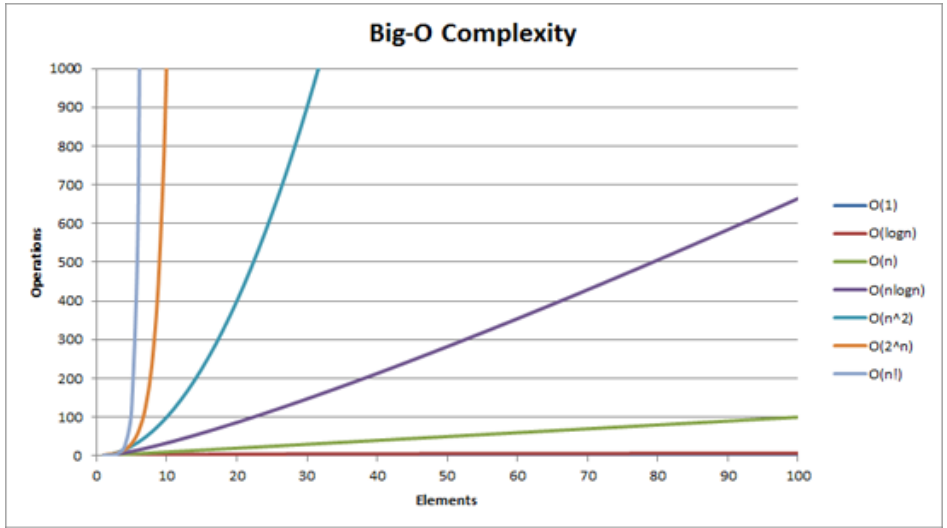
Optimization

2016-04-16

Motivation

- As problem size increase,
  the run time required may not increase linearly
- For example, the bubble sort algorithm
  requires O(N^2) comparisons



Optimization

- In competitive programming,
  we are not interested in speed ups of < 10x.
  - Usually the time limit is more than double of what required
- The goal is to reduce the complexity
  - e.g. from $O(N^2)$ to $O(N \lg N)$
  - e.g. from $O(QN)$ to $O(Q \lg N)$ to $O(Q)$
- Avoid linear scans
- Avoid repeated computation

Agenda

- Partial sum / difference array
  - (continuation of last week's)
- Precomputation
- Sliding Window (Two Pointers)
- Other techniques

Background

- Given an array of integers, find the sum of a contiguous section of the array.
- More formally, $q_i = (s_i, t_i),\ a_i = \sum_{j=s_i}^{t_i} x_j$

0	1	2	3	4	5	6	7	8	9
4	2	7	8	6	1	5	8	4	3

- si = 2, ti = 5, ai = 7 + 8 + 6 + 1 = 22
- si = 1, ti = 6, ai = 2 + 7 + 8 + 6 + 1 = 24

Solution 1

- Loop over Q
- Loop from si to ti to calculate sum

```cpp
#include <cstdio>
int x[10000];
int main() {
    int n, q, si, ti;
    scanf("%d %d", &n, &q);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x[i]);
    }
    for (int i = 0; i < q; i++) {
        int sum = 0;
        scanf("%d %d", &si, &ti);
        for (int j = si; j <= ti; j++) {
            sum += x[j];
        }
        printf("%d\n", sum);
    }
    return 0;
}
```

- What is the run time complexity?

