Data Structures and Algorithms

(**CS210A**)

Semester I – **2014-15**

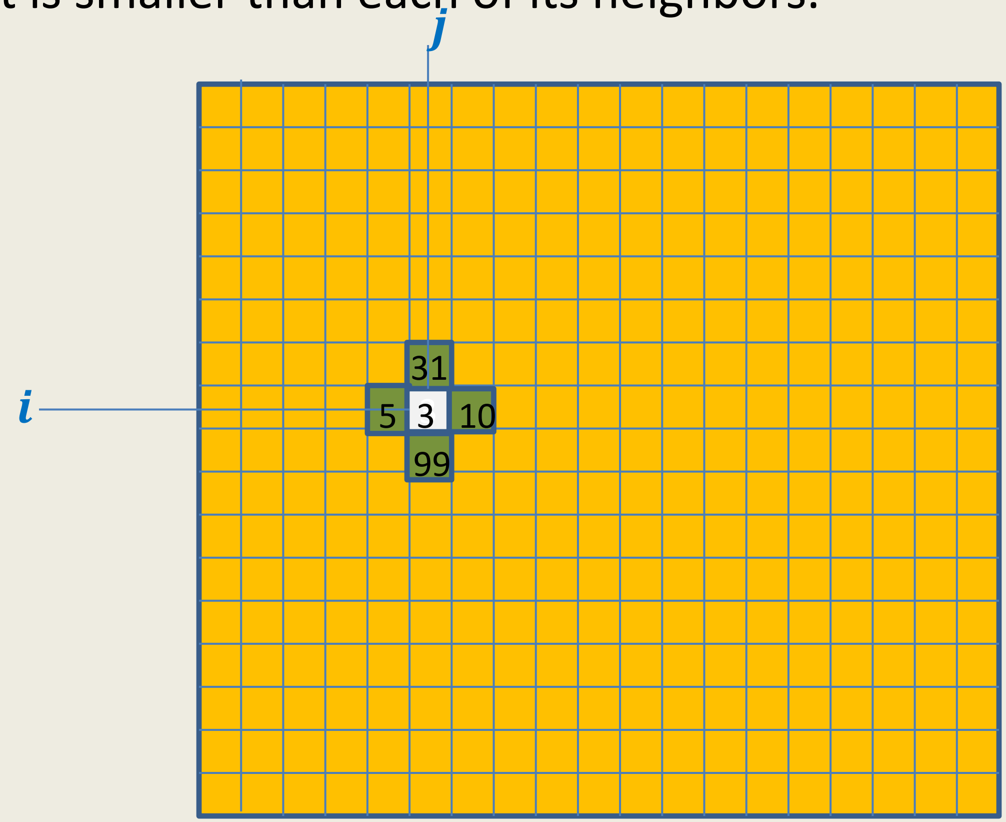
Lecture 5:

- Design of **O**(n) time algorithm for **Local Minima in a grid**
- A new problem : Data structure for **Range-minima problem**

