# Binary Search Advanced Problems

| | |
|---|---|
| Created | @July 27, 2022 8:12 PM |
| Class | |
| Type | |
| Materials | |
| Reviewed | ☐ |

## Minimax Searching Problems

In minimax Searching problem, we are generally expected to minimise the maximum of a value or maximise the minimum of a value.

SPOJ.com - Problem AGGRCOW

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000). His C (2 <= C <= N) cows don't

https://www.spoj.com/problems/AGGRCOW/

In the above given problem, we are expected to maximise the minimum distance between two cows. Hence a Minimax Searching problem it is.

Note: These minimax problems are different than the one mentioned in AI and related subjects.

And if at any point of time we see that we are trying to solve a minimax problem then Binary Search is very helpful in those cases.

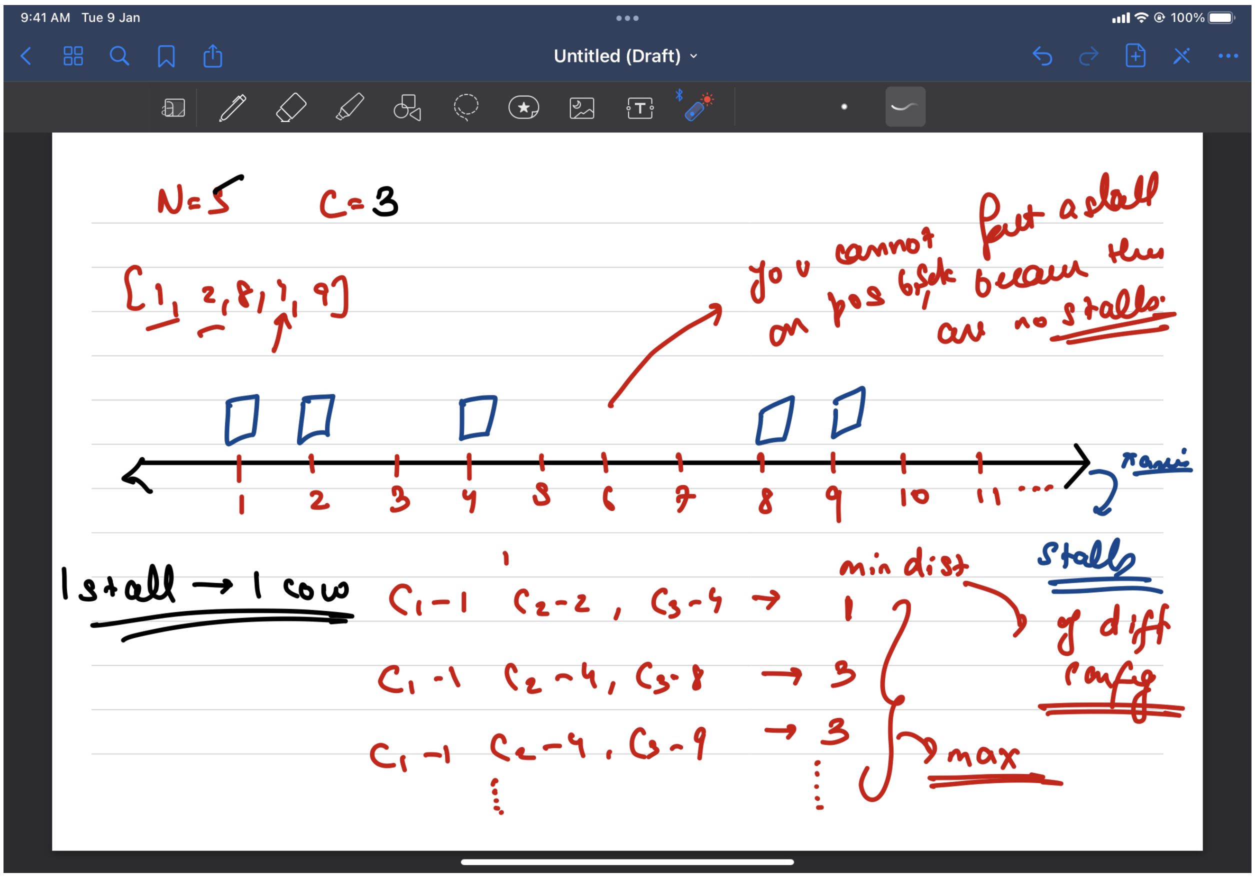
A lot of times these minimax problems are asked clubbed with binary search on answer.

