# Data Structures and Algorithms

(**ESO207**)

**Lecture 5:**

- More on **Proof of correctness** of an algorithm
- Design of **O**($n$) time algorithm for **Local Minima in a grid**

# PROOF OF CORRECTNESS

# What does **correctness of an algorithm** mean ?

For every possible **valid input**, the algorithm must output **correct** answer.

Let us take a toy algorithm ☺

# Algorithm for computing sum of numbers from $\mathbf{0}$ to $\boldsymbol{n}$

```
Sum_of_Numbers(n)
{   Sum ← 0;
    for i = 1 to n
    {
          Sum ← Sum + i;
    }
    return Sum;
}
```

How will you convince any person
that **Sum_of_Numbers**($n$)
indeed is correct ?

**Natural responses**:

- It is obvious !
- Compile it and run it for some random values of $n$.
- Go over first few iterations explaining what happens to **Sum**.

# How will you respond

**if you have to do it for the following code ?**

```
void dij(int n,int v,int cost[10][10],int dist[])
{
 int i,u,count,w,flag[10],min;
 for(i=1;i<=n;i++)
  flag[i]=0,dist[i]=cost[v][i];
 count=2;
 while(count<=n)
 {
  min=99;
  for(w=1;w<=n;w++)
   if(dist[w]<min && !flag[w])
    min=dist[w],u=w;
  flag[u]=1;
  count++;
  for(w=1;w<=n;w++)
   if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
    dist[w]=dist[u]+cost[u][w];
 }
}
```

Think for some time to realize

- the non-triviality
- the Importance

of proof of correctness of an iterative algorithm.

In the following slide, we present an overview of the proof of correctness.

Interestingly, such a proof will be just

Expressing our intuition/insight of the algorithm in a **formal** way ☺.

# Proof of correctness

**For an iterative algorithm**

Insight of the algorithm → **Theorem**

Assertion **P**($i-1$) | Assertion **P**($i$)

Start of **Loop** | 1 | 2 | 3 | 4 | ○ ○ ○ | $i-1$ | $i$ | ○ ○ ○

TARGET

Prove **P**($i$) by

1. Assuming **P**($i-1$)
2. **Theorem**
3. Body of the **Loop**

$\}$ Proof by induction

What would you expect at the end of $i$th iteration ?

The most difficult/creative part of proof : To come up with the right assertion **P**($i$)

# Algorithm for computing sum of numbers from $\mathbf{0}$ to $\boldsymbol{n}$

```
{   Sum←0;
    for i = 1 to n
    {
          Sum← Sum  + i;
    }
    return Sum;
}
```

**Assertion P**($i$) : At the end of $\boldsymbol{i}$th iteration **Sum** stores the sum of numbers from $\mathbf{0}$ **to** $\boldsymbol{i}$.

**Base case:** **P**($0$) holds.

Assuming **P**($i-1$), assertion **P**($i$) also holds.

**P**($n$) holds.

# An O($n$) time Algorithm for Max-sum subarray

Let S($i$): the sum of the **maximum-sum subarray ending at index** $i$.

**Theorem 1 : If** S($i-1$) > $0$ **then** S($i$) = S($i-1$) + **A**[$i$]
**else** S($i$) = **A**[$i$]

**Max-sum-subarray-algo**(**A**[$0$ … $\boldsymbol{n}-\mathbf{1}$])
{ S[$0$] ← **A**[$0$]
**for** $i$ = **1** to $\boldsymbol{n}-\mathbf{1}$
{ **If** S[$i-1$] > $0$ **then** S[$i$] ← S[$i-1$] + **A**[$i$]
**else** S[$i$] ← **A**[$i$]
}
"Scan **S** to return the maximum entry"
}

