
The median of k numbers is defined as the $(k/2)^{th}$ smallest number if k is even; and the $((k+1)/2)^{th}$ smallest number if k is odd. For example, median of the 4 numbers: 2 1 8 7 is the 2nd smallest number i.e. 2, and the median of the 5 numbers: 2 1 8 7 6 is the 3rd smallest number i.e. 6.

In this problem, you are given N numbers. Let the k^{th} median or m(k) be defined as the median of the first k numbers (1<=k<=N), i.e. the 5^{th} median or m(5) is the median of the first 5 numbers, the 8^{th} median or m(8) is the median of the first 8 numbers, etc. In other words, let A_k denote the k^{th} number, then the k^{th} median or m(k) is defined as the median of the numbers A_1, A_2, …, A_k.

Your task is to find m(1) + m(2) + m(3) + ...+ m(n), and output the sum modulo 100000 (10^5).

INPUT:
The first line contains N, the count of numbers. N lines follow, each containing one number.

OUTPUT:
Output a single line, containing the sum of the medians modulo 100000.

CONSTRAINTS:
1<=N<=100000
0<= each number A_i <=100000

Sample Input (Plain text)	Sample Output (Plain text)
5 10 5 1 2 15	27

Explanation
m(1)=median of [10]=10
m(2)=median of [10 5]=5
m(3)=median of [10 5 1]=5
m(4)=median of [10 5 1 2]=2
m(5)=median of [10 5 1 2 15]=5
(m(1) + m(2) + m(3) + m(4) + m(5)) % 100000 = 27

Part 1 [10 marks]) Code and complexity analysis

The following algorithm (pseudocode) solves the above Sum Of Sequence of Median problem, **naively**.
Analyze the algorithm and describe the asymptotic complexity in Big-O notation. Show your steps clearly.

```
INPUT: The first line contains N, the count of numbers. N lines follow, each containing one number.
OUTPUT: Output a single line, containing the sum of the medians modulo 100000.
ALGORITHM:
    N ← Read one line from file, parse as an integer
    i ← 0
    SumOfMedian ← 0                          // to store sum of all median
    while (i < N)
        a[i] ←Read one line from file, parse as an integer
        merge sort (a[0] ... a[i])           // merge sort a[0], a[1], ... a[i]
        i ← i + 1                            // increment i. Now, i is the total number of numbers read.
        if i is even
            median ← a[i/2]                  // see problem definition
        else
            median ← a[i+1/2]                // see problem definition
        end if
        SumOfMedian ← SumOfMedian + median   // update SumOfMedian
    end while
    print out (SumOfMedian % 100000)         // the sum of the medians modulo 100000
```

Part 2 [80 marks]) Asymptotically faster algorithm
Design an algorithm and implement it with the Java programming language such that
a) it is *asymptotically faster* than that in Part 1, **and**
b) it passes all provided test cases.

Requirement:
Complexity analysis using Big-O notation [40marks].

a) Your algorithm must be asymptotically faster than that in Part 1 (otherwise, at most 50% will be given),
b) Correct complexity analysis, show your steps.

Program correctness [40 marks].

a) your program can pass test cases 1-4 correctly without errors,
b) your program can finish **ALL** test cases 1-4 under 10 seconds (in total) on a benchmark server.

Part 3 [10 marks]) Challenge
If your program can pass **ALL 1-6 test cases** under 10 seconds (in total) on the benchmark server, a final 10 marks will be awarded. Analyse the complexity of your algorithm.

Note: Test cases 5 and 6 are using a very large dataset. Only a very fast algorithm can handle it.