LOCAL MINIMA IN A GRID

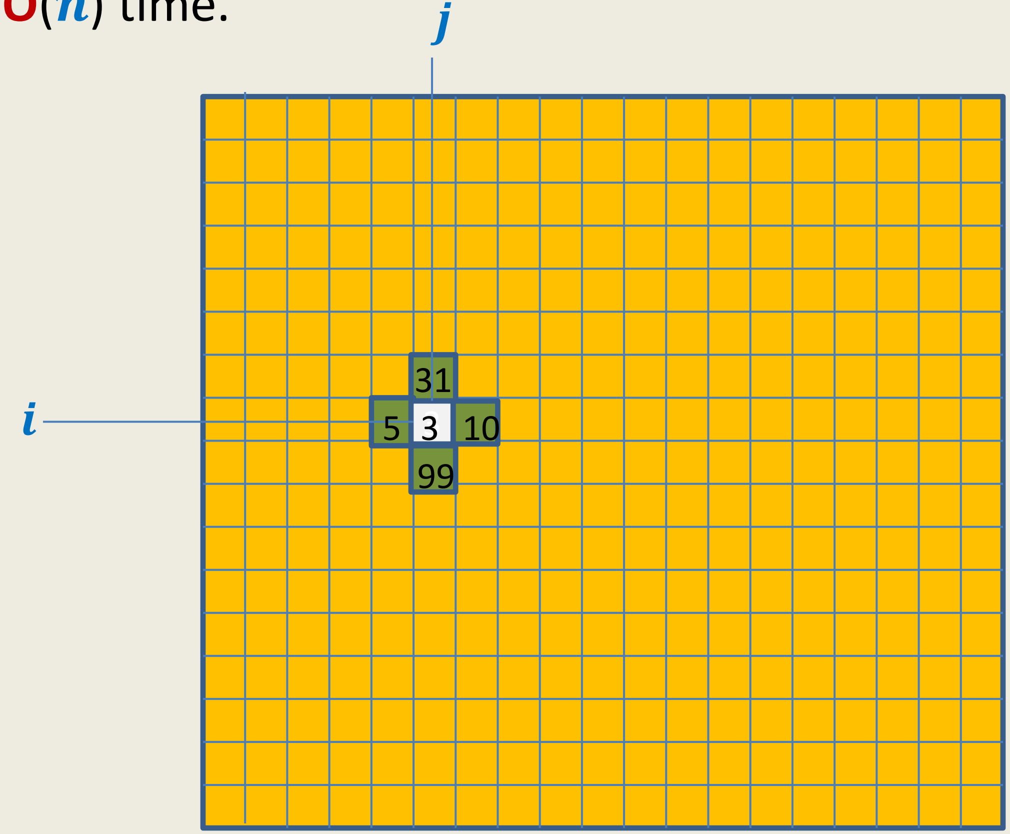
Local minima in a grid

Definition: Given a $n \times n$ grid storing <u>distinct</u> numbers, an entry is local minima if it is smaller than each of its neighbors.



Local minima in a grid

Problem: Given a $n \times n$ grid storing distinct numbers, output any local minima in **O**(n) time.



