The Martians are planning to send you a long stream of *positive* integers.[1] Every once in a while the Martians will want you to figure out the $\lfloor \frac{k+2}{3} \rfloor$rd smallest of the k positive integers sent so far. They indicate this by sending a 0.

For EXAMPLE: Assume that the stream is

500, 100, 800, 1000, 1100, 300, 0, 900, 1200, 600, 200, 0, 700, 400

You should output 300 after seeing the first 0, since $\lfloor \frac{6+2}{3} \rfloor = 2$ (and 300 is the second smallest of the first six integers), and 500 after seeing the second 0, since $\lfloor \frac{10+2}{3} \rfloor = 4$ (and 500 is the fourth smallest of the first ten integers).

PART 1.

You will write a program for this problem. To keep things simple, the input will start with n, which is the number of values being sent (not including the 0's). So the input in the above example will actually look like

12 500 100 800 1000 1100 300 0 900 1200 600 200 0 700 400

For the program you hand in, you can assume that $n \leq 1000$.

We would like to process the data efficiently in real time. One way to do this is to have a maxheap with maximum size about $n/3$ and a minheap with maximum size about $2n/3$. After k values have arrived, the maxheap will have about the $k/3$ smallest values and the minheap will have about the $2k/3$ largest values. When a request for the $\lfloor \frac{k+2}{3} \rfloor$rd smallest value arrives, the value will be at the root of one of the two heaps, so that you can immediately figure out the desired value. When a new value arrives, it should be inserted into the "correct" heap. If one heap gets too large, a value will have to be deleted from that heap and inserted into the other heap.

PART 2.

We also want to do an experiment on how many COMPARISONS the algorithm does as a function of n. For this part, you will need to do experiments with $n > 1000$. Exactly how to do the experiment is up to you. You want to create random lists of n distinct values, which might as well be the numbers from 1 to n (since only their relative value matters). For each experiment, create a random permutation of the n numbers. You must write the permutation code yourself, i.e. do not use a library routine; you may use the random number generator provided. Let $C(n)$ be the average number of comparisons for your experiments of size n. Since we are expecting $C(n)$ to grow as $n \log n$, we expect $D(n) = C(n)/(n \log n)$ to look like a constant. Using this value, we can estimate the exact high order term for $C(n)$. For the following you should choose an appropriate base for the logarithm.

- Make a table with n, $C(n)$, and $D(n)$.
- Graph n versus $C(n)$.
- Graph n versus $D(n)$.
- Graph $\log(n)$ versus $C(n)$.
- Graph $\log(n)$ versus $D(n)$.
- Give a formula that estimates the number of comparisons for a problm of size n. If your experiments do not seem to converge, do the best you can.

[1] These values will somehow help you in building efficient comparison networks.

If you like, there is a wide variety of ways to extend the assignment. For example, you might try to figure out how many comparisons it takes to rebalance the two heaps during the entire process.

SUBMISSION GUIDELINES.

You will implement programs for both parts in JAVA and submit it on Gradescope. A skeleton code is provided on the course website and you must build your own program upon it.

To start, if you use Eclipse, you can create a new project and copy the folder `cmsc351f18` into `src` under your project folder. If you use commandline tools, you can compile the code by `javac cmsc351f18/*.java` and execute Main by `java cmsc351f18.Main`

You should not change the name of `MartianOracle.java` and `PermutationGenerator.java`. You can, however, create new classes and new JAVA file as you see appropriate. When creating new classes, make sure all of them lie in the same package `cmsc351f18` by adding `package cmsc351f18;` at the top of each file. Moreover, you should not change the name and signature of methods that are already in the skeleton code, but you can create your own helper functions and class members.

To help you test your code, we've included a `Main.java` to read spaced-separated numbers from `stdin` and calls correspondent functions in `MartianOracle.java`.

When you implement `PermutationGenerator.java`, you should use `m_random` defined in the constructor as your source of randomness.

After you finish, pack all your `.java` files in a zip archive and submit it under 'Martian Numbers - Code'. After you submit the code, make sure that your code compiles by checking the autograder output on Gradescope. In addition, submit a pdf file containing experiment results from PART 2 under 'Martian Numbers - Writeup'.