# Problem 1:

SPOJ.com - Problem AGGRCOW

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000). His C (2 <= C <= N) cows don't

https://www.spoj.com/problems/AGGRCOW/




In this problem, we have to maximise the minimum distance. By maximising the minimum distance we say that we have placed all the cows as far apart as possible.

There can be different configurations to place the cows in the stalls. In each configuration we can calculate min distance between any two adjacent cows in the configuration. Now among all of these minimum distance we have to return the distance which is maximum of all. We don't need the config, but only to return the value of maximised minimum distance between any two cows.

# Solutions:

## Brute Force ?

In the given problem, we have N stalls and C cows, so we can try all possible combinations ? Try to put the next available cow to all the next available stalls.

## How to optimise ?

Because it is a minimax problem, we can try first of all to reduce it to a binary search based problem. Binary search is a searching algorithm, and what is the quantity of ans we want to search for ? Distance is the quantity. If somehow we can define the range of minimum distances possible between any two cows then we can try to think this range as a search space and apply binary search.

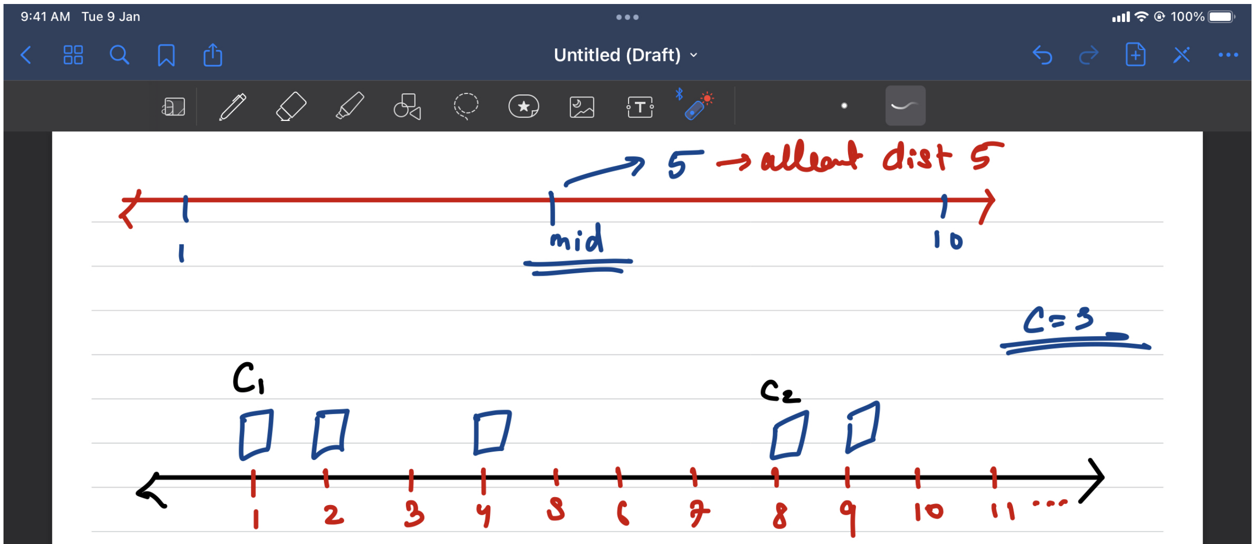
For N = 5, [1,2,8,4,9] → In the worst case what can be the minimum distance between any two cows for this given array ? Ans : 1

In the worst case what can be the maximum distance between two cows for this array ? 7

Calculating the range of the search space in few problems can be tedious. So what we can do is assume some very large and some very small valid values.

