Data Structures and Algorithms

(ESO207)

Lecture 6:

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

Local minima in an array

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

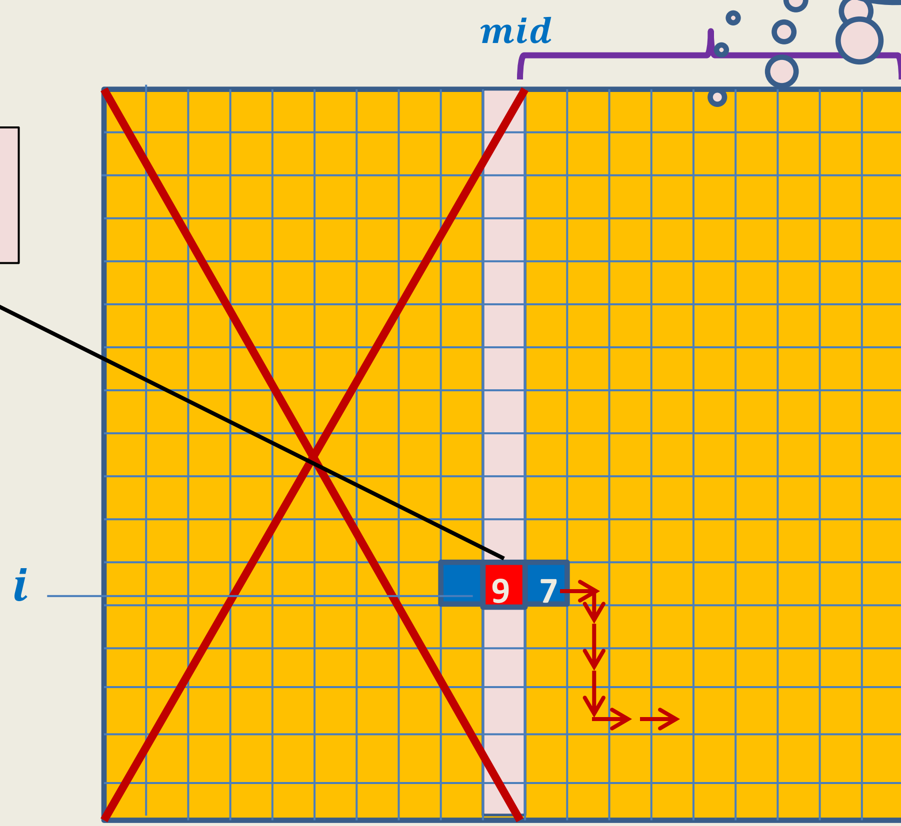
Local minima in a grid

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

Search for a local minima in the column **M**$[*, \boldsymbol{mid}\ ]$

Under what circumstances even this smallest element is not a local minima ?

Smallest element of the column



Execute **Explore()** from **M**$[\boldsymbol{i}, \boldsymbol{mid}+\mathbf{1}\ ]$

Homework:

Use this idea to design an **O**$(\boldsymbol{n}$ **log** $\boldsymbol{n})$ time algorithm for this problem.

… and do not forget to prove its correctness ☺.

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**; O(n) time

else {

let **M**[k, mid] be the smallest element in **M**[*, mid] O(n) time

if(**M**[k, $mid+1$] < **M**[k, mid]) L← $mid+1$;

else R← $mid-1$

}

}

return mid;