Worst case test case

- N = 200000
  |Q| = 200000
- si = 0
- ti = N-1
- How many times will line 13 be run?

```
1  #include <cstdio>
2  int x[10000];
3  int main() {
4      int n, q, si, ti;
5      scanf("%d %d", &n, &q);
6      for (int i = 0; i < n; i++) {
7          scanf("%d", &x[i]);
8      }
9      for (int i = 0; i < q; i++) {
10         int sum = 0;
11         scanf("%d %d", &si, &ti);
12         for (int j = si; j <= ti; j++) {
13             sum += x[j];
14         }
15         printf("%d\n", sum);
16     }
17     return 0;
18 }
```

Partial sum

- We compute another array $y_i = \sum_{j=1}^{i} x_j$

	1	2	3	4	5	6	7	8	9	10
x[i]	4	2	7	8	6	1	5	8	4	3
y[i]	4	6	13	21	27	28	33	41	45	48

- $a_i = \sum_{j=s_i}^{t_i} x_j = (\sum_{j=1}^{t_i} x_j) - (\sum_{j=1}^{s_i-1} x_j)$
- To find the sum from x[2] to x[5], calculate y[5] – y[1] = 28 – 6 = 22

Partial sum

- What is the time complexity?

```cpp
#include <cstdio>
int x[10001], y[10001];
int main() {
    int n, q, si, ti;
    scanf("%d %d", &n, &q);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &x[i]);
        y[i] = y[i - 1] + x[i];
    }
    for (int i = 0; i < q; i++) {
        scanf("%d %d", &si, &ti);
        printf("%d\n", y[ti] - y[si - 1]);
    }
    return 0;
}
```

2D Partial sum

- Given a 2D array, we want to calculate the sum of a rectangular region

- $X = \begin{bmatrix} 1 & 5 & 2 & 5 & 4 \\ 8 & 0 & 1 & 6 & 0 \\ 2 & 3 & 7 & 3 & 5 \\ 4 & 5 & 1 & 4 & 0 \end{bmatrix}, q_i = (r_i, c_i, r'_i, c'_i).$

- $a_i = \sum_{j=r_i}^{r'_i} \sum_{k=c_i}^{c'_i} X_{j,k}$ e.g. ri = 1, ci = 3, ri' = 3, ci' = 5

ai = 2 + 5 + 4 + 1 + 6 + 0 + 7 + 3 + 5 = 33

Solution 1

- What is the run time complexity?

```cpp
#include <cstdio>
int x[1001][1001];
int main() {
    int n, m, q;
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            scanf("%d", &x[i][j]);
        }
    }
    for (int i = 0; i < q; i++) {
        scanf("%d %d %d %d", &r, &c, &rr, &cc);
        int sum = 0;
        for (int j = r; j <= rr; j++) {
            for (int k = c; k <= cc; k++) {
                sum += x[j][k];
            }
        }
        printf("%d\n", sum);
    }
    return 0;
}
```

Applying 1D partial sum

- We can compute another 2D array Y $Y_{r,i} = \sum_{j=1}^{i} X_{r,j}$

$$X = \begin{bmatrix} 1 & 5 & 2 & 5 & 4 \\ 8 & 0 & 1 & 6 & 0 \\ 2 & 3 & 7 & 3 & 5 \\ 4 & 5 & 1 & 4 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 6 & 8 & 13 & 17 \\ 8 & 8 & 9 & 15 & 15 \\ 2 & 5 & 12 & 15 & 20 \\ 4 & 9 & 10 & 14 & 14 \end{bmatrix}$$

$$a_i = \sum_{j=r_i}^{r'_i} \sum_{k=c_i}^{c'_i} X_{j,k} = \sum_{j=r_i}^{r'_i} ((\sum_{k=1}^{c'_i} X_{j,k}) - (\sum_{k=1}^{c_i-1} X_{j,k})) = \sum_{j=r_i}^{r'_i} (Y_{j,c'_i} - Y_{j,c_i-1})$$

Solution 2

- What is the run time complexity?

```cpp
#include <cstdio>
int x[1001][1001], y[1001][1001];
int main() {
    int n, m, q;
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            scanf("%d", &x[i][j]);
            y[i][j] = y[i][j - 1] + x[i][j];
        }
    }
    for (int i = 0; i < q; i++) {
        scanf("%d %d %d %d", &r, &c, &rr, &cc);
        int sum = 0;
        for (int j = r; j <= rr; j++) {
            sum += y[j][cc] - y[j][c - 1];
        }
        printf("%d\n", sum);
    }
    return 0;
}
```

