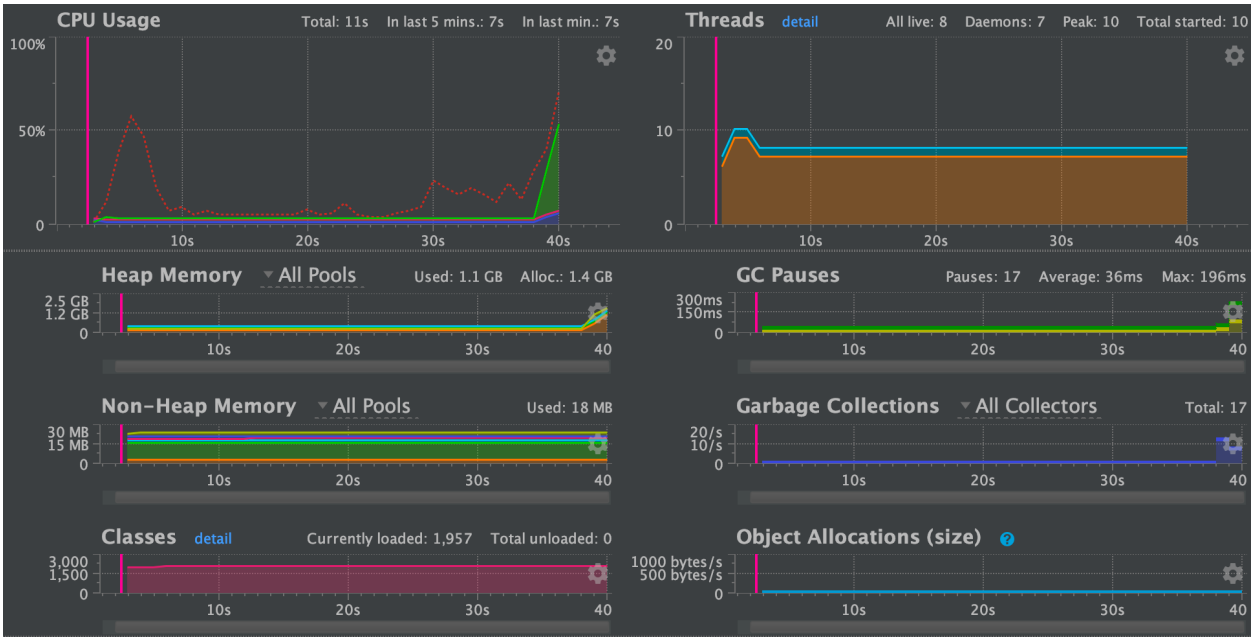
آز مهندسی نرم افزار
گزارش آزمایش اول

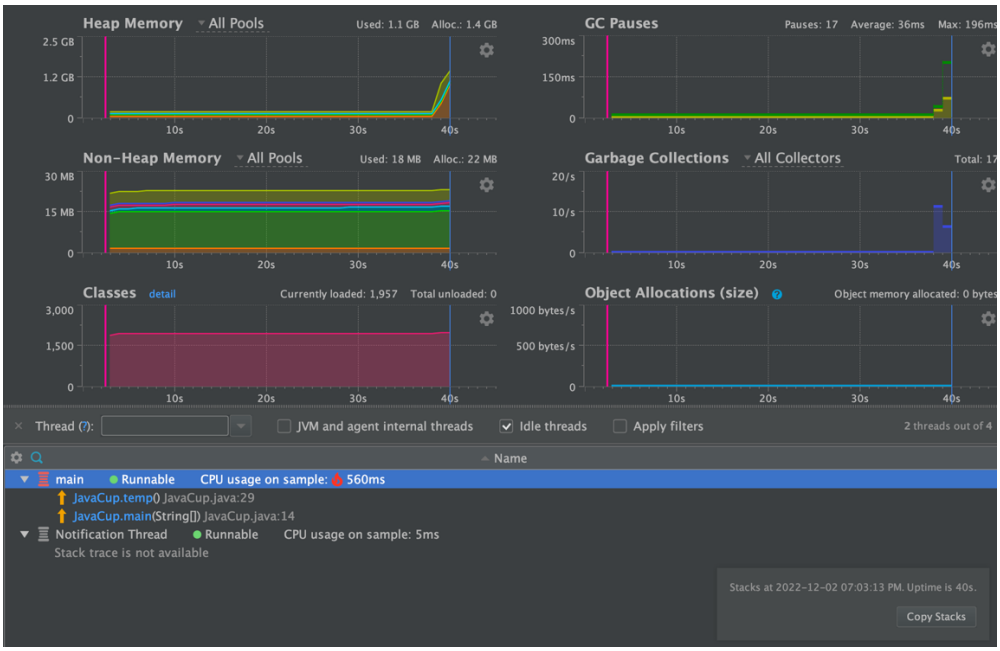
شایان چشم جهان – محسن کثیری

سوال ۱.