}

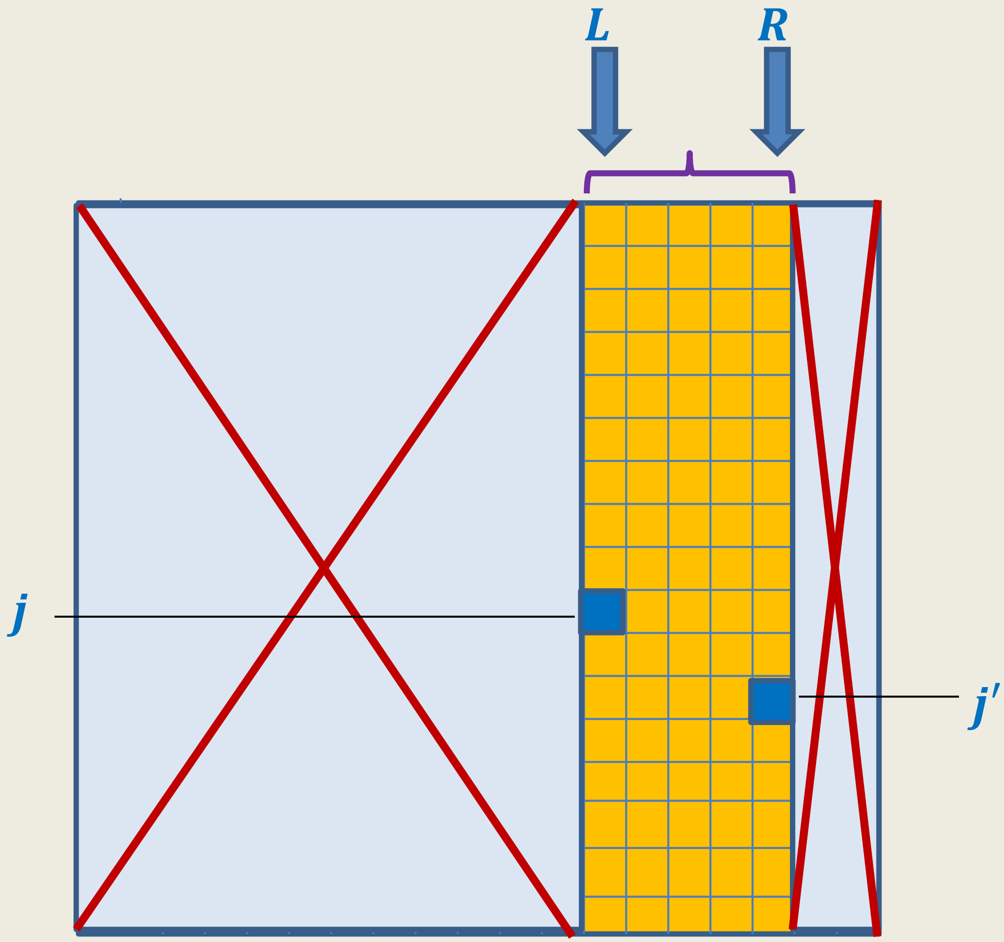
O(n) time

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

Proof of **correctness** ?

Local minima in a grid

(Proof of Correctness)



P(i) :

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

$\exists$ j such that **M**[j, L] < **M**[$*, L-1$]" **and** $\exists$ j' such that **M**[j', R] < **M**[$*, R+1$]"

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.

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

What about the following series

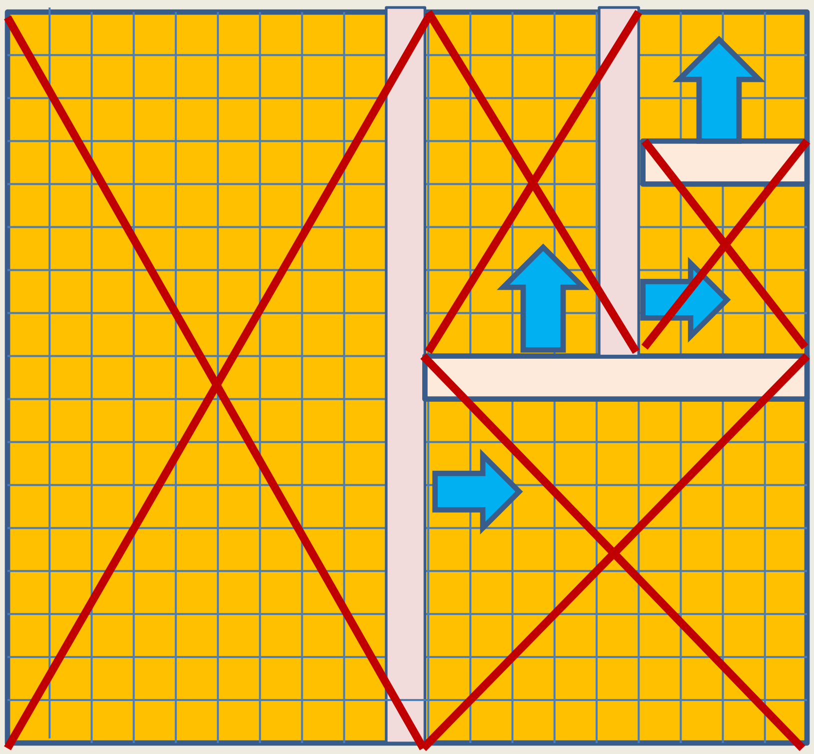
$c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} +$ … (log n terms) … $+ cn$ = ?