One step further

$$a_i = \sum_{j=r_i}^{r'_i} (Y_{j,c'_i} - Y_{j,c_i-1}) = (\sum_{j=r_i}^{r'_i} Y_{j,c'_i}) - (\sum_{j=r_i}^{r'_i} Y_{j,c_i-1})$$

- We can compute another 2D array Z

$$X = \begin{bmatrix} 1 & 5 & 2 & 5 & 4 \\ 8 & 0 & 1 & 6 & 0 \\ 2 & 3 & 7 & 3 & 5 \\ 4 & 5 & 1 & 4 & 0 \end{bmatrix} \qquad Y_{r,i} = \sum_{j=1}^{i} X_{r,j} \qquad Z_{i,c} = \sum_{j=1}^{i} Y_{j,c}$$

$$Y = \begin{bmatrix} 1 & 6 & 8 & 13 & 17 \\ 8 & 8 & 9 & 15 & 15 \\ 2 & 5 & 12 & 15 & 20 \\ 4 & 9 & 10 & 14 & 14 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 6 & 8 & 13 & 17 \\ 9 & 14 & 17 & 28 & 32 \\ 11 & 19 & 29 & 43 & 52 \\ 15 & 28 & 39 & 57 & 66 \end{bmatrix}$$

2D Partial sum

$$a_i = \left(\sum_{j=r_i}^{r'_i} Y_{j,c'_i}\right) - \left(\sum_{j=r_i}^{r'_i} Y_{j,c_i-1}\right)$$

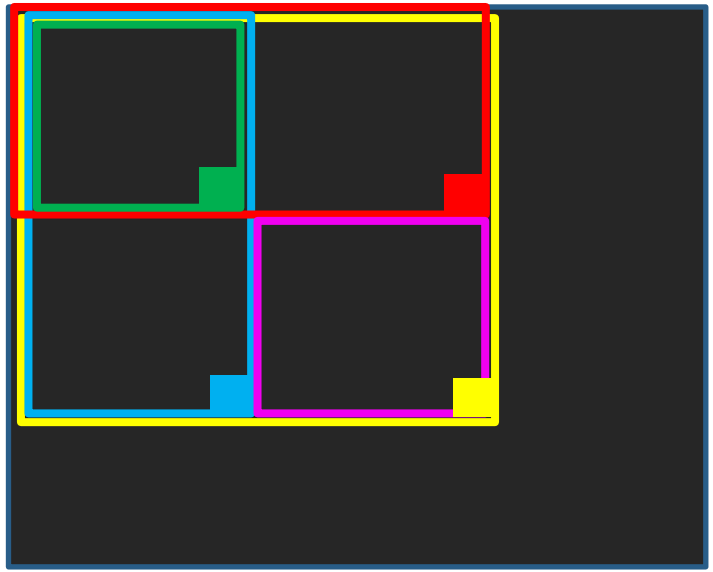
$$= (Z_{r'_i,c'_i} - Z_{r_i-1,c'_i}) - (Z_{r'_i,c_i-1} - Z_{r_i-1,c_i-1})$$

$$= Z_{r'_i,c'_i} - Z_{r_i-1,c'_i} - Z_{r'_i,c_i-1} + Z_{r_i-1,c_i-1}$$

$$Z = \begin{bmatrix} 1 & 6 & 8 & 13 & 17 \\ 9 & 14 & 17 & 28 & 32 \\ 11 & 19 & 29 & 43 & 52 \\ 15 & 28 & 39 & 57 & 66 \end{bmatrix}$$