Two simple principles

1. Respect every new idea which solves a problem even partially.

2. Principle of simplification:

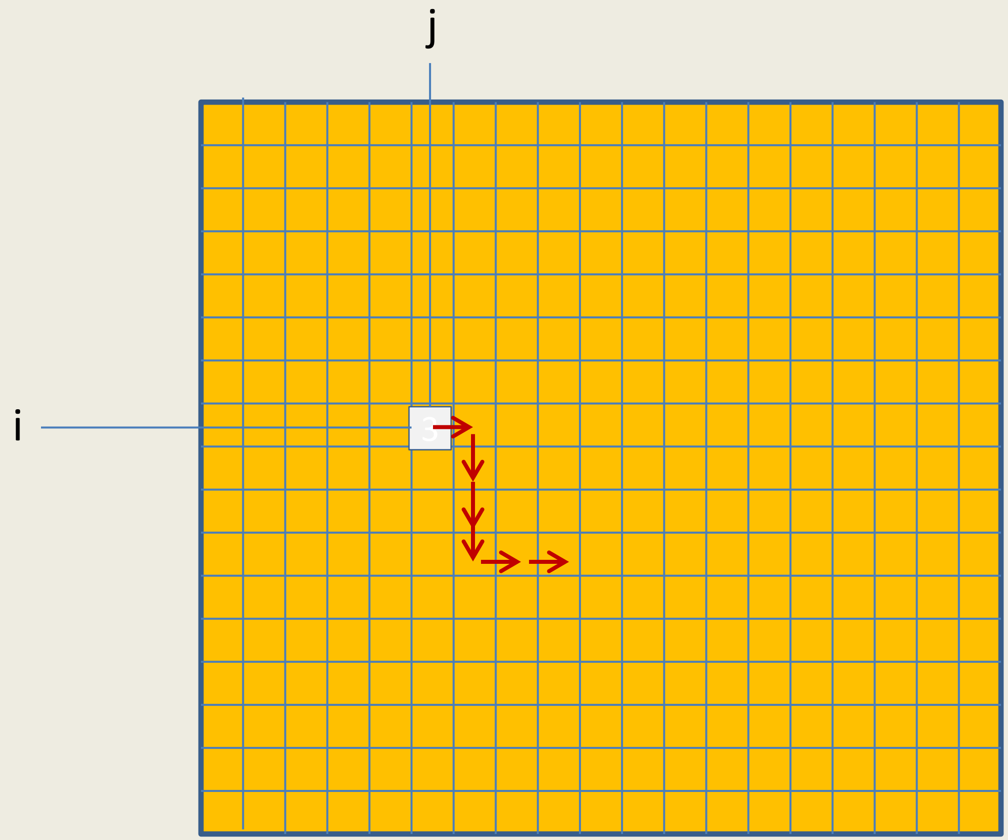
If you find a problem difficult,

➔ try to solve its simpler version, and then

➔ extend this solution to the original (difficult) version.

A new approach

Repeat : *if current entry is not local minima, explore the neighbor storing smaller value.*



A new approach

Explore()

{ Let c be any entry to start with;

While(c is not a local minima**)**

{

c $\leftarrow$ a neighbor of c storing smaller value