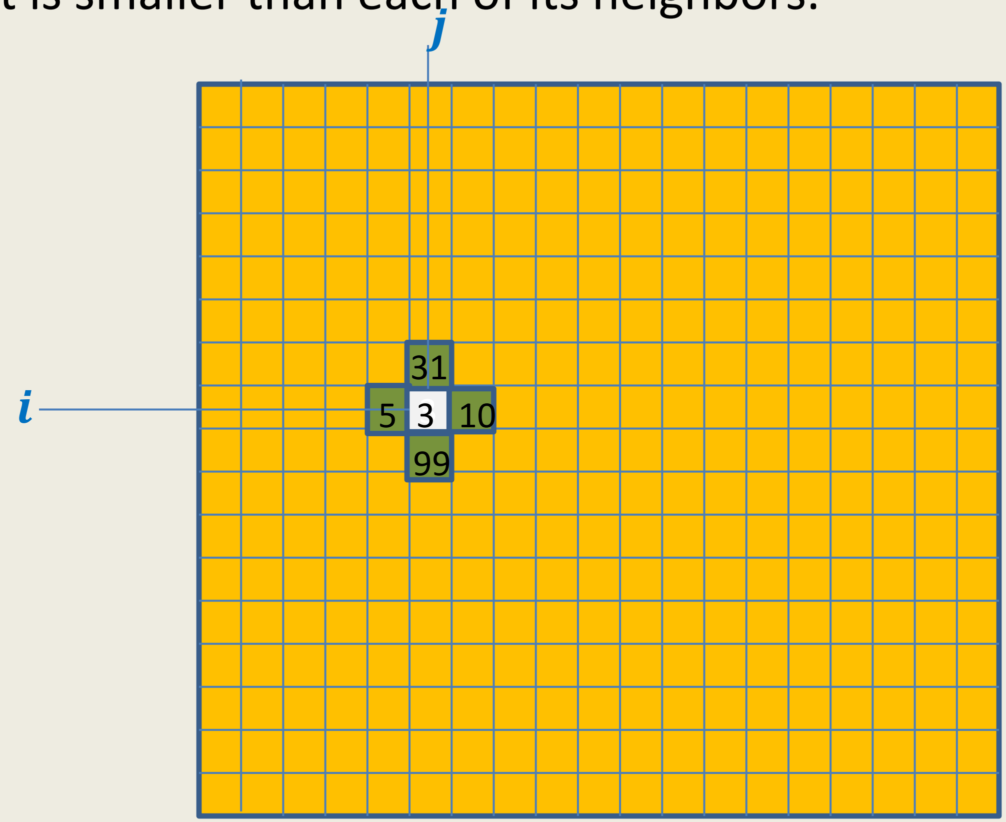
**Assertion P**($i$) : S[$i$] stores the sum of maximum sum subarray ending at **A**[$i$].