Inclusion Exclusion

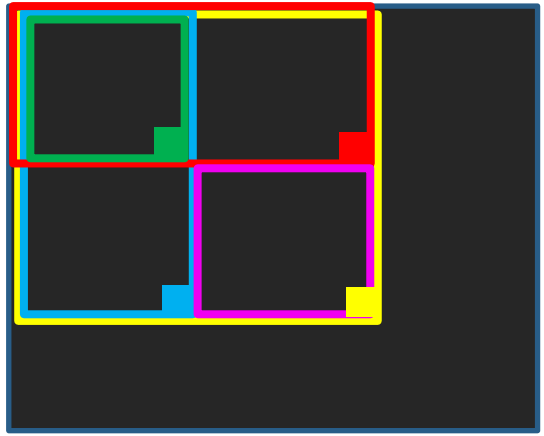
- Magenta = Yellow – Red – Blue + Green



Solution 3


```cpp
#include <cstdio>
int x[1001][1001], y[1001][1001], z[1001][1001];
int main() {
    int n, m, q;
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            scanf("%d", &x[i][j]);
            y[i][j] = y[i][j - 1] + x[i][j];
            z[i][j] = z[i - 1][j] + y[i][j];
        }
    }
    for (int i = 0; i < q; i++) {
        scanf("%d %d %d %d", &r, &c, &rr, &cc);
        printf("%d\n", z[rr][cc] - z[rr][c - 1] -
                       z[r - 1][cc] + z[r - 1][c - 1]);
    }
    return 0;
}
```

Creating the 2D partial sum directly



- `scanf("%d", &x);`
- `z[i][j] = _____;`

Difference array

- Partial sum is summation over the terms (like integration)

$$\int f(x)\ dx$$

- Next we will introduce difference array, which is like “differentiation”

$$\frac{df(x)}{dx}$$

Background

- There are many operations, operation Qi increases each element in a contiguous section of the array by some value.
- $q_i = (s_i, t_i, v_i)$
- $x_j := x_j + v_i$ for $j = s_i \ldots t_i$
- $q_1 = (3, 8, 2)$, $q_2 = (1, 4, 6)$

	0	1	2	3	4	5	6	7	8	9
After q_1	0	0	0	**2**	**2**	**2**	**2**	**2**	**2**	0
After q_2	0	**6**	**6**	**8**	**8**	2	2	2	2	0

Solution 1


```cpp
#include <cstdio>
int x[100001];
int main() {
    int n, q, si, ti, vi;
    scanf("%d %d", &n, &q);
    for (int i = 0; i < q; i++) {
        scanf("%d %d %d", &si, &ti, &vi);
        for (int j = si; j <= ti; j++) {
            x[j] += vi;
        }
    ...
    return 0;
}
```

Difference array

- $\Delta x_i = x_i - x_{i-1}$
- Note that this is the exactly the reverse of partial sum, where dx was the x and x was the y
- To increase the value of elements from si to ti, we add vi to dx[si] and substract vi from dx[ti + 1]

q_1 = (3, 8, 2)	0	1	2	3	4	5	6	7	8	9
dx[i]	0	0	0	**2**	0	0	0	0	0	**-2**
x[i]	0	0	0	2	2	2	2	2	2	0

q_1 = (1, 4, 6)	0	1	2	3	4	5	6	7	8	9
dx[i]	0	**6**	0	2	0	**-6**	0	0	0	-2
x[i]	0	6	6	8	8	2	2	2	2	0

Solution 2


```cpp
#include <cstdio>
int dx[100001];
int x[100001];
int main() {
    int n, q, si, ti, vi;
    scanf("%d %d", &n, &q);
    for (int i = 0; i < q; i++) {
        scanf("%d %d %d", &si, &ti, &vi);
        dx[si] +=  vi;
        dx[ti + 1] -= vi;
    }
    for (int i = 1; i <= n; i++) {
        x[i] = x[i - 1] + dx[i];
    }
    ...
    return 0;
}
```