}

return c;

}

Question: What is the proof of correctness of **Explore** ?

Answer:

➔It suffices if we can prove that **While** loop eventually terminates.

➔Indeed, the loop terminates since **we never visit a cell twice**.

A new approach

```
Explore()
{    Let c be any entry to start with;
     While(c is not a local minima)
     {
         c ← a neighbor of c storing smaller value
     }
     return c;
}
```

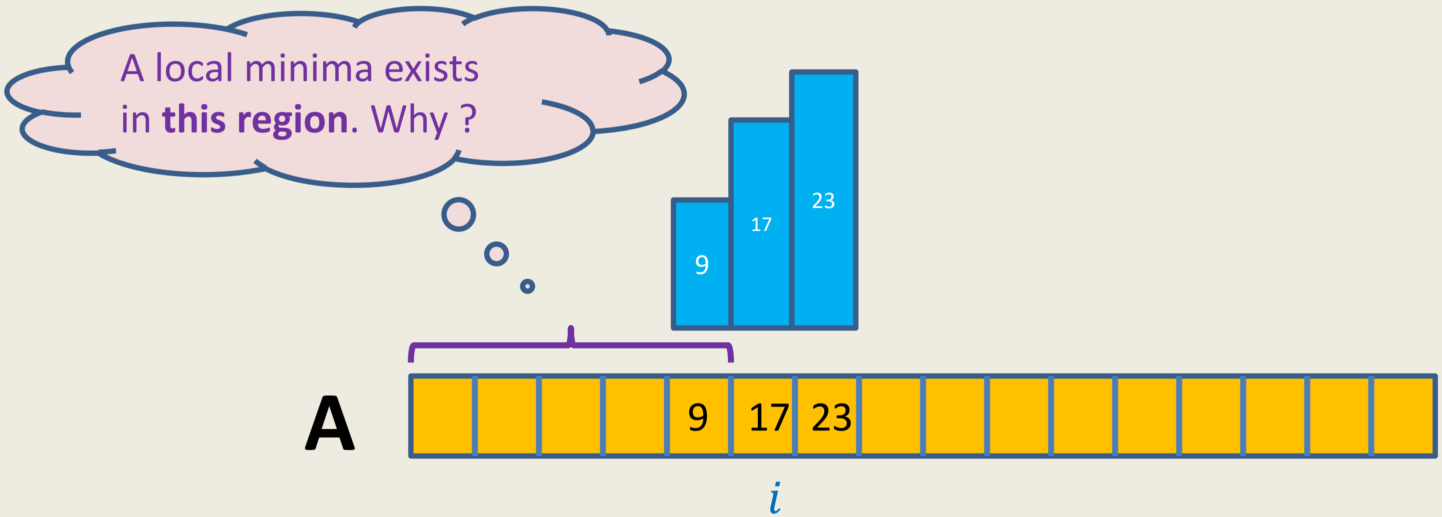
Worst case time complexity : **O**(n^2)

How to apply this principle ?

