

# Worksheet

# Search Space Reduction

---

## Summary

1. Subject: Algorithms
2. Level: Medium/Hard

## Fundamentals

For the most part this worksheet applies when our data is sorted. If our data is sorted **we should think “How can I apply binary search?”**. If there is no log in our final time complexity then we have a problem. We are not using the property of sorted data to its fullest extent.

## Search Spaces

At any point in our searching algorithm we must ask ourselves **“How can I reduce my search space while maintaining my invariances?”**. In the examples we saw during the workshop, our invariance was the fact that the matrix was sorted.

At every data point, we must make a decision that answers the question in red. In the case of binary search, the decision spaces are:

- Go left -> Lower values
- Go Right -> Higher values

Each decision must bring us closer to our solution **AND** reduce our search space.

## Searching Ascending Matrix Problem

This is one of the more difficult searching problems without understanding fundamentals of this topic. Given an ascending matrix:

1	4	7	11
8	9	10	20
11	12	17	30

Where the rows and columns are sorted in ascending order, but the rows are not continuous. Given a value  $k$  we must write a method that tells us whether  $k$  is in the matrix or not.

### Brute Force

Loop through all cells in the matrix  $O(\text{row} * \text{cols})$

### Approach 2





Binary search every row in the matrix  $O(\text{row} * \log(\text{cols}))$

### Approach 3

Think back to this idea of “decision spaces”. If we look carefully, notice how to increase in value we must move down or right. To decrease in value we must move up or left.

- Increase: Down, Right
- Decrease: Up, Left

We must now pick a starting position in our matrix such that it lets us express our entire decision space, the ability to increase or decrease in value. So let's enumerate over all the corners of the matrix and see what decisions we can make.

-  Top Left
  - Increase: Down
  - Increase: Right
-  Bottom Right
  - Decrease: Up
  - Decrease: Left
-  Top Right
  - Decrease: Left
  - Increase: Down
-  Bottom Left
  - Decrease: Up
  - Increase: Right

So there are two possible valid starting positions in the matrix: top right corner or bottom left corner since we can choose to increase or decrease in value at any step in our algorithm. Coding this algorithm is left as an exercise for you. The final time complexity of this algorithm is  $O(\text{row} + \text{col})$ .