**Homework:** Prove that **P**($i$) holds for all $i \leq n-1$

# LOCAL MINIMA IN A GRID

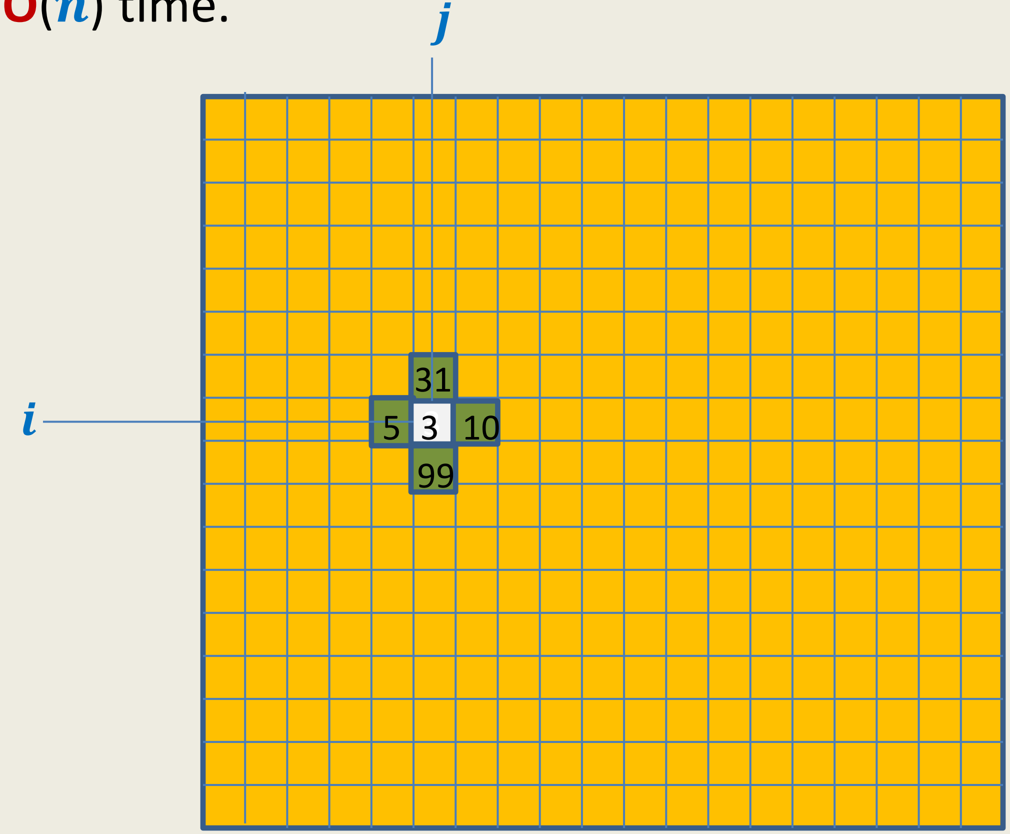
# Local minima in a grid

**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:

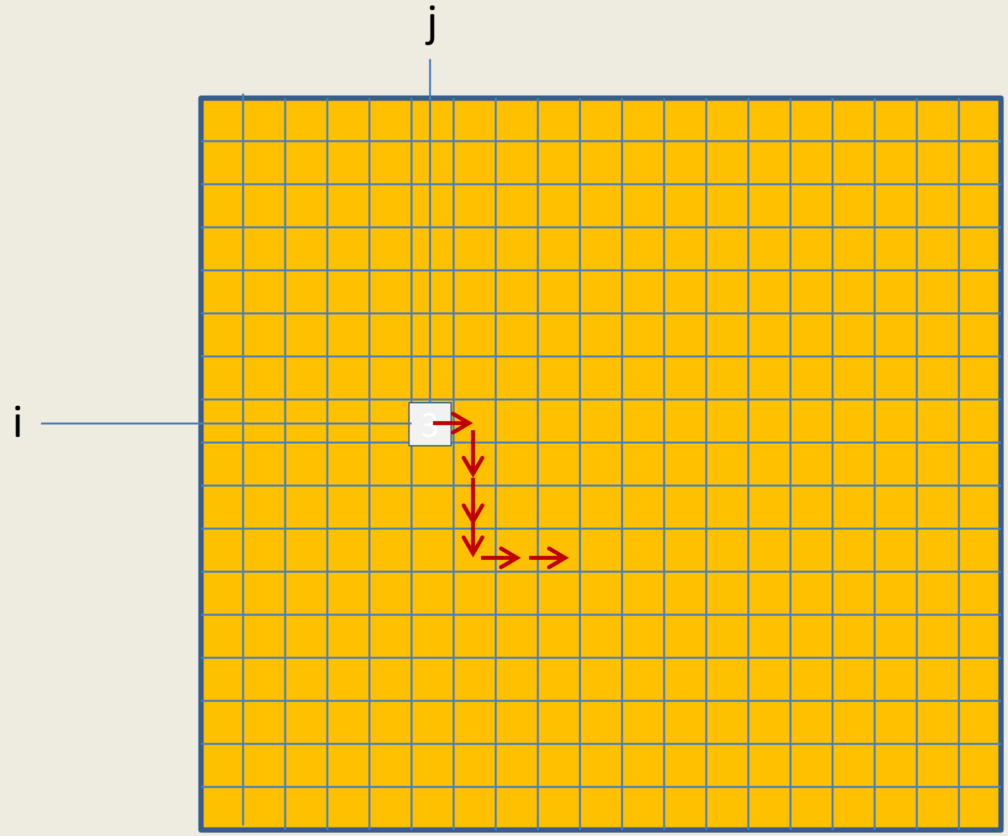
If you find a problem difficult,

→ 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 ← a neighbor of c storing smaller value
    }
    return c;
}
```

# 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;
}
```

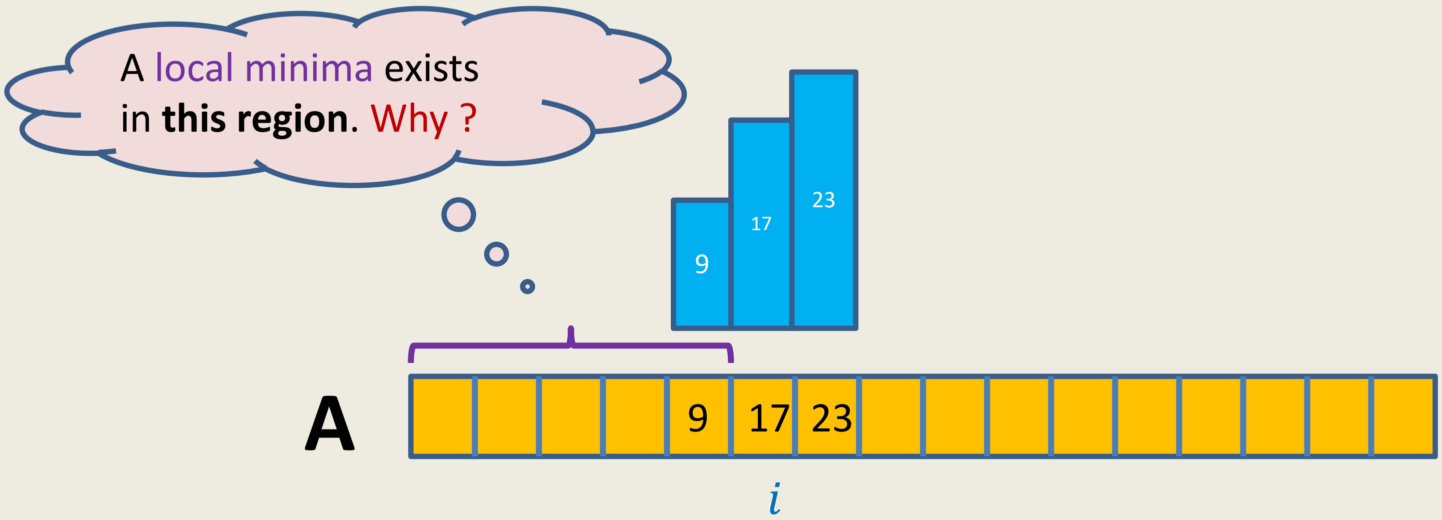
Worst case time complexity : **O**($n^2$)