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



INCORRECT

Exercise: Think of a counterexample!

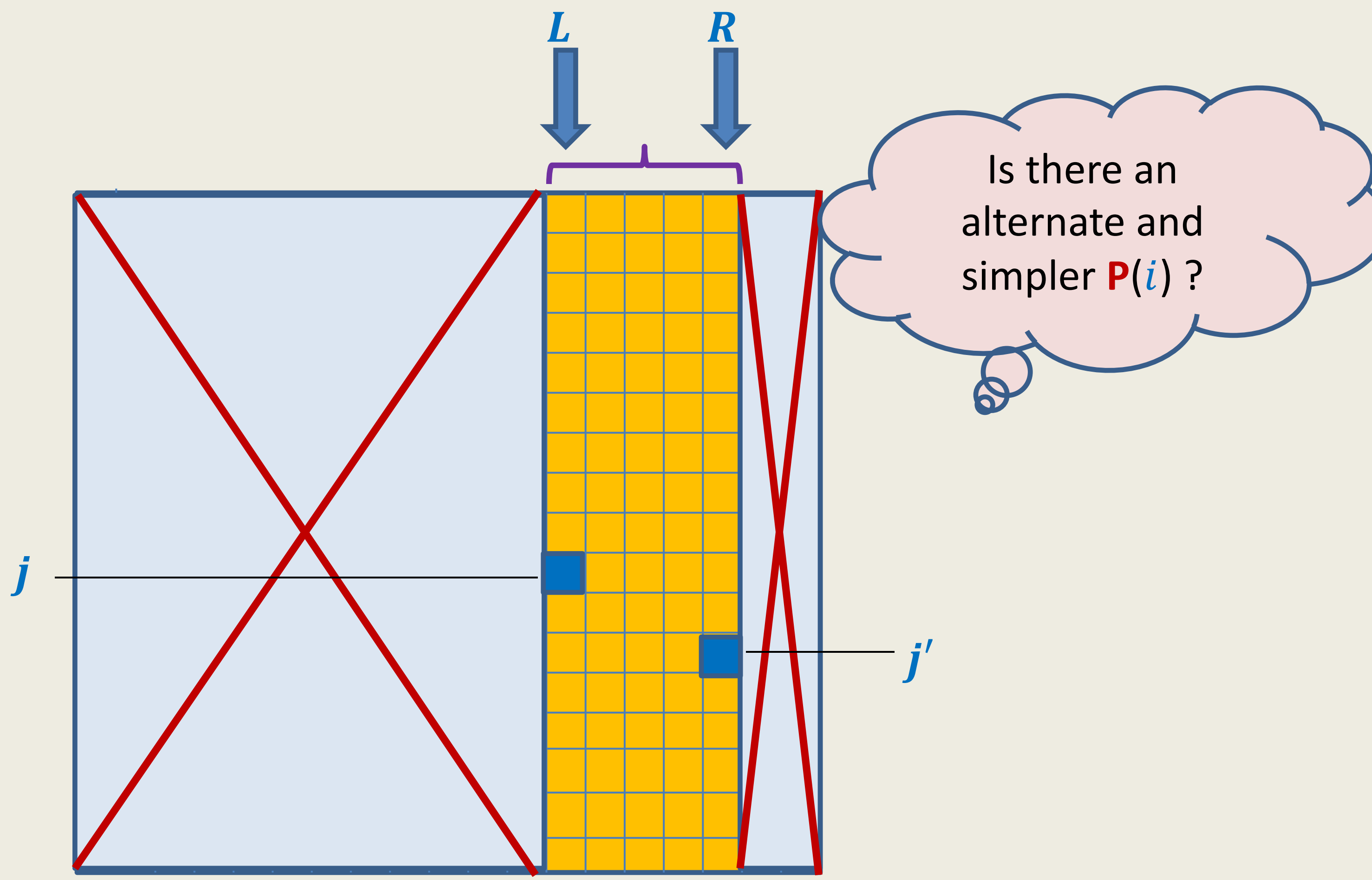
Lessons learnt

- No hand-waving works for iterative algorithms ☺
- We must be sure about
  - What is **P**(i)
  - Proof of **P**(i).