First principle:
Do not discard **Explore**()

Second principle:
Simplify the problem

Local minima in an array



Theorem: There is a local minima in **A**[0,…, $i-1$].

Proof: Suppose we execute **Explore()** from **A**[$i-1$].

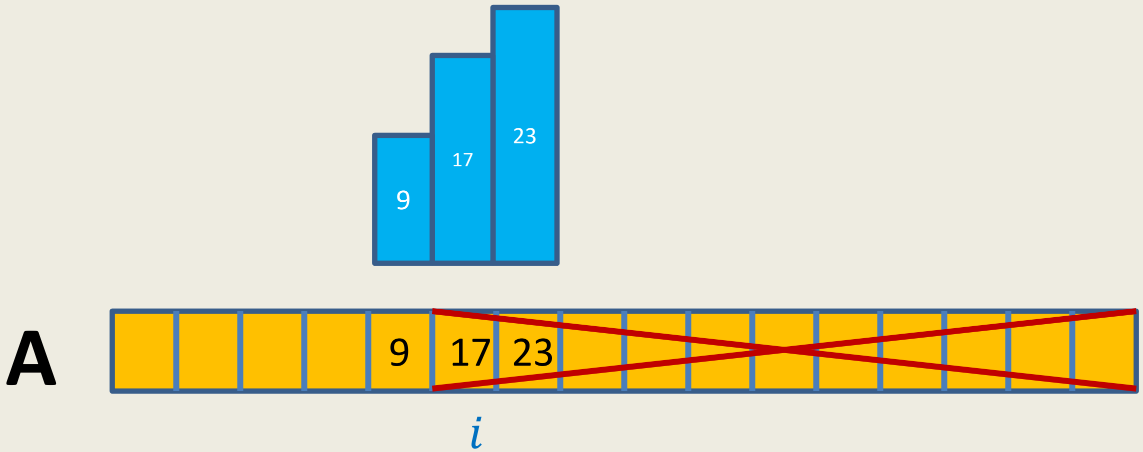
Explore(), if terminates, will return local minima.

It will terminate <u>without ever entering</u> **A**[i,…, $n-1$].

Hence there is a local minima in **A**[0,…, $i-1$].

Algorithmic proof

Local minima in an array



Theorem: There is a local minima in **A**[$0,\ldots, i-1$].

➔ We can confine our search for local minima to only **A**[$0,\ldots, i-1$].

➔Our problem size has reduced.

Question: Which i should we select so as to reduce problem size significantly ?

Answer: $middle$ point of array **A.**

Local minima in an array

(Similar to binary search)

int Local-minima-in-array(A) {

L ← 0;

R ← $n - 1$;

found ← FALSE;

while(not found)

{

mid ← (L + R)/2;

If (mid is a local minima)

found ← TRUE;

else if(A[$mid + 1$] < A[mid]) L← $mid + 1$

else R← $mid - 1$

}

return mid; }

➔ Running time of the algorithm = O(log n)

How many iterations ?

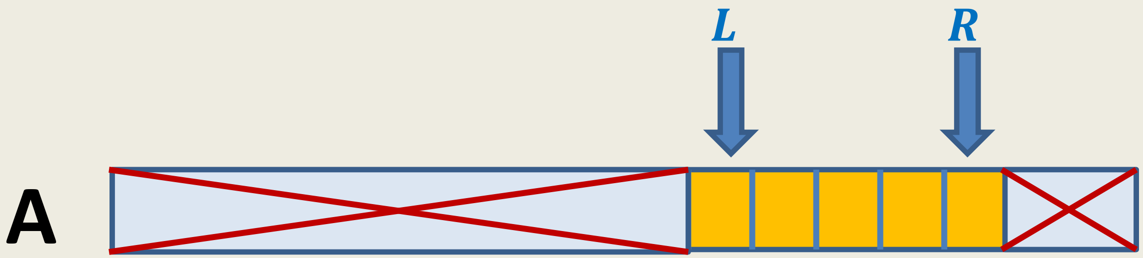
O(log n)

O(1) time in one iteration

Proof of correctness ?