Tip 1: You are provided a compressed project file for this coursework - **CW2.zip**.
Once extracted, the folder structure looks like the following:

```
/CW2
|  diff.exe            - a program which finds and shows the difference between two files (see Tip 4-6).

|  libiconv2.dll       - a helper file for diff.exe to run on a windows machine
|  libintl3.dll        - a helper file for diff.exe to run on a windows machine.
|  Stopwatch.java      - a java class provided to handle timing  - no modification allowed.
|  TestClass.bat       - a script to test your program (see description below, see Tip 6)
|  TestClass.sh        - same as TestClass.bat, see Tip 6 also.
|  TestClass.java      - your java program (see Tip 2). modify this template for submission.
+---input              - a folder that contains 6 input test cases (see Tip 3)
|     test1.txt        - test case#1
|     test2.txt        - test case#2
|     test3.txt        - test case#3
|     test4.txt        - test case#4
|     test5.txt        - test case#5
|     test6.txt        - test case#6
\---result             - a folder that contains correct results from each test cases (see Tip 4)
      Result1.txt      - results for test case #1
      Result2.txt      - results for test case #2
      Result3.txt      - results for test case #3
      Result4.txt      - results for test case #4
      Result5.txt      - results for test case #5
      Result6.txt      - results for test case #6
```

Tip2: You must use the following code template, and fill in the blanks.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

// ......  your code here ......
// ......
// ......  define your own functions/classes, if need to ......
// ......

class TestClass {
   // ......  your code here ......
   // ......
   // ......  define your own functions/classes, if need to ......
   // ......

   public static void main(String args[] ) throws Exception {

       // read number of data from system standard input.
       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
       String line = br.readLine();
       int N = Integer.parseInt(line);

       // median sum
       long SumMedians = 0;

       // ......  your code here ......
       // ......  your code here ......
       // ......  your code here ......

       // write output to system standard output
       System.out.println(SumMedians % 100000);
   }
}
```

Tip3: You should prepare and test your program **locally** at your own machine. Use the provided test cases.

```
>javac *.java
>java TestClass < input\test1.txt
27
>java TestClass < input\test2.txt
45
>java TestClass < input\test3.txt
11
>java TestClass < input\test4.txt
4086
>java TestClass < input\test5.txt
20888
>java TestClass < input\test6.txt
3023
```

Tip 4: All test cases #1-6 are provided with expected **correct** results in respective files Result1.txt ... Result6.txt.
You can easily verify your program output as follows:
Use "`> Output1.txt`" to redirect/pipe your result from standard output (screen) into a file Output1.txt.
Use "`diff Result1.txt Output1.txt`" to check and compare your result.
`diff` is a program to find "difference" in two files. If there is **nothing** returned like below, your result is **correct**.

```
~/CW2> javac TestClass.java
~/CW2> java -Xmx1024m TestClass < input\test1.txt > Output1.txt
Elapsed Time: 0.001 s
~/CW2> diff result\Result1.txt Output1.txt

~/CW2>
```

On linux/mac, the `diff` program is available from the operating system by default.

Tip 5: If your program produces incorrect results, you will see the difference when you run the **diff** program. For example, if you use the provided java template (Tip 2) which **does nothing**, this is the output.

```
~/CW2> javac TestClass.java
~/CW2> java -Xmx1024m TestClass < input\test1.txt > Output1.txt
Elapsed Time: 0.0 s
~/CW2> diff result\Result1.txt Output1.txt
1c1
< 27
---
> 0
~/CW2>
```

Tip 6: If you want to process all test cases in one go, use **TestClass.bat** (on windows) or **TestClass.sh** (on Linux/Mac). The following example indicates that all results are **incorrect**.