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**[$i-1$].

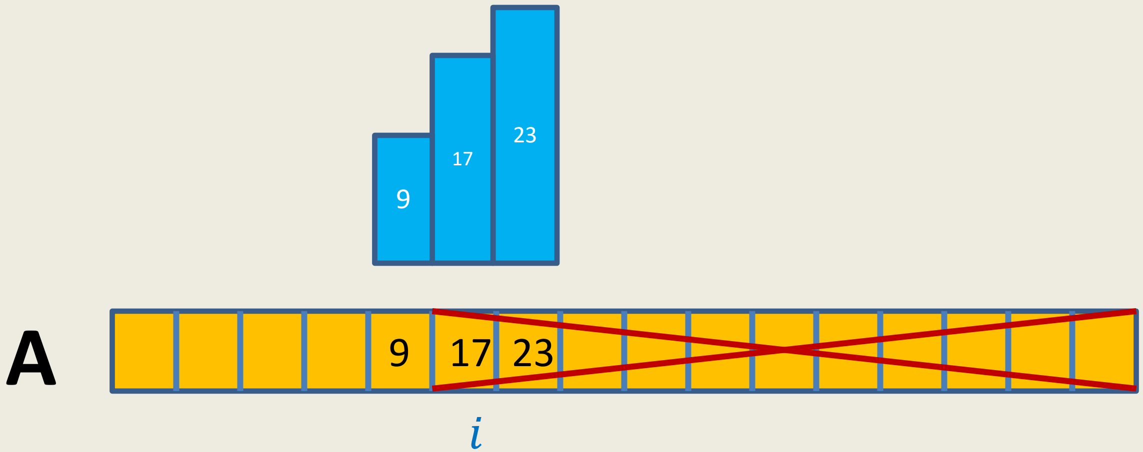
**Explore()**, if terminates, will return local minima.