Local minima in an array

(Proof of correctness)



Assertion: In the beginning of each iteration, the following assertion holds

"A local minima of array **A** exists in **A**[L,…, R]."

Question: How to prove the assertion ?

Hint: Express it **differently** .

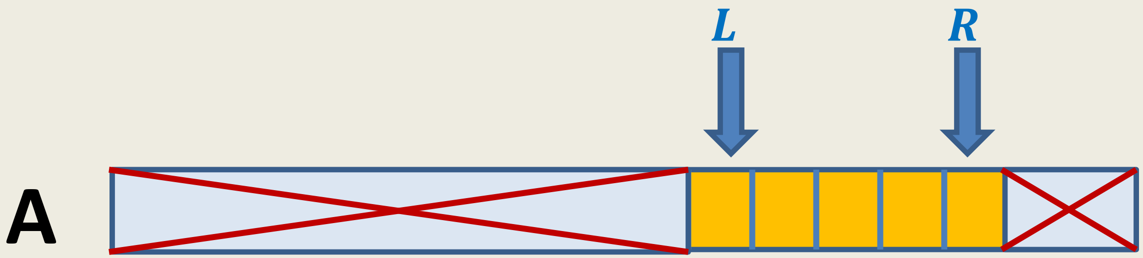
"**A**[L]< **A**[$L-1$]" and "**A**[R]< **A**[$R+1$]".

Homework: Make sincere attempts to prove the assertion.

Hint: mathematical Induction … focus on an iteration… use **Explore**() …

Local minima in an array

(Proof of correctness)



Assertion: In the beginning of each iteration, the following assertion holds

“**A**[L]< **A**[$L-1$]” and “**A**[R]< **A**[$R+1$].”

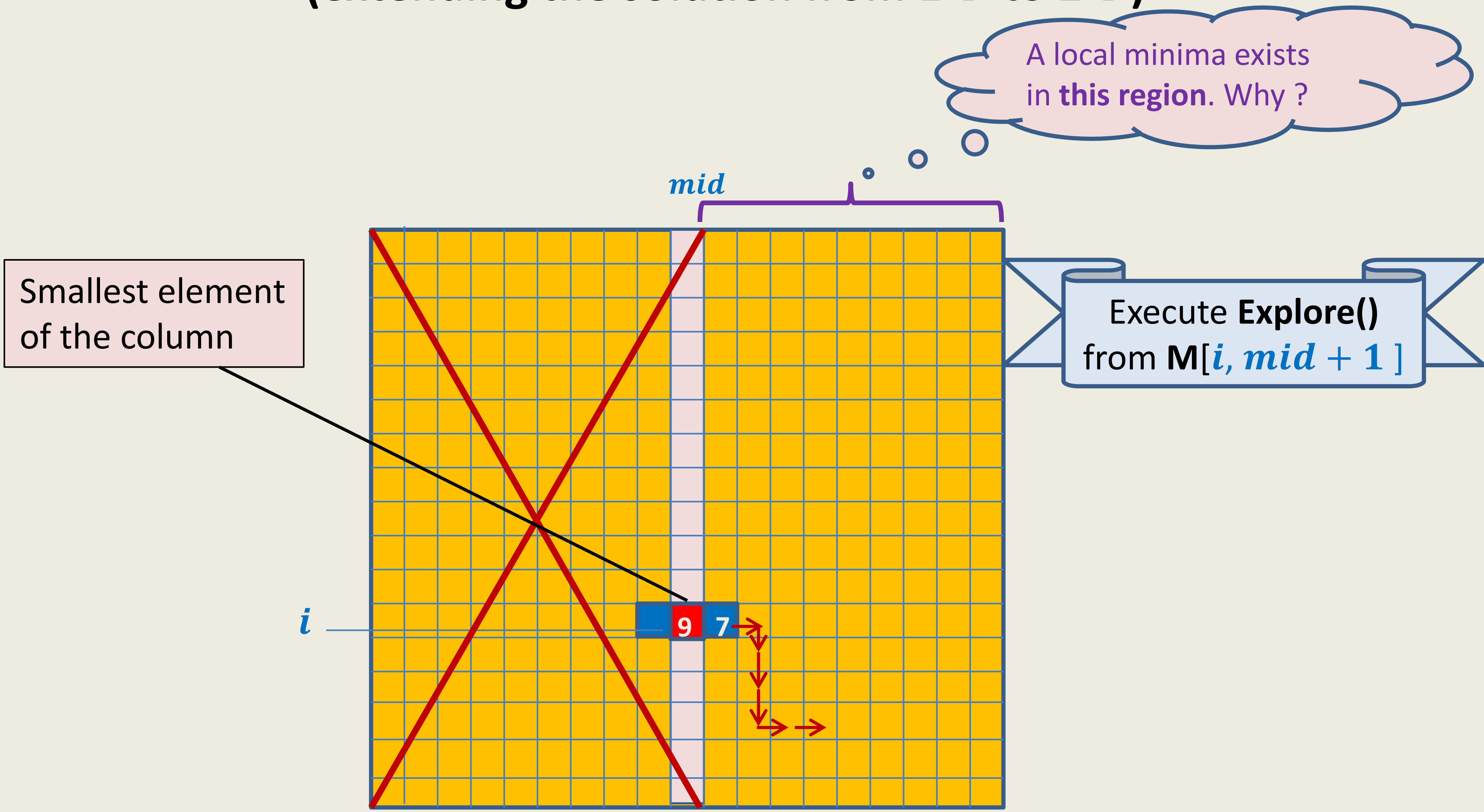
Homework: How to prove the correctness of the algorithm using the assertion ?

