It needs to be REALLY fast!



A space-exploration Java coding challenge sought a rapid algorithm to assess the amount of radiation impinging on a square sensor. The sensor would “white out” if exposed to a critical amount of radiation, and we need a rapid exposed-area determination in order to invoke a protective filter in real time.

We will learn how the development team explored algorithms and exploited Test Driven Development (**TDD**) to create and test a variety of algorithms. We will also learn how to organize algorithm variants by trade-offs to optimize section. Let’s start down the path of discovery as we work toward space exploration. A reference section is included with URLs to all code and run logs.

## The Sensor Problem

We have a square radiation sensor with an overlaid grid of exposure regions, with the lower left grid as the origin. The positive X direction, to the right of the origin, is labeled 0 through 1000. Similarly, the positive Y direction is upward from the origin, and is also labeled 0 through 1000. The grid uses arbitrary units and defines a logical partition of the sensor area (e.g., microns.)

We validate the sensor using a radiation generator that emits rapid, random bursts of radiation falling on a rectangular region of the sensor. A sensor in use is expected to receive no more than a million bursts a second. The measurement period will encounter **N** total radiation bursts. If the accumulated bursts reach or exceed a threshold of **K** bursts over a monitoring period, then the sensor will “white out” for a period of time. For example, more than two bursts per millisecond in a grid square temporarily “blind” that square.

Ideally, we will be able to compute the exposure level by area (bursts impinging on a sensor region) in sufficient time for the filter to be deployed. For example, we need to deploy the filter if we detect that 60% (600K squares) of the exposed sensor area has reached or exceeded the threshold of **K** bursts of radiation after **N** exposures.

## Expected Results of Investigation

Space agencies have two important concepts: verification and validation. For purposes of this problem, verification is showing the functional requirements are met. That is, we are able to correctly assess exposed areas and levels of impinging radiation. Validation is the effort to show that the algorithm is both fast enough, and uses acceptable levels of memory, to solve the problem during an actual mission.

We will deliver:

1. Algorithms implemented as a Java function accepting a list of exposed rectangles (length **N**) and thresholds (**K** bursts in a grid.)
2. A “Big Oh” analysis of time complexity.
3. A validation of the algorithm in a test environment.

## Verification and Validation Tools

We will use unit tests with specific test cases, each with a known result, to verify our algorithms. We will create a timing mechanism to validate the execution time (wall clock.)

## References

The code, run-logs, and documentation are all located in a GitHub repo at xxxxxx. A Maven project is located in the repo at