Let us revisit the ($n \log n$) algorithm

Local minima in a grid

(Proof of Correctness)

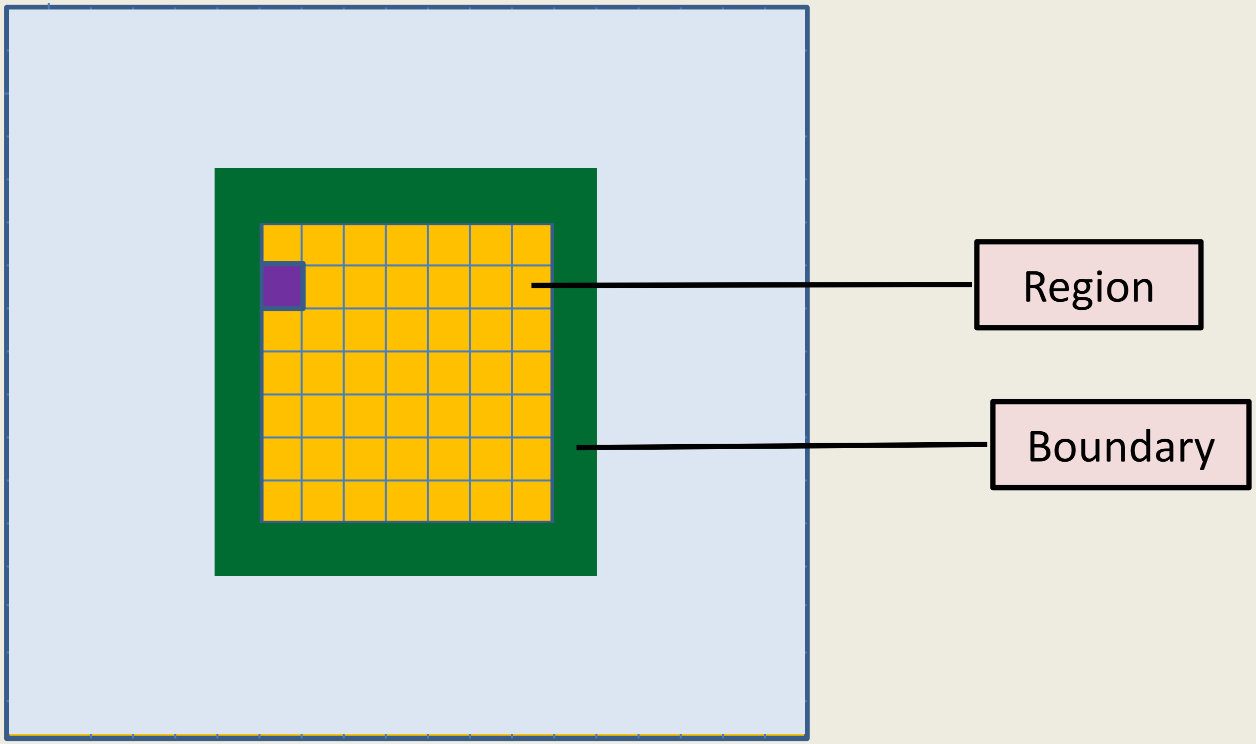


$\mathbf{P}(i)$:

"A local minima of grid **M** exists in $\mathbf{M}[L,\ldots,R]$."

$\equiv$ "$\exists\ j$ such that $\mathbf{M}[j, L] < \mathbf{M}[*, L-1]$" **and** "$\exists\ j'$ such that $\mathbf{M}[j', R] < \mathbf{M}[*, R+1]$"

At any stage, what guarantees the existence of local minima in the region ?