Local minima in an array

Theorem: A local minima in an array storing n distinct elements can be found in **O(log** n) time.

Local minima in a grid

(extending the solution from 1-D to 2-D)



Local minima in a grid

```
Int Local-minima-in-grid(M)  // returns the column containing a local minima
{       L ← 0;
        R ← n − 1;
        found ← FALSE;
        while(not found)
        {       mid ← (L + R)/2;
                If (M[*, mid] has a local minima)     found ← TRUE;
                   else {  let M[i, mid] be the smallest element in M[*, mid]
                           if(M[i, mid + 1] < M[i, mid])   L ← mid + 1 ;
                           else   R ← mid − 1
                        }
        }
        return mid;
}
```

O(n) time

O(n) time

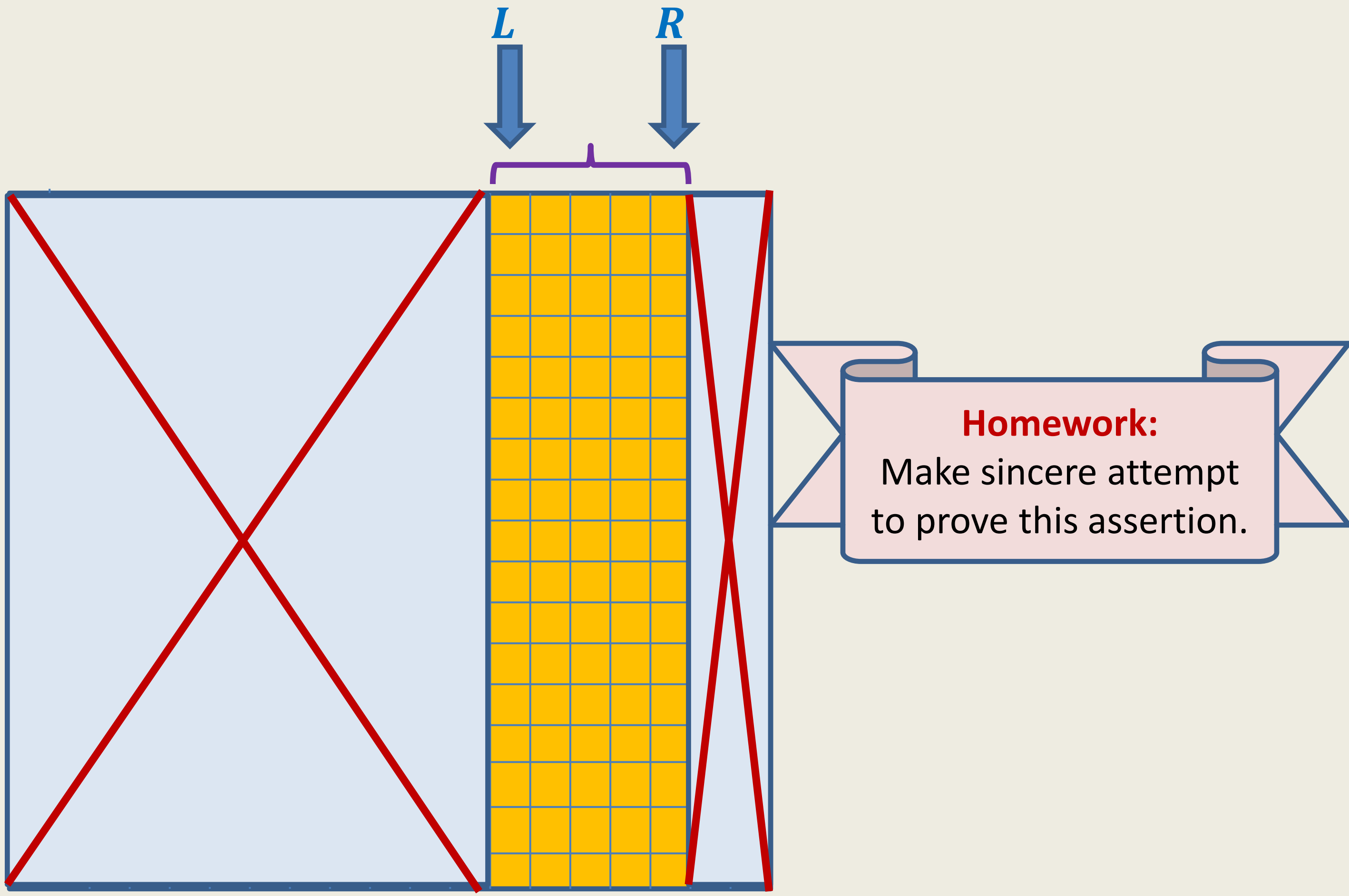
O(n) time

➔ Running time of the algorithm = O(n log n)

Proof of correctness ?

Local minima in a grid

(Proof of Correctness)



Assertion: In the beginning of each iteration, the following assertion holds

"A local minima of grid **M** exists in **M**[L,…, R]."

Local minima in a grid

Theorem: A local minima in an $n \times n$ grid storing distinct elements can be found in **O**(n **log** n) time.

How to improve it to **O**(n) ?

Local minima in a grid in O(n) time

Let us carefully look at the calculations of the running time of the current algo.

$cn + cn + cn + \ \ldots$ (log n terms) $\ldots + cn$ = O($n \log n$)