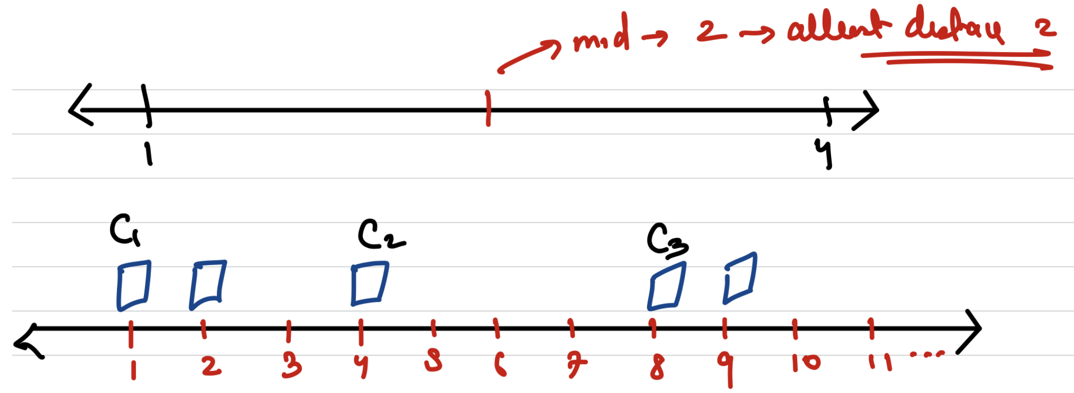
Range: [1, 10] → All those values that won't be possible in our ans will be eventually removed from the search space.

In this range, I will try to get a possible value say mid, and then check is it possible to place the cows such that minimum distance between any two cows is mid.