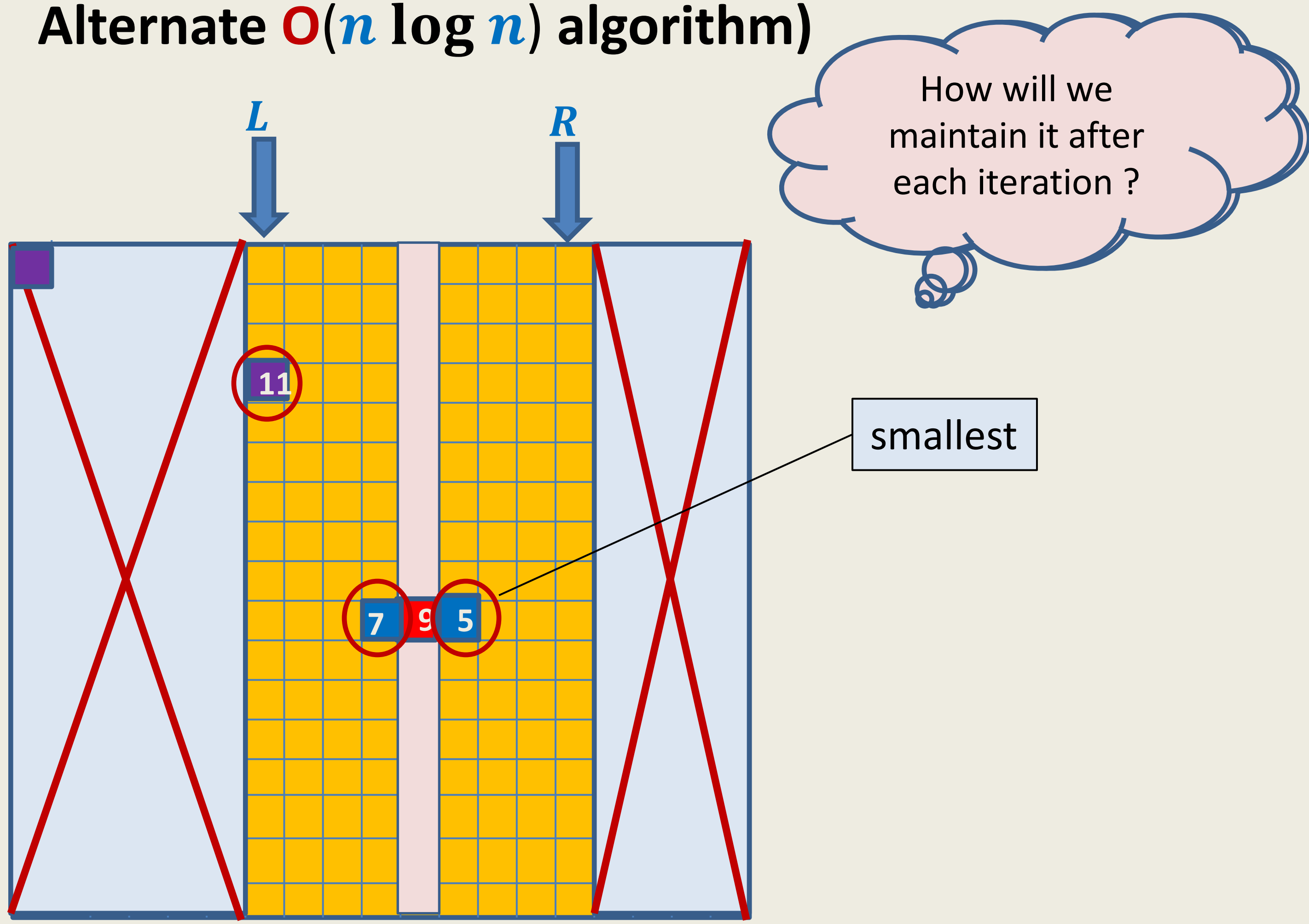


- At each stage, our algorithm may maintain a cell in the region whose value is <u>smaller</u> than all elements lying at the **boundary** of the **region** ?

(Note the boundary lies outside the region)

