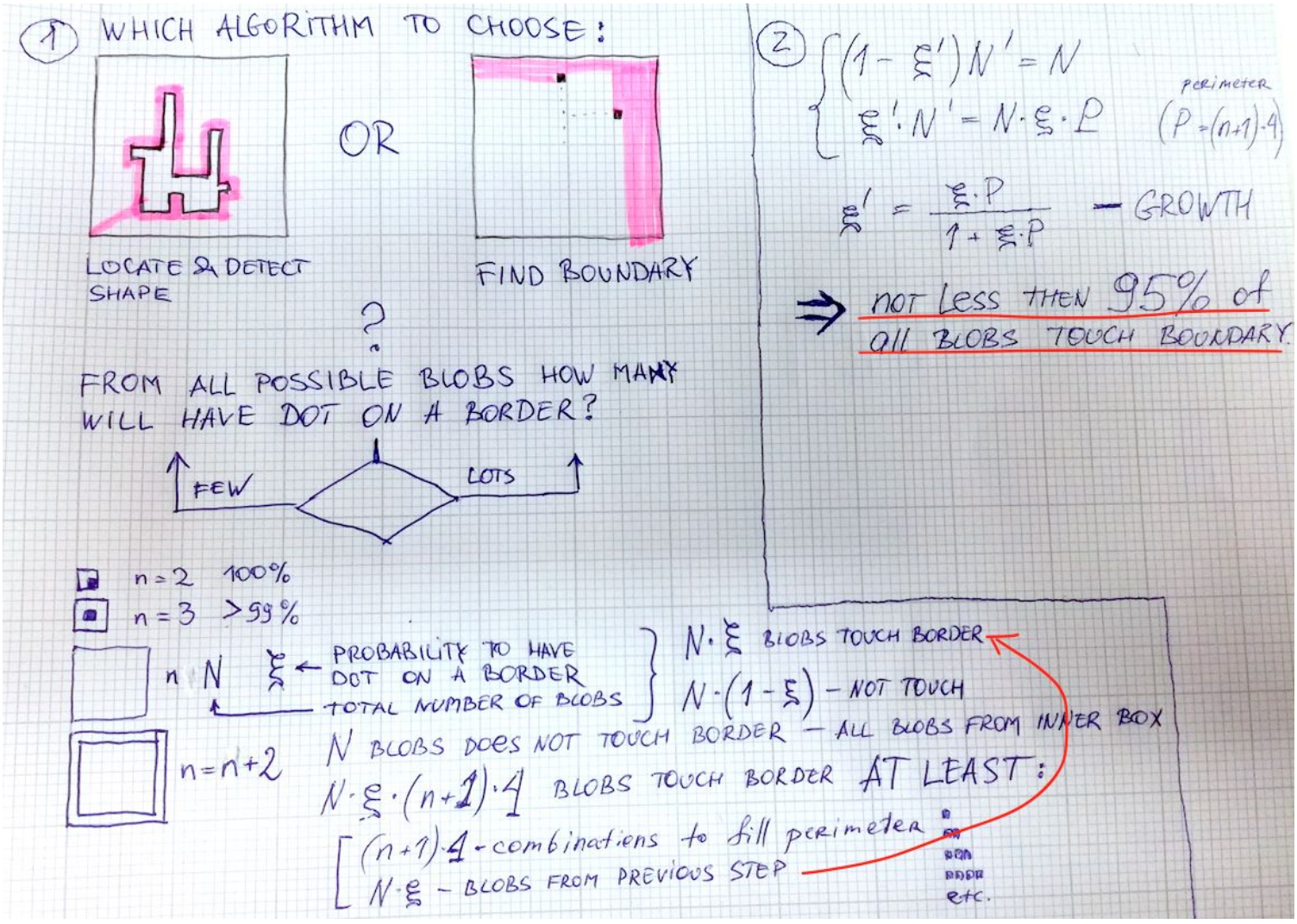
BlobBoundary solution

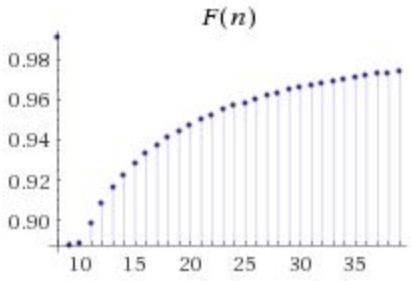
Andrey_Zorin@epam.com

1. Problem analysis

It is crucial to understand typical shape of the Blob for right decision on applicable algorithm: for small compact Blobs, surrounded by space, we would use “locate and walk around” approach, for filled up spaces - it is cheaper to just scan borders to find first dot. Which kind of Blob is more probable, given the definition of Blob generative process - **uniformly selected at random from the set of all possible Blobs?**



Looking at problem recursively, I derived probability to find **at least one dot** on the border of the Field, from the such probability for Field of smaller size. To do so, I estimated number of all possible Blobs touching border in Field of size (n+2) as number of Blobs touching border for inner Field (size n), multiplied by number of ways to grow onto perimeter from one dot touching this perimeter. In calculations I use pessimistic estimation of P - for one dot touching perimeter we can grow up to P dots on the perimeter, generating a valid Blob on each step.



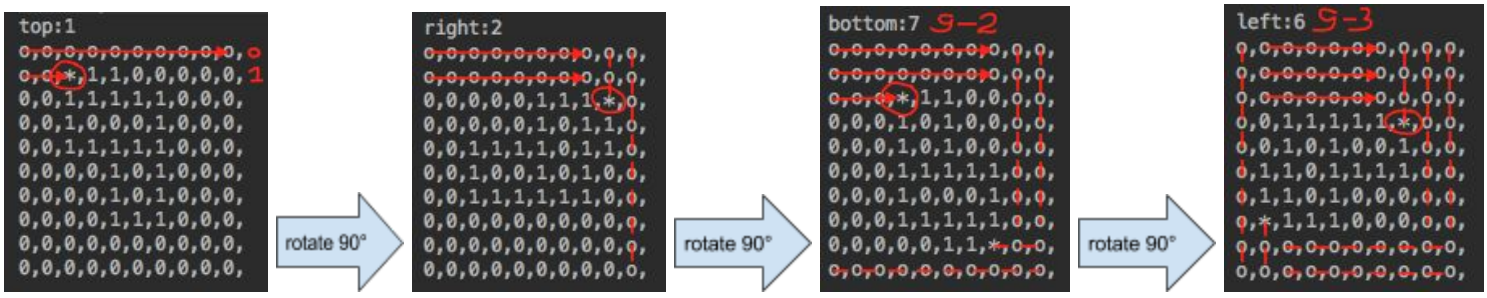
F = ξ = probability to have dot on the border
n = P = perimeter, or linear size of the field

Raw estimations gives probability above 95% for P=36 (perimeter of 10x10 Field). Now it is proven that more than 95% of Blobs have at least 1 dot on perimeter, and accepted as **working hypothesis** that all 4 sides of the Field are likely to have dots.
It is understood that Variability of Blobs grows faster than factorial[1], thanks to number of combinations on outermost layer. With high degree of certainty, answer can be given without writing any code, true random Blob has following boundary:

Top:0
Right:0
Bottom:9
Left:9

2. Solution

Only reading original data is penalised, so I use simple geometrical abstractions - for given Field I calculate depth of first dot, then rotate field 90 degree clockwise and repeat. No need to visit dots that are already opened, and go deeper then known positive result.



This algorithm is suboptimal for bigger arrays, both in memory and CPU, and parallelized poorly, but quick to write and easy to test. My implementation, technically, reads original data (but not analyzes) every time Field is rotated, those reads are not count toward score. Also for test readability it uses integer, not boolean.

Source code can be seen on github: https://github.com/helioraptor/BlobFinder

My program performs not as good on provided data as requested (67 against 44), and test fails, but again, meet such Blob is very improbable. For realistic cases it will be done in **less than 36** steps, which is coincidentally the best possible score for walk-around algorithm.

[1] Must be proven formally on face-to-face interview