2D Difference array

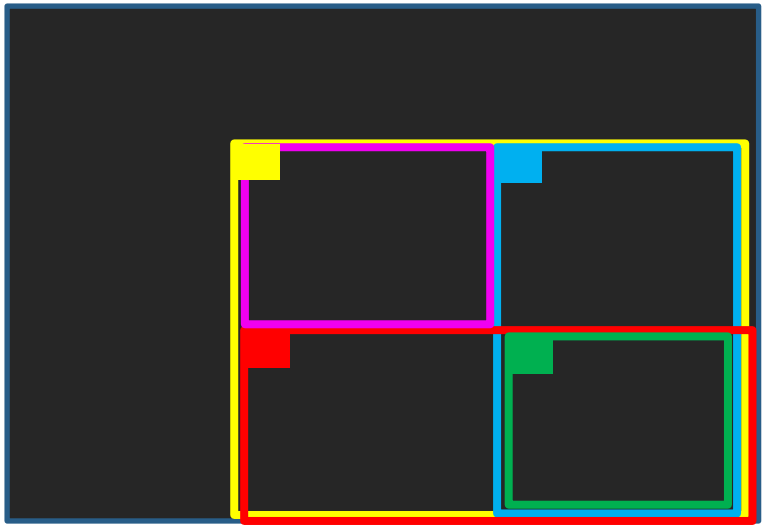
- Similar to 2D partial sum, we can have 2D difference array

$$X = \begin{bmatrix} 1 & 6 & 8 & 13 & 17 \\ 9 & 14 & 17 & 28 & 32 \\ 11 & 19 & 29 & 43 & 52 \\ 15 & 28 & 39 & 57 & 66 \end{bmatrix} \quad X^{(c)} = \begin{bmatrix} 1 & 6 & 8 & 13 & 17 \\ 8 & 8 & 9 & 15 & 15 \\ 2 & 5 & 12 & 15 & 20 \\ 4 & 9 & 10 & 14 & 14 \end{bmatrix}$$

$$X^{(rc)} = \begin{bmatrix} 1 & 5 & 2 & 5 & 4 \\ 8 & 0 & 1 & 6 & 0 \\ 2 & 3 & 7 & 3 & 5 \\ 4 & 5 & 1 & 4 & 0 \end{bmatrix}$$

2D Difference array

- Magenta = + Yellow − Red − Blue + Green



Precomputation

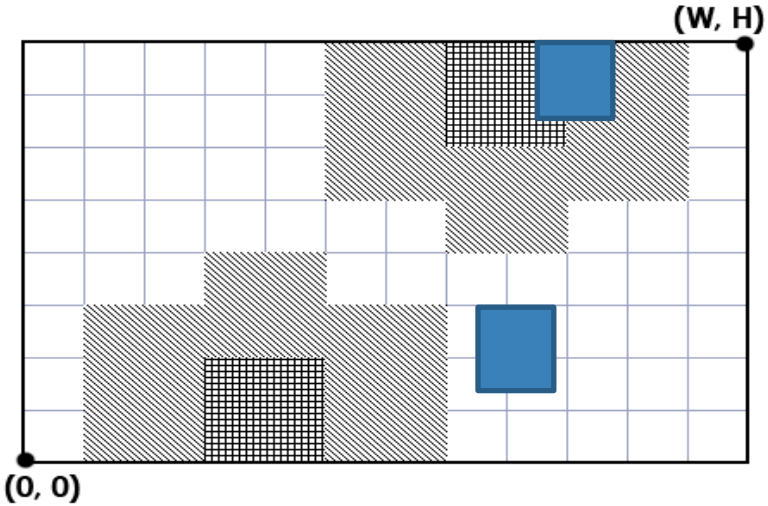
- If each query takes $O(N^2)$ time,
  we may try to reduce it to O(N lg N), O(N),
  O (lg N) or O(1) by using special data structures
- Use an efficient way to partially compute
  the result of the queries

J134 Lucky Rainbow

- A square coin of size L x L is thrown at (xi, yi), what prize would you win?
- Complexity without optimization: O(Q x L^2)