```
~/CW2> javac TestClass.java
~/CW2> ./TestClass.bat
Elapsed Time: 0.0 s
Elapsed Time: 0.0 s
Elapsed Time: 0.0 s
Elapsed Time: 0.001 s
Elapsed Time: 0.001 s
Elapsed Time: 0.001 s
Files result/Result1.txt and Output1.txt differ
```

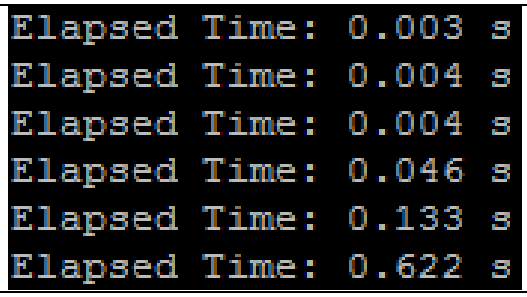
```
Files result/Result2.txt and Output2.txt differ
Files result/Result3.txt and Output3.txt differ
Files result/Result4.txt and Output4.txt differ
Files result/Result5.txt and Output5.txt differ
Files result/Result6.txt and Output6.txt differ
~/CW2>
```

To allow linux/mac OS to run the TestClass.sh, use the following command to change permission:
chmod 755 TestClass.sh and replace ./TestClass.bat with **./TestClass.sh** in the above example.

Tip 7: Your program must process all 4 test cases and give the correct results. **All** test cases must be processed in **under 10 seconds (in total)** on the benchmark server (IP: 137.44.2.196). Login details have been sent individually to your email. On the benchmark server, there is NO javac compiler available. You should work locally on your own machine, and upload TestClass.class when you are ready. Use the benchmark server **only** to test your java program **for timing**.

Tip 8: test case #5-6 are large datasets. You need a fast algorithm to handle them.
That's why we use the **java -Xmx1024m** flag to allow JVM to use more memory and to create large integer arrays.
To score full marks, your program must obtain the correct result for **ALL test case#1-6** in under **10 seconds** on the benchmark server. **Check out the provided *A2-ServerSetup.pdf* to setup your testing environment on the server.**
A sample solution gives the following results, with timing on the benchmark server (see screenshot):

```
~/CW2> ./TestClassSvr_CW2.sh
Elapsed Time: 0.003 s
Elapsed Time: 0.004 s
Elapsed Time: 0.004 s
Elapsed Time: 0.046 s
Elapsed Time: 0.133 s
Elapsed Time: 0.622 s
~/CW2>
```



The Central Server has 6GB memory and four 2GHz CPU cores. It will support four testings simultaneously. Use Ctrl-C to stop your program if it runs too long **- no more than 30 seconds!**
You are advised to use the **timeout** command to automatically kill your process if it runs more than 30 seconds on the benchmark server. Example:
~/CW2> **timeout 30** java -Xmx1024m TestClass < input\test1.txt > Output1.txt

./TestClass.bat (window) and ./TestClass.sh (linux/mac) are intended to be used on your local machines.
./TestClassSvr.sh is to be used on the benchmark server for testing and timing.
./TestClassSvr.sh and **./TestClass.sh** are the same except that there is a "**timeout 30**" command prepended for each test cases in **./TestClassSvr_CW2.sh**.

Tip 9: Marker will check and run each submitted coursework using **./TestClassSvr_CW2.sh** on the benchmark server. Please make sure your program runs as expected.

Tip 10: Run your program 10 times, select the best timing, and report it.

Release Date: 15 March, 2016.
Due Date: 26 April, 2016, 11:00
Late submission : 0 marks - in line with college of science policy.

Submission:

You should submit the following to the Blackboard coursework collection page:

1. A written report (***pdf ONLY***) to questions in Part1, Part2, and Part 3. Provide screenshots if you are able to achieve test cases #1-4/-6 on the benchmark server.

2. ***ONE*** working java source file.
The marker will use the ***submitted*** file to test against additional large test cases.

The java source file must be ***clearly commented.***
If your program does not work and the marker is unable to understand your code, no marks will be given.

Tip 11: Download your submitted file and verify its content. Make sure you submit the ***correct*** version of your java source file, and that you ***did*** submit it.

Tip 12: How do you achieve full marks? Read the marking assessment matrix.

Marking:

Part 1/Part2: You must show your steps to obtain the complexity of your algorithm in Big-O notation.

Part 2 : If your algorithm has the same or asymptotically slower complexity than that in Part 1, no more than 50% of the total marks will be given.

Part 3 :
If your program passes all **test cases #1-4**, and processes all test cases in under **10 seconds in total**, **90% of this part** will be awarded.

If your program passes all **test cases #1-6**, and processes all test cases in under **10 seconds in total**, **100% of this part** will be awarded.

Final tip:
If you struggle in this coursework, do not feel despair / panic. Convert any ideas into pseudocodes and discussion, and submit it to me. If they are in the right direction, partial marks may be awarded. (Same for examination! Do not leave it blank!)

Feedback
Feedback and sample solution will be provided 2 weeks after the submission deadline.