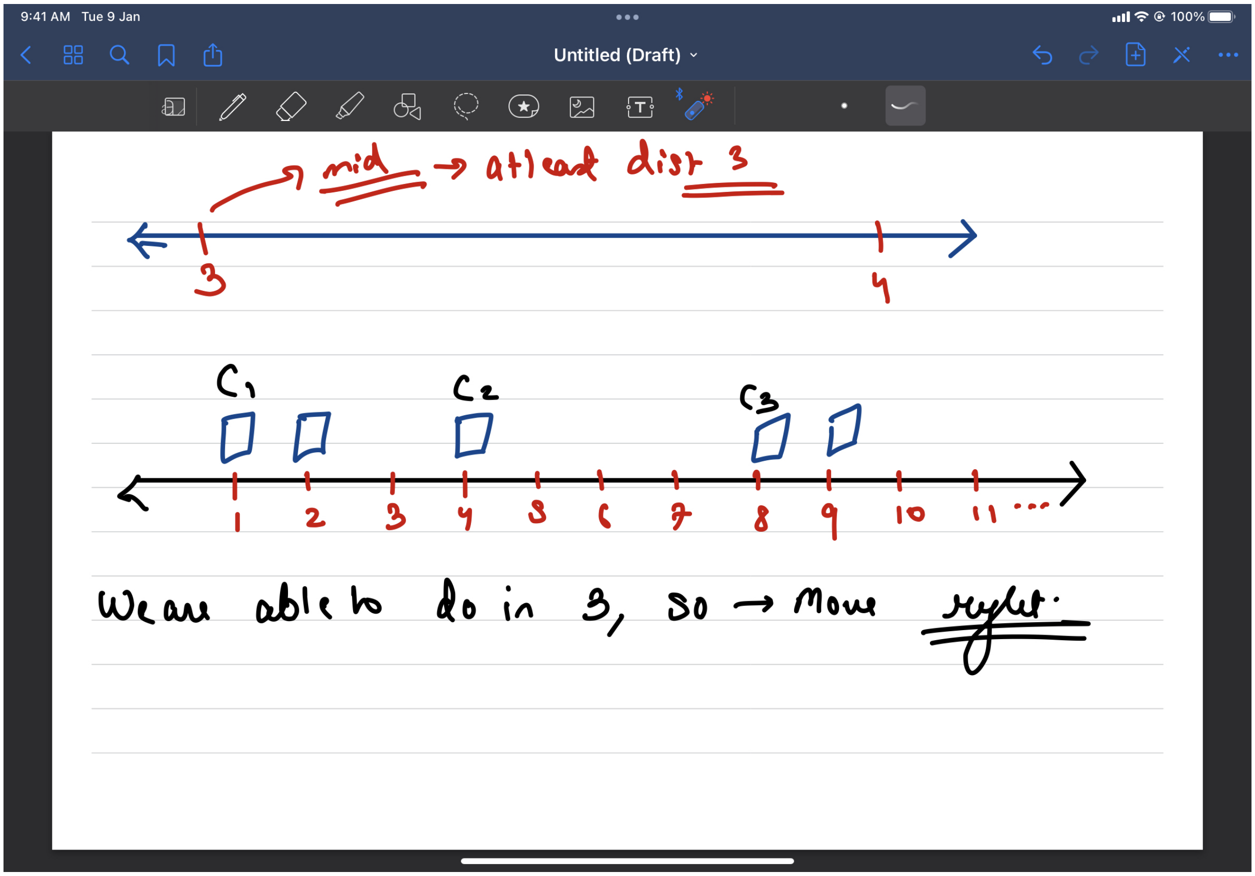


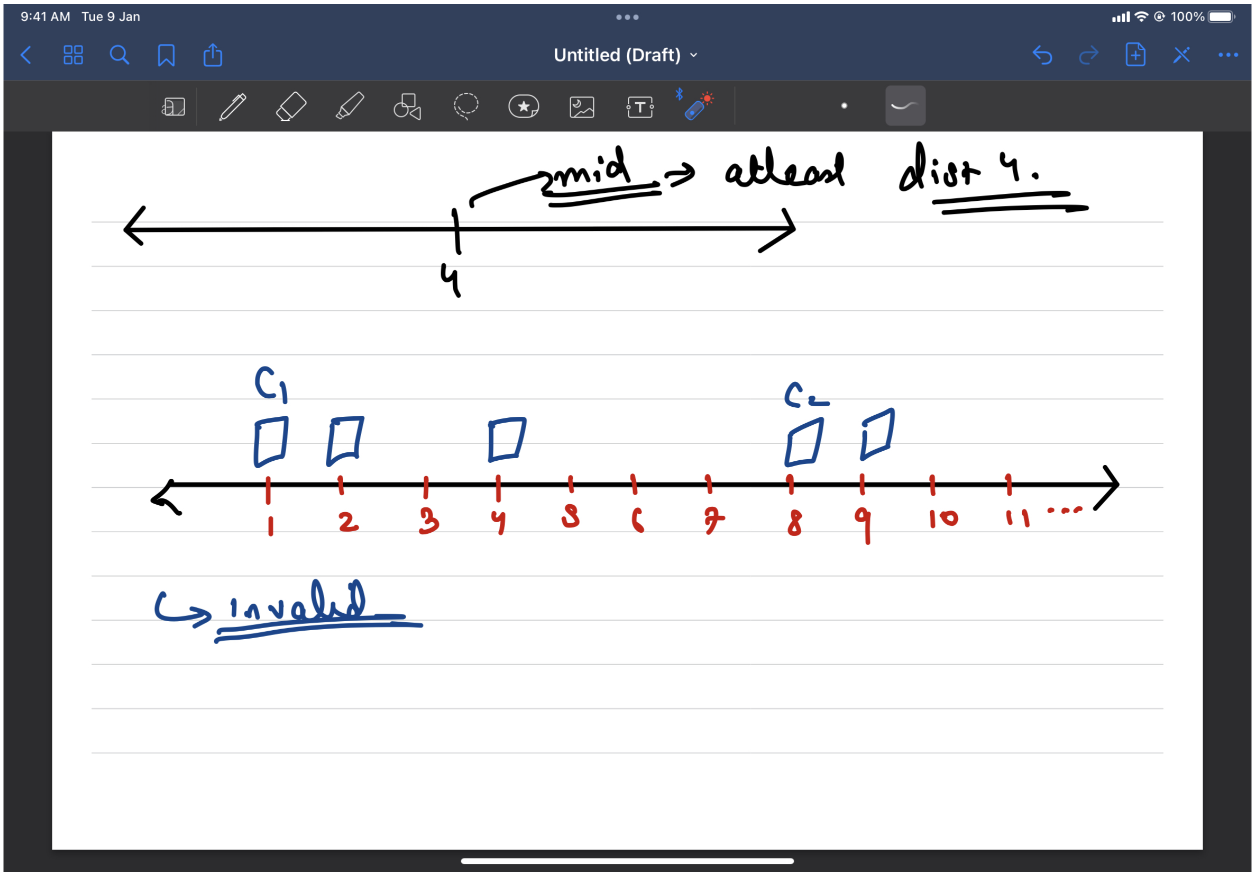

In this config we can't place cows.