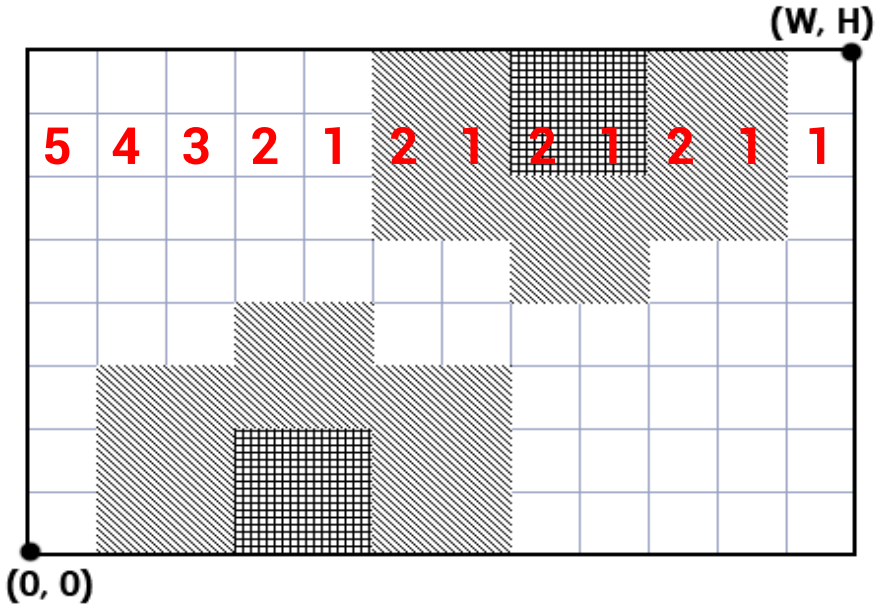
In all test cases,

- $1 \leq W, H \leq 1000$
- $1 \leq P \leq 25$
- $1 \leq L \leq 100$
- $1 \leq R \leq 10000$
- $1 \leq T_i \leq 1000$
- $-100 \leq X_i \leq W, -100 \leq Y_i \leq H$
- X_i and Y_i are real numbers.



Solution 1

- For each cell, count the number of cells on the **right** that have the same color
- After that, check that each row:
  - Has same letter
  - Has count >= L
- Precomputation cost $O(N^2)$
- Complexity becomes $O(QL)$



Solution 2

- Notice that for a coin to win a prize, all the cells must be of the same color
- Sounds like 2D partial sum?
- The sum must be
  - L x L for integer, integer
  - L x (L+1) for integer, real
  - (L+1) x (L+1) for real, real

Other useful arrays to be precomputed

- Prefix / suffix sum (partial sum) 前/後綴和
- Prefix / suffix max / min
- Prefix / suffix XOR sum
- Prefix / suffix count
  - e.g. number of odd numbers
  - e.g. number of ‘A’s

Two Pointers

- When performing exhaustion, we want to avoid as many impossible combinations as possible
- Given two sorted arrays A and B, find the number of pairs i, j such that

$$A_i + B_j = C$$

- C = 20

1	5	8	10	12	14	15	18	25
3	6	6	7	13	14	15	18	19

Solution 1

- C = 20

1	5	8	10	12	14	14	18	25
3	6	6	7	13	14	15	18	19

- for i in 1 .. N
  for j in 1 .. N
  if (A[i] + B[j] = c)
  ans := ans + 1;
- Complexity: $O(N^2)$

Solution 2

- for i in 1 .. N
  binary search in B for (c – A[i])
- Complexity: O(N lg N)

Solution 3

A[i] + B[j] = C = 20

Observe that as we increase i
A[i] becomes larger
c - A[i] would become smaller
(that's what we want to find in B)