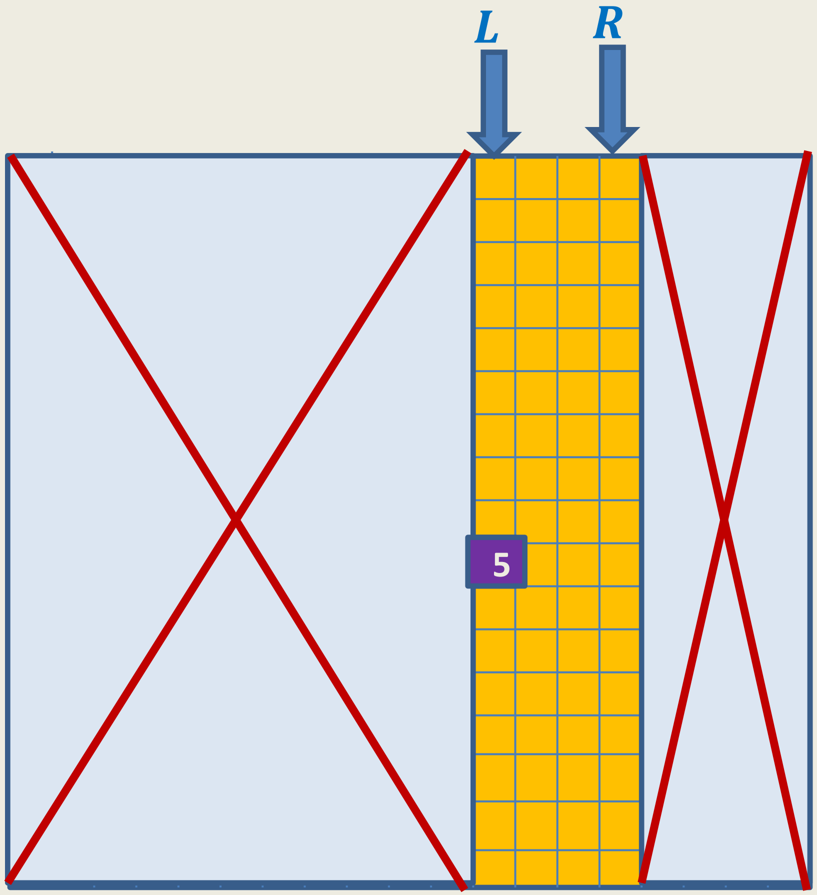
Local minima in a grid

Alternate O(n log n) algorithm)



