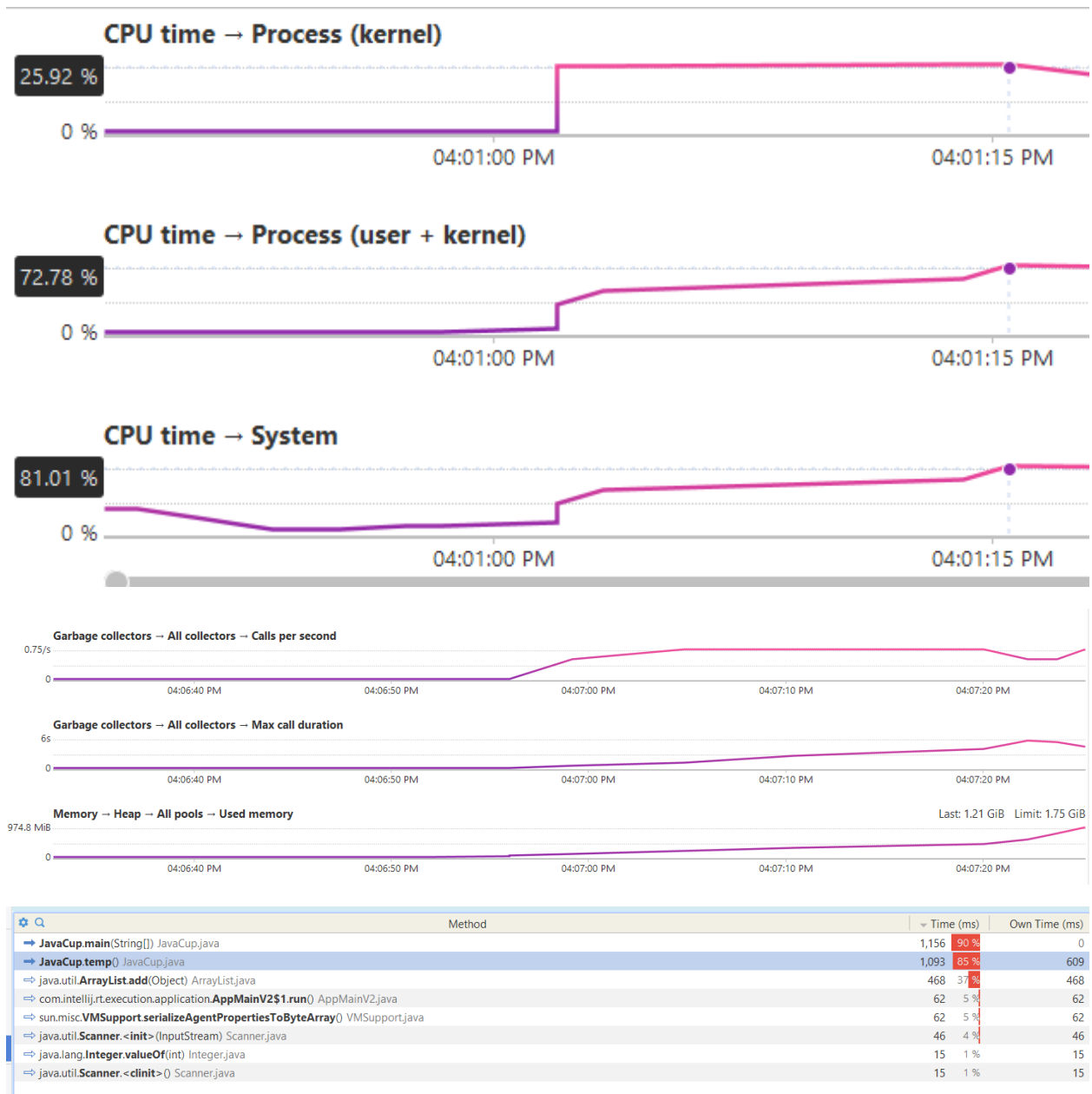
حال پروژه ذکر شده را در IntelliJ باز می‌کنیم و عمل **profiling** را روی فایل **JavaCup** اجرا می‌کنیم:



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:



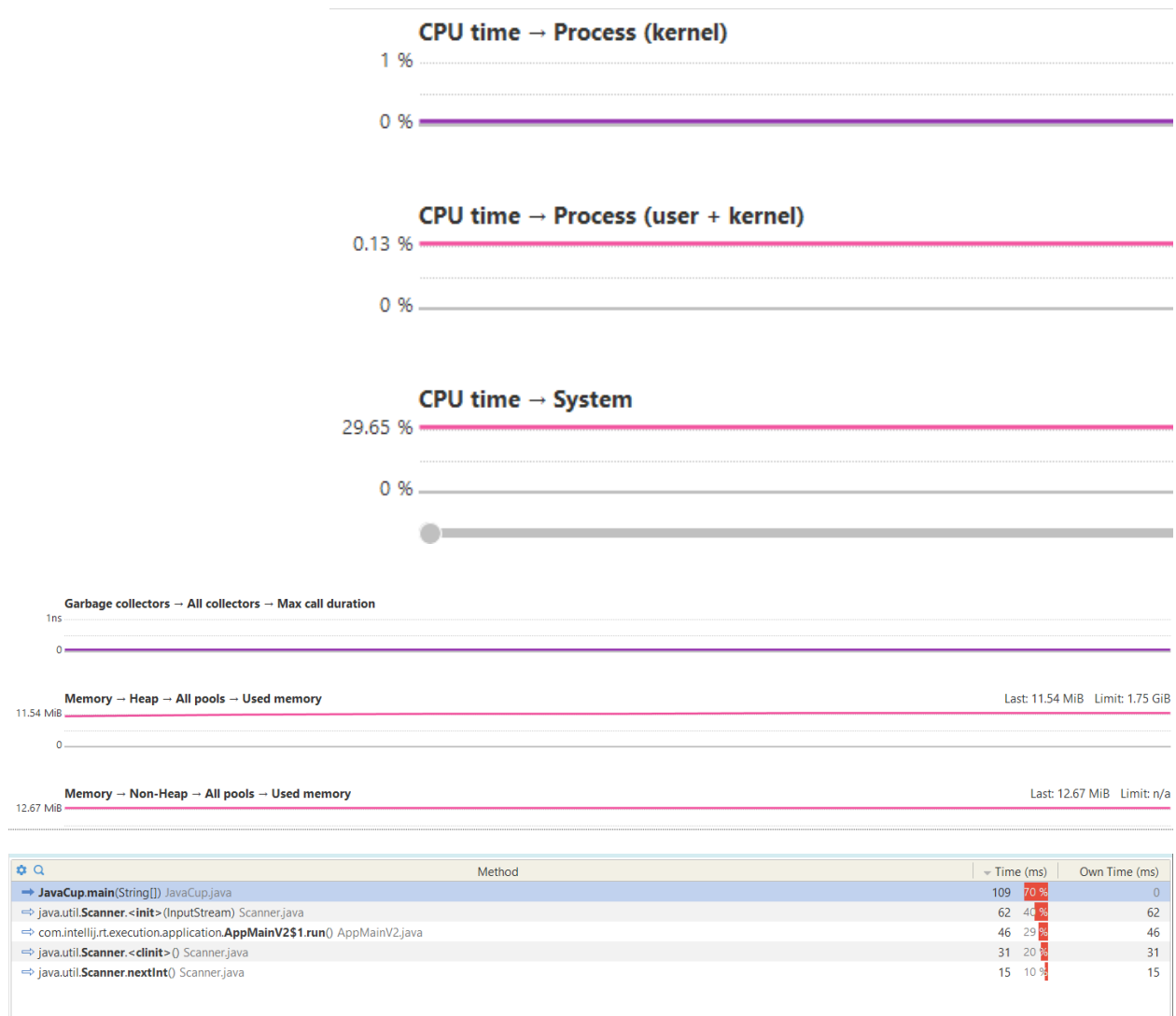
همانطور که مشخص است، در زمان اجرا، ۸۰ درصد پردازنده سیستم اشغال شده بود و Memory مربوط به Heap نیز به خاطر اینکه در حال ایجاد یک ArrayList بزرگ بودیم در حال پر شدن بود و تابعی که بیشترین مصرف را داشته، تابع temp بوده است.

```
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 کم کردن زمان اجرا و مصرف منابع این تابع این است که می توانیم 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 می آید:

```
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 فایل به نام result.txt دارد. قطعه کد در ابتدا به این شکل است:

```

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

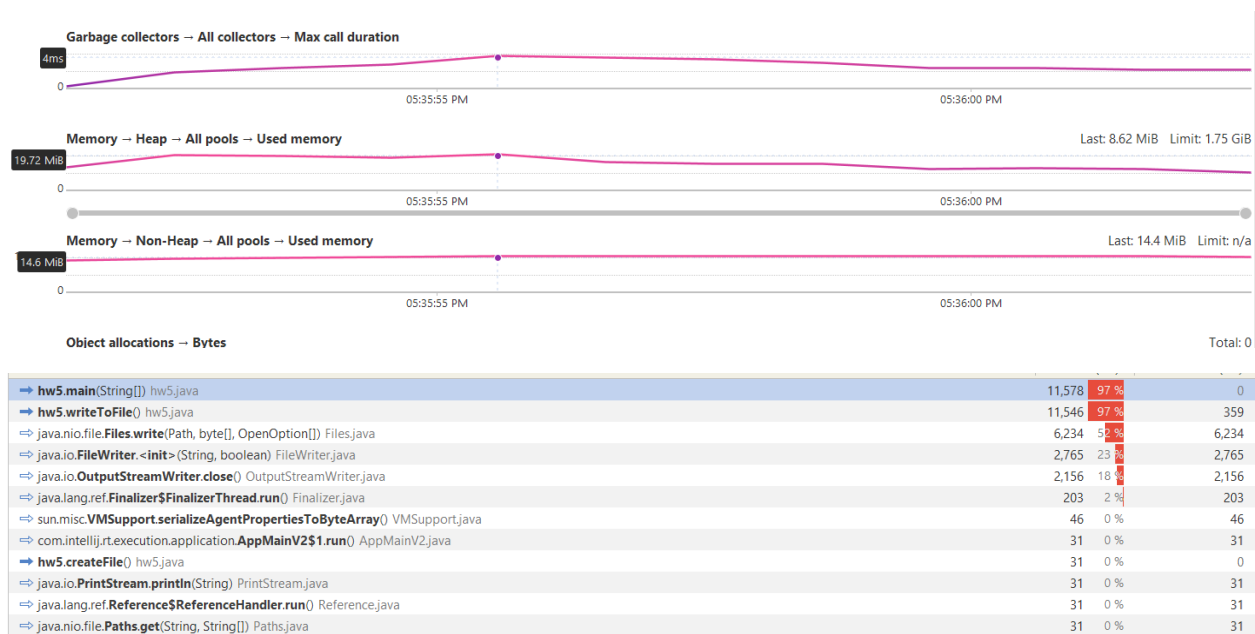
    private static void writeToFile() {
        for (int i = 0; i < 100000; i++) {
            try {
                FileWriter myWriter = new FileWriter( fileName: "result.txt", append: true);
                Files.write(Paths.get( first: "result.txt"), (i + "\n").getBytes(), StandardOpenOption.APPEND);
                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 می دهیم:





همانطور که مشاهده می‌شود، اجرای برنامه حدوداً ۱۰ ثانیه طول کشیده است و حین اجرا، ۲۰ مگابایت حافظه و ۱۲ درصد پردازنده درگیر شده است. حال برای بهبود عملکرد برنامه، قطعه کد را به شکل زیر تغییر می‌دهیم:

```

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("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.");
}

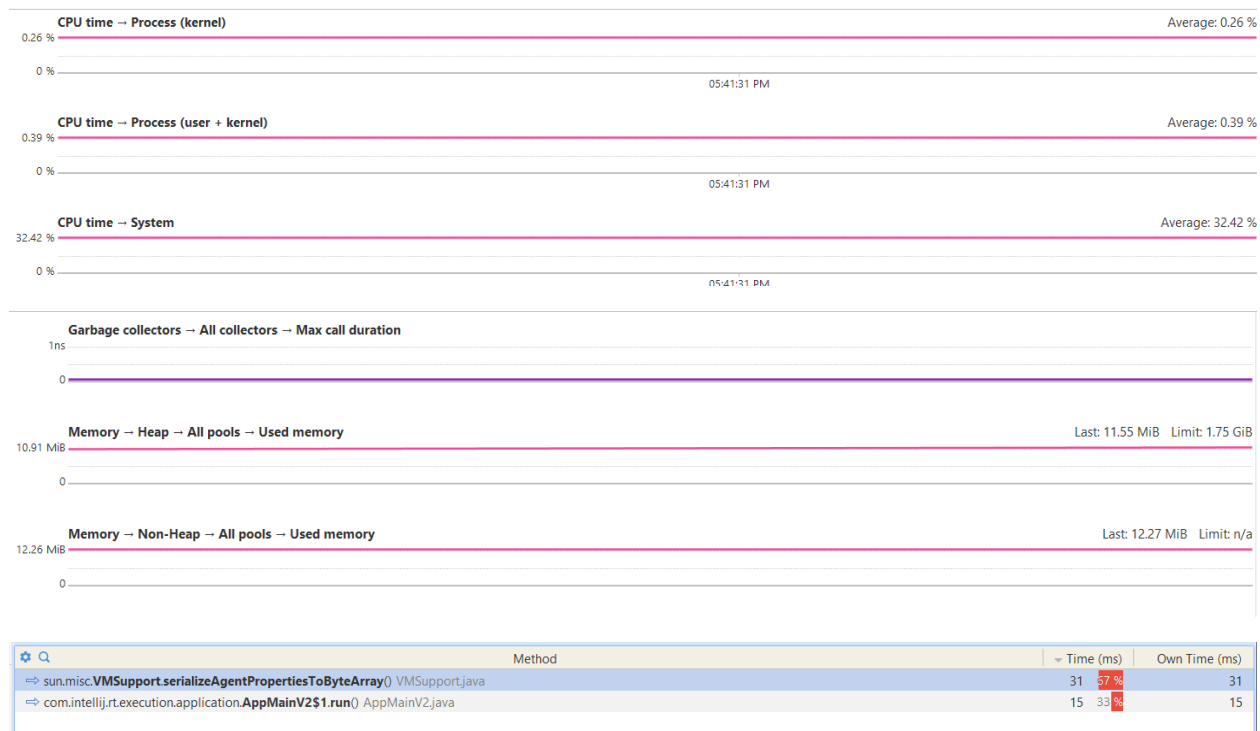
private static void createFile() {
    try {
        File myObj = new File("result.txt");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException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