1	**5**	8	10	12	14	14	18	25
3	5	6	7	13	14	**15**	18	19

1	5	**8**	10	12	14	14	18	25
3	5	6	**7**	13	14	15	18	19

1	5	8	**10**	12	14	14	18	25
3	6	6	**7**	13	14	15	18	19

1	5	8	10	**12**	14	14	18	25
3	6	6	**7**	13	14	15	18	19

1	5	8	10	12	**14**	14	18	25
3	**6**	**6**	7	13	14	15	18	19

Implementation Hint

- Use while loop to check if we need to move the pointer
- To avoid index out of range, add a **stopper element** at the end of array B
- e.g. use 1-based indexing and set b[0] = -2147483648
- Alternative: consider a[n] … a[1] add stopper element b[n] = 1000000005

```
1 int j = n - 1;
2 for (int i = 0; i < n; i++) {
3     while (j >= 0 && a[i] + b[j] > c) {
4         j--;
5     }
6     if (j >= 0 && a[i] + b[j] == c) {
7         count++;
8     }
9 }
```

This works only if the numbers are distinct!!!

M0652 Museum

- Find the shortest consecutive segment that contains all the numbers 1 .. M

12 5

2 5 3 1 **3 2 4 1 1 5** 4 3

In all inputs, $1 \leqslant N \leqslant 1000000$ and $1 \leqslant M \leqslant 2000$.

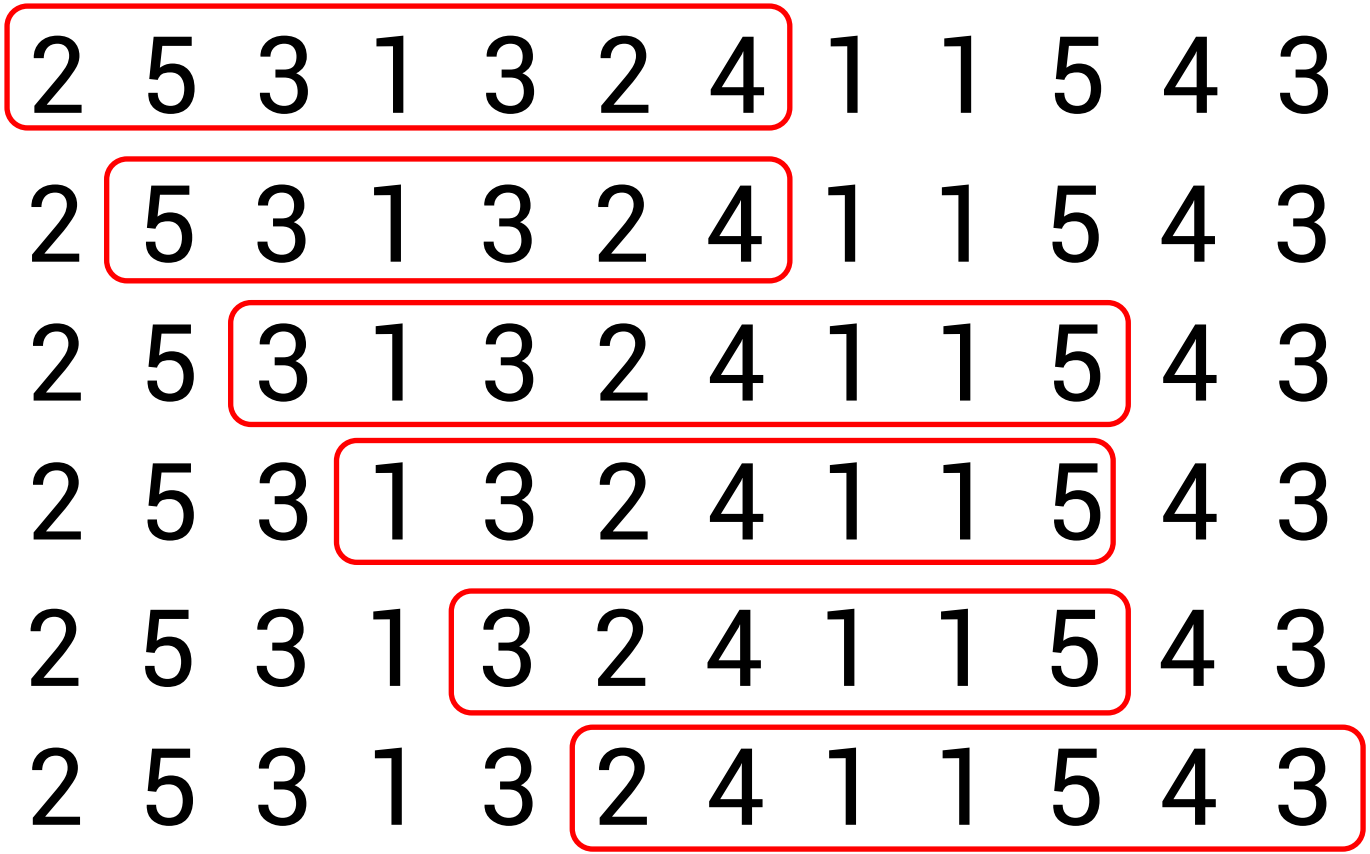
Additionally, in 50% of the inputs, $1 \leqslant N \leqslant 10000$.