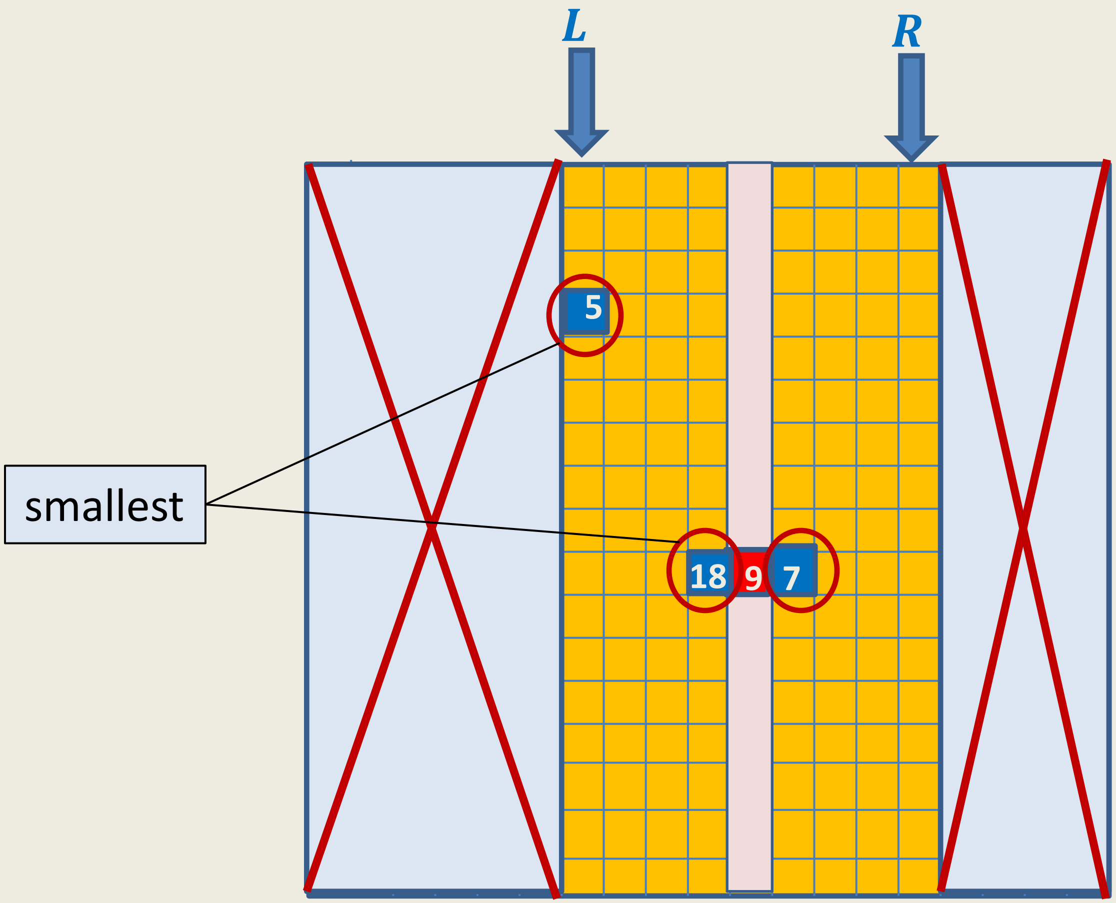
Local minima in a grid

Alternate O(n log n) algorithm



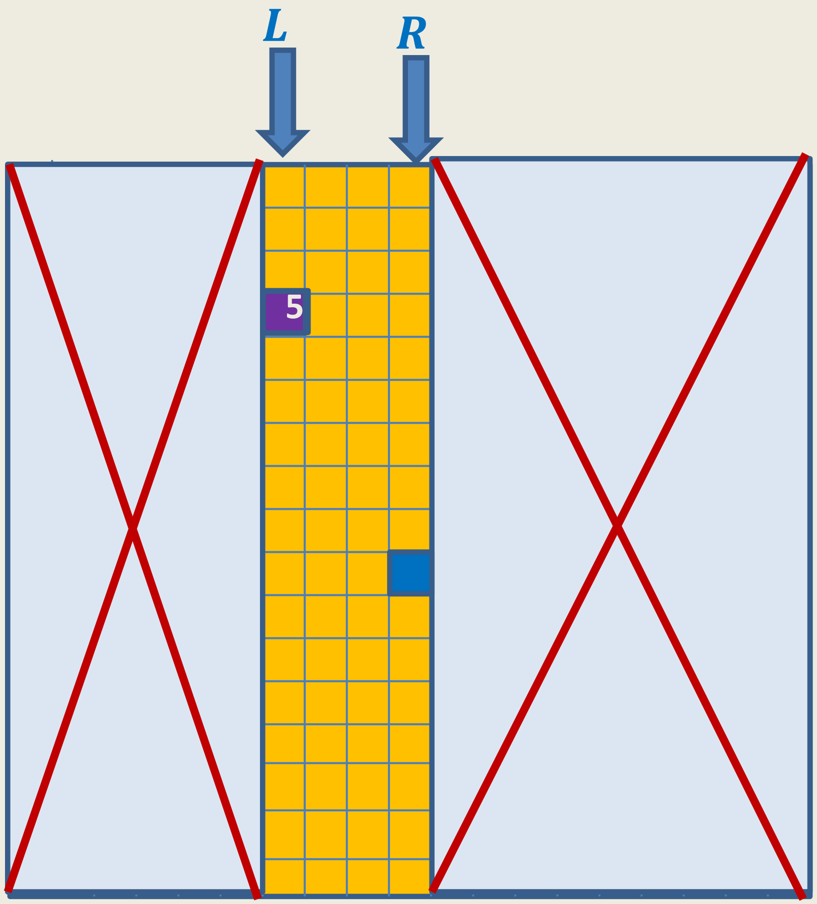
Local minima in a grid

Alternate O(n log n) algorithm



Local minima in a grid

Alternate O(n log n) algorithm



- A neat pseudocode of this algorithm is given on the following slide.

Local minima in a grid

Alternate O(n log n) algorithm

Int Local-minima-in-grid(M) // returns the column containing a local minima

```
{       L ← 0;
        R ← n − 1;
        found ← FALSE;
        C ←(0,0);
        while(not found)
        {       mid ← (L + R)/2;
                If (M[*, mid] has a local minima)    found ← TRUE;
                   else {
                              let M[k, mid] be the smallest element in M[*, mid]
                              C' ←(k,mid − 1);
                              C'' ←(k,mid + 1);
                              C_min ← the cell containing the minimum value among {C, C', C''};
                              If (column of C_min is present in (mid + 1, R))     L ← mid + 1;
                                 else R ← mid − 1;
                              C ← C_min ;
                        }
        }
        return mid;
}
```

Extending it to O(n) is easy now.
Do it as a homework.

Homework: Prove the correctness of this algorithm.

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**

Proof of correctness of algorithms

Worked out examples:

- **GCD**
- **Binary Search**

GCD

```
GCD(a,b)     // a ≥ b
{
    while (b <> 0)
    {     t ← b;
          b ← a mod b ;
          a ← t
    }
    return a;
}
```

Lemma (Euclid):

If $n \geq m > 0$, then

gcd(n,m) = **gcd**$(m, n \bmod m)$

Proof of correctness of GCD(a,b) :

Let a_i : the value of variable a after ith iteration.

b_i : the value of variable b after ith iteration.

Assertion $P(i)$: $\boxed{\mathbf{gcd}(a_i, b_i) = \mathbf{gcd}(a,b)}$

Theorem : $P(i)$ holds for each iteration $i \geq 0$.

Proof: (By induction on i).

Base case: ($i = 0$) hold trivially.