ابتدا در کلاس JavaCup عملیات Profiling را با استفاده از ابزار YourKit انجام می‌دهیم. در تصاویر زیر می‌توانیم نمودارهای مربوطه که با دادن ورودی‌های ۱ و ۱ و ۱ بدست آمده‌اند را مشاهده کنیم.



مهم‌ترین نکته‌ای که در اینجا جلب توجه می‌کند، پیک استفاده از مموری در انتها است. برای بررسی دلیل این استفاده‌ی بی‌رویه از مموری روی نقطه‌ی پیک کلیک کرده و می‌بینیم در آن لحظه چه متدهایی در حال اجرا بوده‌اند. در تصویر زیر اطلاعات مربوطه را مشاهده می‌کنیم.



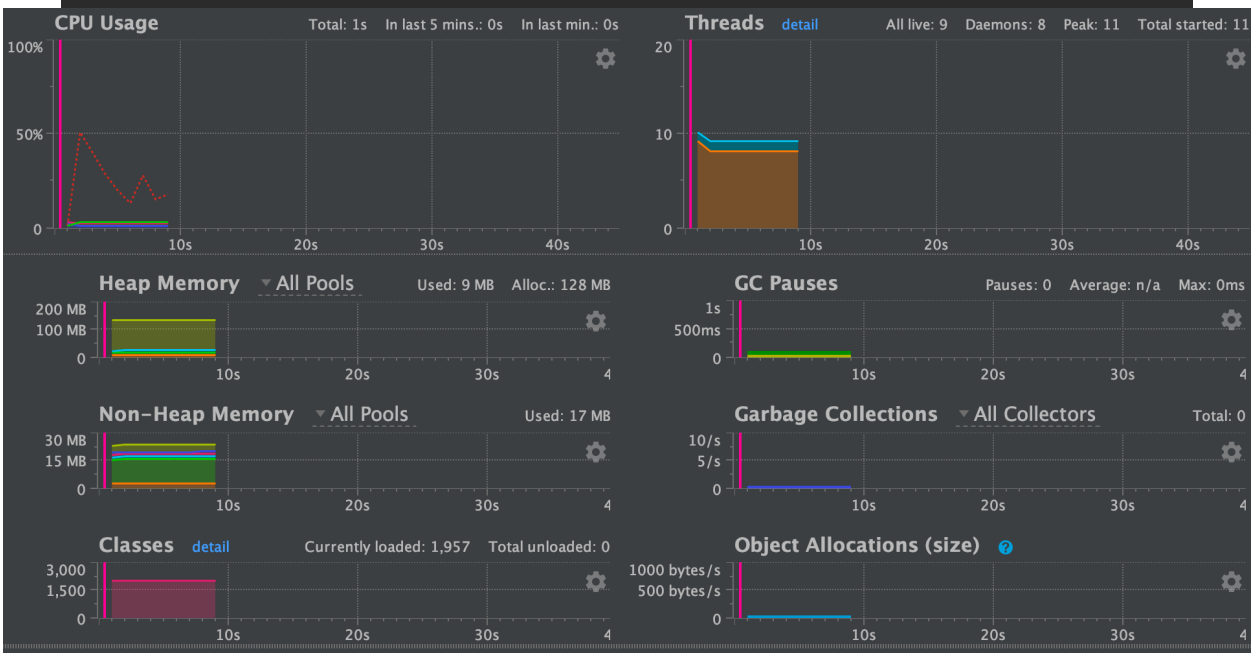
می‌بینیم که در آن لحظه متد JavaCup.temp() در حال اجرا بوده است. روی آن کلیک کرده تا در IntelliJ این متد به ما نمایش داده شود. در شکل زیر این متد قابل مشاهده است.

```
1 usage
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);
        }
    }
}
```

در تصویر بالا می‌بینیم که در این متد ۲×۱۰^۸ عدد در یک ArrayList ریخته شده اند، که مموری بسیار زیادی را مصرف می‌کنید و همین مشکل برنامه ما است. با بررسی بیشتر متوجه می‌شویم این ArrayList در هیچ‌جا هم استفاده نمی‌شود و کلا این متد temp هم تنها یکبار صدا زده می‌شود که تاثیری در عملکرد برنامه ندارد. در نتیجه برای تصحیح عملکرد برنامه کافی است به طور کلی هر دو حلقه‌ی موجود در این تابع را پاک کنیم و می‌توانیم مطمئن باشیم که تاثیری روی عملکرد برنامه نمی‌ذارد، همینطور می‌توانستیم کلا هم تابع temp را حذف کنیم.