What about the following series

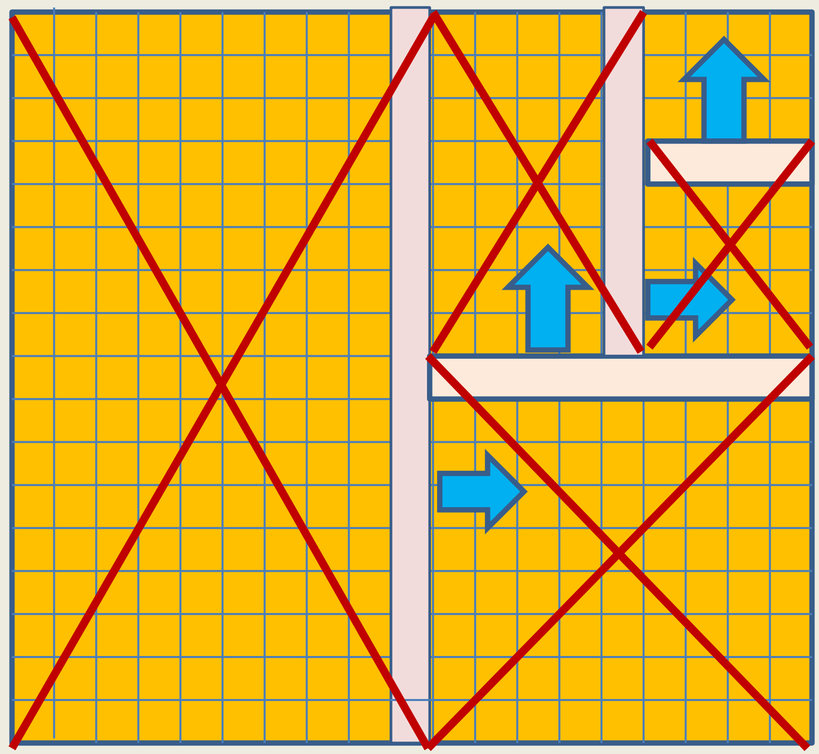
$c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} + \ \ldots$ (log n terms) $\ldots + cn$ = ?

It is $2cn$ = O(n) .

Get an !DEA from this series to modify our current algorithm

Local minima in a grid in O(n) time

Bisect alternatively along rows and column



You will have to do it **completely** as a part of the next assignment ☺

Theorem:

Given an $n \times n$ grid storing n^2 distinct elements, a local minima can be found in $\mathbf{O}(n)$ time.

Question:

On which algorithm paradigm, was this algorithm based on ?

- **Greedy**
- **Divide and Conquer**
- **Dynamic Programming**

Range-Minima Problem

A Motivating example

to realize the importance of data structures

Range-Minima Problem

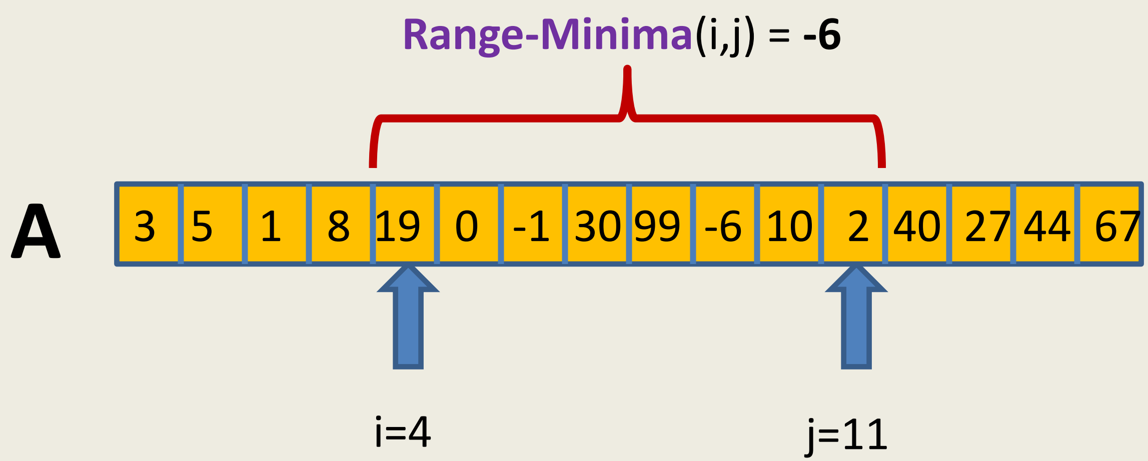
Given: an array **A** storing n numbers,

Aim: a data structure to answer a sequence of queries of the following type

Range-minima(i,j) : report the smallest element from **A**[i],…,**A**[j]

Let **A** store one **million** numbers

Let the number of queries be **10 millions**



Range-Minima Problem

Question: Does there exist a data structure which is

- **Compact**

  (O(n log n) **size**)
- **Can answer each query efficiently ?**

  (O(1) time)

Homework 4: Ponder over the above question.

(we shall solve it in the next class)