Two pointers

- Maintain two pointers (指針) L and R
  - Here, we can have L = i (the loop counter) and R = j
- When we move L to the right by 1,
  move R to the right until the range still have all M numbers
  (this time both points work with the same array and
  both move in the same direction)

- Answer = shortest ***R − L*** ever encountered
- Complexity: O(N)

Two pointers



When to use two pointers?

- Input is two sorted array or one sorted array that references itself

- If you have a sorted and a unsorted array
  - Binary search on the sorted array … O(n lg n)

- If you have two unsorted arrays
  - Sort one of them and binary search … O(n lg n)

Other techniques

- Finding cycles
  - In simulation problems, you may have a solution that the run time depends on the answer
  - e.g. O(ans x N)
  - Identify cycles and come up with a formula for the cycle result
  - Simulate iterations before and after those cycles

Other techniques

- Batching steps (calculate answer directly)
  - Find the date of now + N days
    - Instead of iterating over days, iterate over months
    - Deduct Mi days from N
  - J144 Fair Santa Claus
    - Instead of increasing / decrease the values by 1 to reduce the difference by 2
    - Swap gifts to reduce the difference by 2 x B

Memory Optimization

- S152 Apple Garden
- There are some apples in a 2D array, find a M x M subarray that contains max number of apples
- If the coordinates are small, we can easily solve this using 2D partial sum

SUBTASKS

	Points	Constraints
1	40	$1 \leq N, M \leq 50, 1 \leq K \leq 30$
2	15	$1 \leq N, M \leq 300, 1 \leq K \leq 100$
3	15	$1 \leq N, M \leq 2000, 1 \leq K \leq 200$
4	30	$1 \leq N, M \leq 10^9, 1 \leq K \leq 2000$

Discretization

- Discretization (離散法) is a technique that converts values (not necessarily integers) into integers, while maintaining their relative order
- Example: 7654321, 123456, 934602, 123456789
  -> 3, 1, 2, 4 (or 2, 0, 1, 3)
- Put the values into an array, sort the array
  123456, 934602, 7654321, 123456789
- For each value in the original array,
  find its rank using **binary search**
- Note: it's better for the same value to be converted into the same number

Discretization

- Note that there are K <= 2000 trees, so there are at most 2000 different X coordinates and 20 different Y coordinates
- Build 2D partial sum
- Use two pointers to keep the size of rectangle selected <= M * M

		1	2	3	4	5
		15	**24**	**63**	**80**	**97**
1	**5**			**1**		
2	**66**		**1**			
3	**79**					**1**
4	**111**	**1**				
5	**200**				**1**	