If we can't maintain min dist 5, then anything greater than 5 is also not possible. → Move Left




So with atleast dist 2 between any 2 cows we are place all the cows. So we can place it in less than 2 also. → Move Right.


9:41 AM Tue 9 Jan
100%
Untitled (Draft)
mid → atleast dist 3
3
4
C1
C2
C3
1 2 3 4 5 6 7 8 9 10 11 ...
We are able to do in 3, so → move right.




```
function canPlaceCows(c, mid, stalls) {
  // Time: O(n)
  let lastPlacedPosition = 0;
  c--; // as first cow is already placed;
  for(let i = 1; i < stalls.length; i++) {
    if(stalls[i] - stalls[lastPlacedPosition] >= mid) {
      c--; // place one more cow
      lastPlacedPosition = i;
    }
    if(c == 0) return true; // we placed all the cows
  }
  // if we didn't place all the cows, then we will never retrun true from above
  return false;
}
function getMaximisedMinDist(stalls, c) {
// time: O(nlogn) Space: O(1)
  stalls.sort((a, b) => a-b); // sorting the stalls to iterate easily
  let lo = 1, hi = stalls[stalls.length - 1];
  let ans = 1;
  while(lo <= hi) {
    let mid = lo + Math.floor((hi - lo) / 2);
    if(canPlaceCows(c, mid, stalls) == true) {
      // if we are able to place the cows with atleast mid distance, then
      // mid is a possible ans, but we can find something greater than mid also
```

```
      ans = mid;
      lo = mid + 1;
    } else {
      hi = mid - 1;
    }
  }
  return ans;
}
```

# Problem 2: Book Allocation

[12,34,67,90]

- Case 1: S1 → 12, S2 → 34 + 67 + 90 = 191 → MAX = 191
- Case 2: S1 → 12 + 34 = 46, S2 → 67 + 90 = 157 → MAX = 157
- Case 3: S1 → 12 + 34+ 67 = 113, S2 → 90 → MAX = 113

The minimum value among all of the above max values is 113 → ans

## Observations:

What will be the maximum number of pages that can be read by any student ? → Sum of pages of all the books

In this problem we have to find the minimum value of the maximum pages that can be allocated to a student, so the search space should represent no of pages.

Upper limit of the search space: - sum of pages of all the books

Lower limit of the search space: - books[books.length - 1] → because this is the minimum max pages that a student can get.

```
function canAllocateBooks(mid, books, s) {
    // Time: O(n)
    // we will try to allocate books such that any student reads atmost mid page
    let students = 1;
    let currAllocatedPages = 0;// this is the current pages allocated to last student
    for(let i = 0; i < books.length; i++) {
      if(currAllocatedPages + books[i] > mid) {
        // we cannot allocate ith book to last student
        students++; // start allocating for the next student
        currAllocatedPages = books[i];// ith book goes to the new student
        if(students > s) return false;// we have less students
      } else {
```

```
        currAllocatedPages += books[i]; // give the book to the last student
      }
    }
    // if we never returned false from above that means allocation is possible
    return true;
}


function getMinimisedMaxValue(books, s) {
  // time: O(nlog(sum_of_all_pages))
  let lo = books[books.length - 1]
  let hi = 0;
  for(let i = 0; i < books.length; i++) hi += books[i];
  let ans = lo;
  while(lo <= hi) {
    let mid = lo + Math.floor((hi - lo) / 2);
    if(canAllocateBooks(mid, books, s)) {
      ans = mid;
      hi = mid - 1;
    } else {
      lo = mid + 1;
    }
  }
  return ans;
}
```

# Problem 3: Painter Partition

[10, 20, 30, 40]

P = 2

- Case 1: P1 → 10 , P2 → 20 + 30 + 40 = 90 → Total time: 90
- Case 2: P1 → 10 + 20 = 30, P2 → 30 + 40 = 70 → Total Time: 70
- Case 3 P1 → 10 + 20 + 30 = 60, P2 → 40 → Total time: 60

Case with the minimum time is case 3 so ans is 60 .

Do try to code it.