محتوای جدید متد temp و عملکرد برنامه که با YourKit آنالیز شده است در تصاویر زیر موجود است.

```
1 usage
public static void temp() {
    ArrayList a = new ArrayList();
}
```



می‌بینیم که عملکرد برنامه تصحیح شده و دیگر شاهد هیچ پیکی در استفاده از CPU یا مموری نیستیم.

سوال ۲.

برای این سوال ابتدا به شیوه‌ی بدی sort را پیاده‌سازی می‌کنیم. الگوریتم ما به این صورت است که بزرگترین عدد در آرایه را بدست می‌آوریم، سپس یک فور از ۰ تا بزرگترین عدد می‌زنیم، و با یک فور روی آرایه، جایگاه‌هایی که عدد مورد نظرمان در آرایه ظاهر شده را پیدا می‌کنیم و در آرایه‌ای می‌ریزیم، در انتهای به ازای هر کدام از آن جایگاه‌ها عدد مورد نظر را به انتهای آرایه‌ی نهایی اضافه می‌کنیم. اردر این الگوریتم $O(n.A)$ است (که در اینجا A به عنوان بزرگترین عددی که ممکن است کاربر در آرایه ورودی بدهد است) که بسیار بد است. در تصویر زیر پیاده سازی این روش قابل مشاهده است.

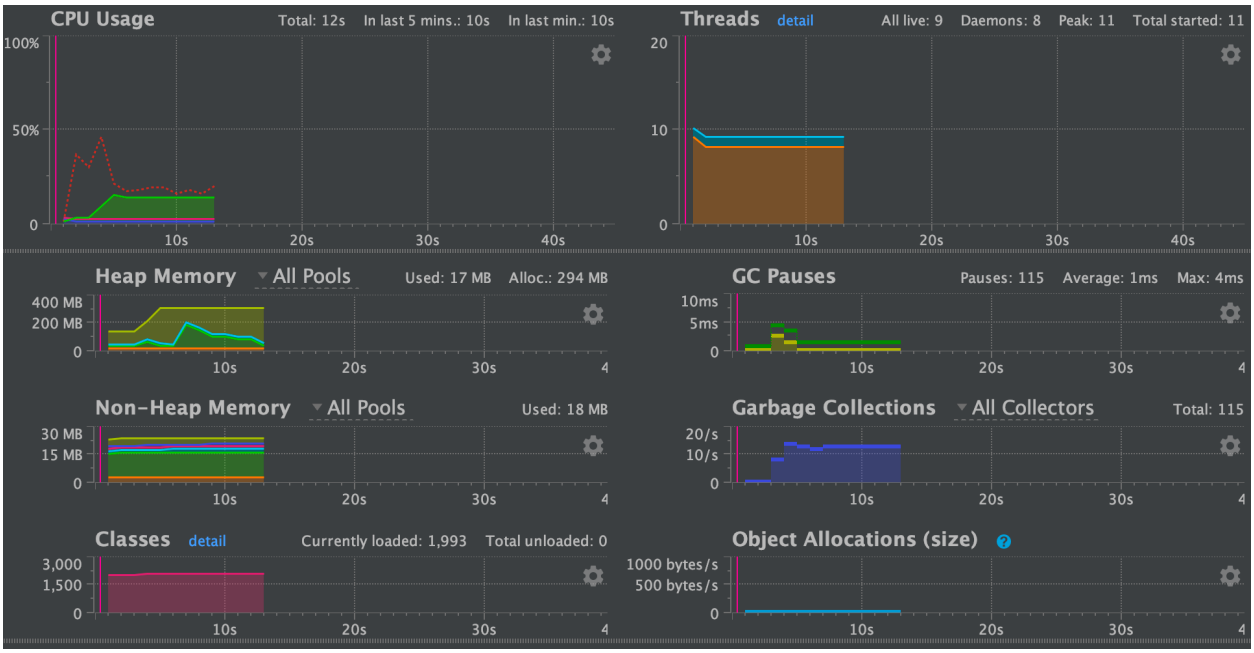
```java
public class BadSort {
    1 usage
    public ArrayList<Integer> sort (ArrayList<Integer> arrayList) {
        ArrayList<Integer> result = new ArrayList<>();
        int n = arrayList.size(), maxElement = 0;

        for (int i = 0; i < n; i++){
            maxElement = Math.max(arrayList.get(i), maxElement);
        }

        for (int value = 0; value <= maxElement; value++) {
            ArrayList<Integer> indices = new ArrayList<>();
            for (int i = 0; i < n; i++){
                if (arrayList.get(i) == value) {
                    indices.add(i);
                }
            }
            for (int i = 0; i < indices.size(); i++) {
                result.add(value);
            }
        }

        return result;
    }
}
```

حال با ورودی دادن آرایه‌ای که بزرگترین عضو آن $۱۰^۹$ است کاری می‌کنیم تا performance این متد باشد، و با استفاده از YourKit این performance را بررسی می‌کنیم. در تصاویر زیر عملکرد برنامه قابل مشاهده است.