Induction step:

(Assume $P(j)$ holds, show that $P(j+1)$ holds too)

$P(j)$ ➔ $\boxed{\mathbf{gcd}(a_j, b_j) = \mathbf{gcd}(a,b). \quad \text{----(1)}}$

$(j+1)$ iteration ➔ $\boxed{a_{j+1} = b_j \textbf{ and } b_{j+1} = a_j \bmod b_j \text{ ---(2)}}$

Using **Euclid**'s Lemma and (2),

gcd(a_j, b_j) = **gcd**(a_{j+1}, b_{j+1}) ------(3).

Using (1) and (3),assertion $P(j+1)$ holds too.

Binary Search

Binary-Search(**A**[$0 \ldots n-1$], x)

L ← 0;

R ← $n-1$;

Found ← false;

While (**L** ≤ **R** and **Found** = false)

{ **mid** ← (**L+R**)/2;

If (**A**[**mid**] $= x$) **Found** ← true;

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

else **R** ← **mid - 1**

}

if Found return true;

else return false;

Observation: If the code returns true, then indeed **output** is correct.

So all we need to prove is that whenever code returns false , then indeed x is not present in **A**[].

This is because Found is set to true only when x is indeed found.

Binary Search

Binary-Search(A[$0...n-1$], x)

L ← 0;

R ← $n-1$;

Found ← false;

While (L ≤ R and Found = false)

{ mid ← (L+R)/2;

If (A[mid] $=x$) Found ← true;

else if (A[mid] $< x$) L ← mid + 1;

else R ← mid - 1

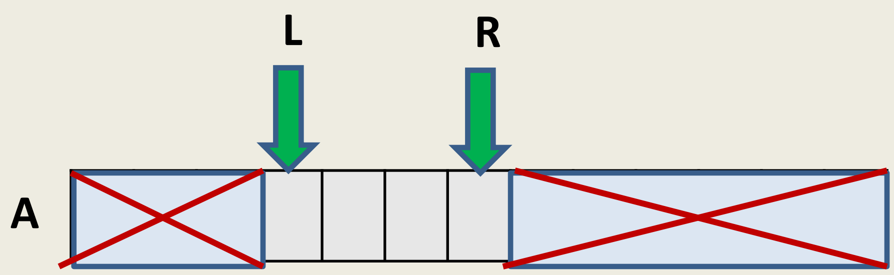
}

if Found return true;

else return false;

Assertion $P(i)$: $x \notin$ { A[0],…,A[L−1] }

and $x \notin$ { A[R+1],…,A[n −1] }



Range-Minima Problem

A Motivating example

to realize the importance of data structures

Range-Minima Problem

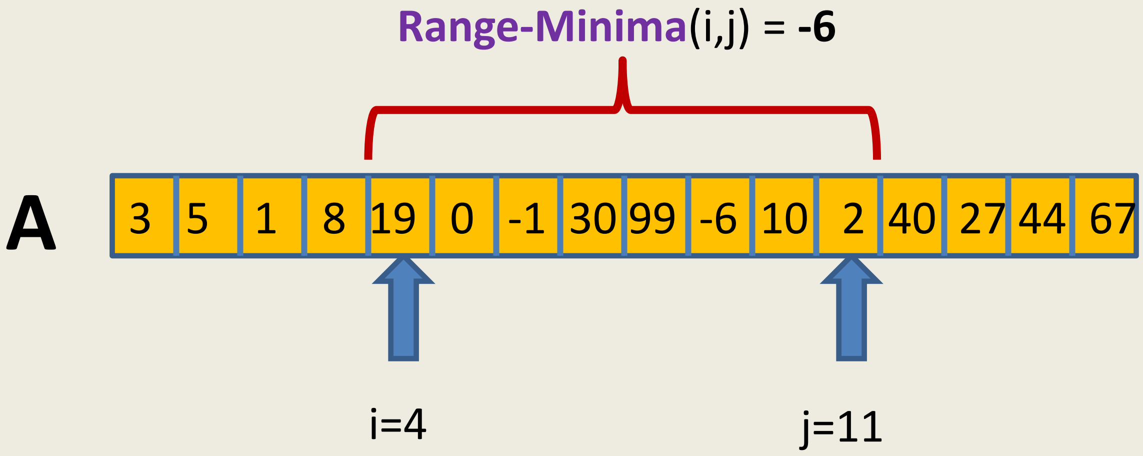
Given: an array **A** storing n numbers,

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 : Ponder over the above question.

(we shall solve it in the next class)