

Searching a Matrix

This is a **regular task**. You must submit a PDF, which can be produced using the L^AT_EX template on Moodle, exported from a word processor, hand-written or any other method.

In this question, a matrix is said to be sorted if each row and column is strictly increasing. The following is an example of a sorted matrix.

$$\begin{bmatrix} 2 & 3 & 6 & 8 \\ 4 & 5 & 7 & 10 \\ 6 & 8 & 9 & 13 \\ 9 & 11 & 12 & 15 \end{bmatrix}$$

Design an $O(n)$ algorithm that takes an $n \times n$ sorted matrix M and a value k , and determines whether M contains k .

Hints:

- Binary search maintains that if the target value exists in the array, it must be in the part of the array we are currently searching (between l and r). Can you use a similar approach here?
- It not viable to start at $(1, 1)$. Why?
- Try to eliminate n entries each step.

Rubric.

- Your algorithm should give a ‘yes’ answer if k is an entry in M , and ‘no’ otherwise.
- As always:
 - You *must* justify that your algorithm produces the correct answer, i.e. responds with ‘yes’ if the value is present and ‘no’ otherwise. **In particular**, if your algorithm decides that k is or isn’t in a certain part of the matrix, you must argue why this is the case with respect to the sorting.
 - You *must* also justify that your algorithm runs in $O(n)$ time.
 - Your algorithm must be written in English, not code or pseudocode, and not transcribed from code or pseudocode.

Expected response length: less than half a page.

Solution.

- consider top right corner $M(1, n)$ first. If $M(1, n) = k$, then we find the target, return ‘yes’
- else if $M(1, n) < k$, then target cannot be in current row, hence we eliminate the row.
- else if $M(1, n) > k$, then target cannot be in the rightmost column, hence we eliminate this column.
- Repeat the steps until target is found or we eliminate all entries, return ‘no’