دوباره یک پیک در عملکرد برنامه می‌بینیم که استفاده از ممووری و CPU بسیار زیاد شده است.

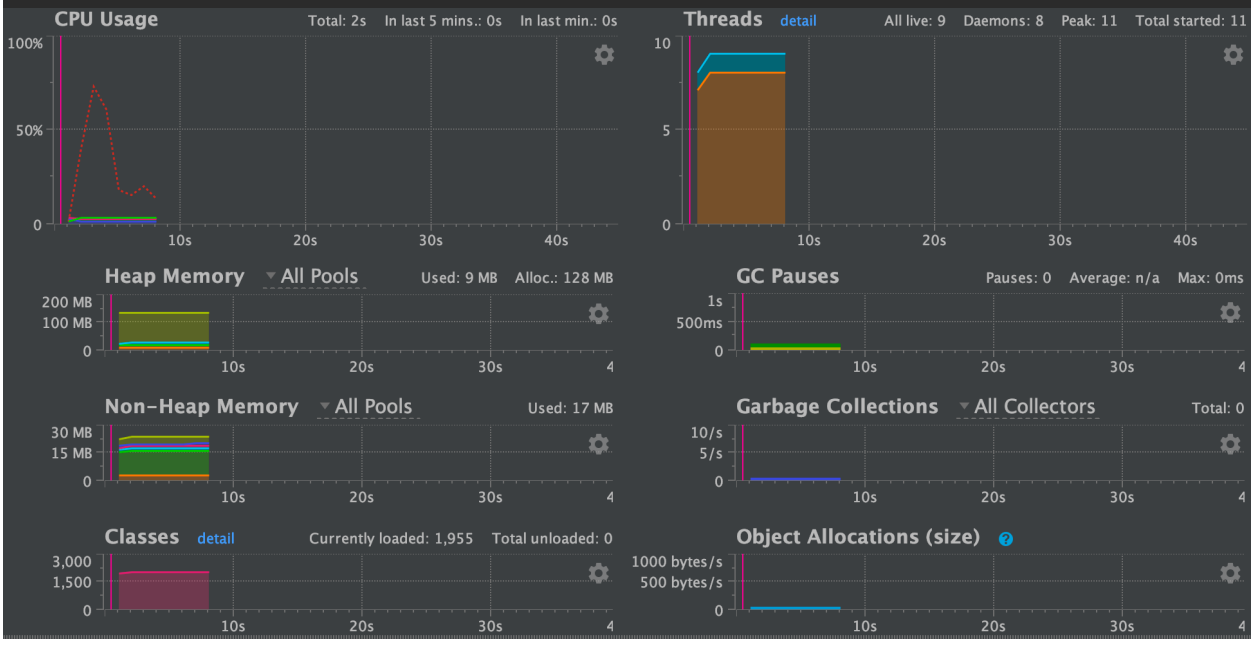
بار دیگر با استفاده از BubbleSort و در $O(n^2)$ سورت را پیاده سازی می‌کنیم و عملکرد متد جدید را با قبلی مقایسه می‌کنیم. در تصاویر زیر پیاده سازی جدید و performance آن قابل مشاهده است.

```java
public class BubbleSort {
    1 usage
    public ArrayList<Integer> sort (ArrayList<Integer> arrayList) {
        ArrayList<Integer> result = new ArrayList<>();
        int n = arrayList.size();

        for (int i = 0; i < n; i++) result.add(arrayList.get(i));

        for (int step = 0; step < n; step++) {
            for (int i = 0; i < n - 1; i++) {
                if (result.get(i) > result.get(i+1)) {
                    Collections.swap(result, i,  j: i+1);
                }
            }
        }

        return result;
    }
}
```



همانطور که در تصاویر پیداست عملکرد برنامه ما با استفاده از روش جدید بسیار بهتر می‌شود.

تمامی کدها و نتایج در ریپازیتوری گیت‌هاب موجود است:

https://github.com/ShayanJahan/SELAB_Session1