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; }
```

How many iterations ?

O(log $n$)

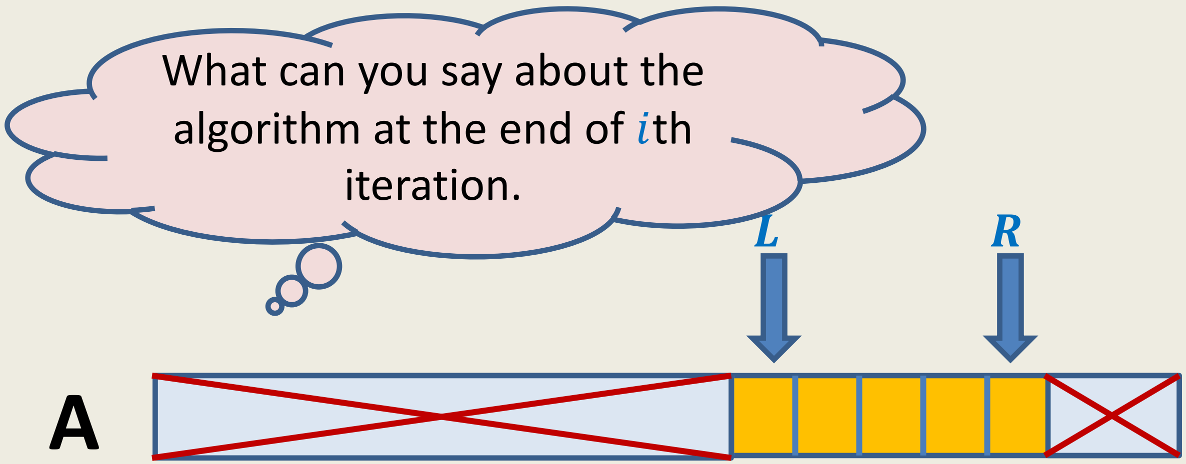
O(1) time in one iteration

→ Running time of the algorithm = O(log $n$)

Proof of correctness ?

# Local minima in an array

## (Proof of correctness)

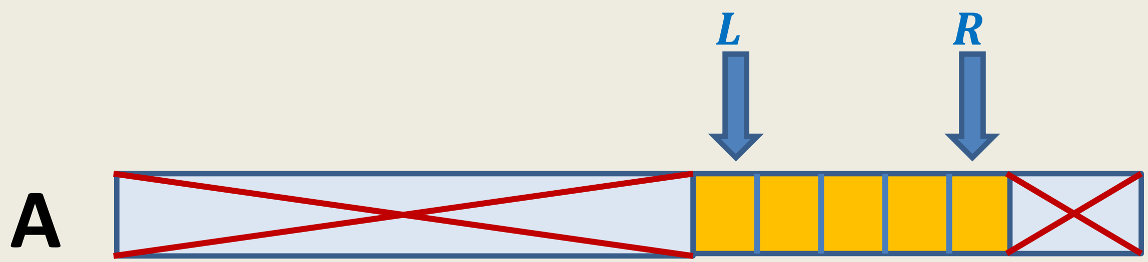



**P**($i$) : At the end of $i$th iteration,

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

# Local minima in an array

## (Proof of correctness)




**P**($i$) : At the end of $i$th iteration,

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

=

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

**Homework:**

- Make sincere attempts to prove the assertion **P**($i$).
- How will you use it to prove that **Local-minima-in-array(A)** outputs a local minima ?

# Local minima in an array

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

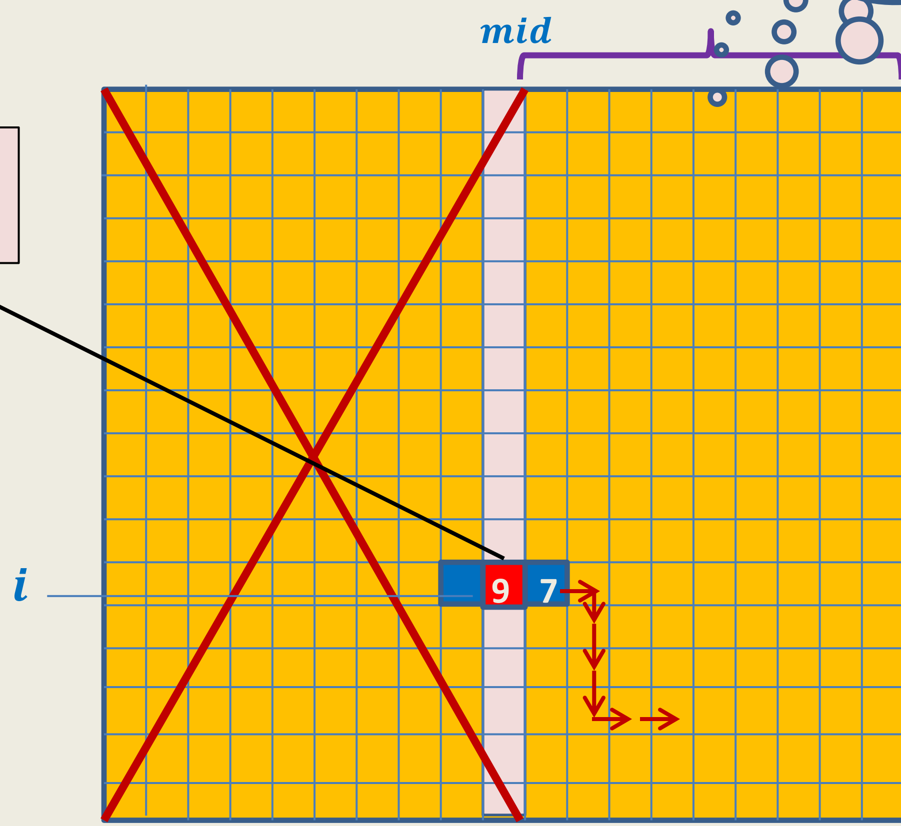
# Local minima in a grid

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

Search for a local minima in the column **M**[$*$, $mid$ ]

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

Smallest element of the column




Execute **Explore()** from **M**[$i$, $mid+1$ ]

**Homework**:

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

… and do not forget to prove its correctness ☺.

Make sincere attempts to

answer all questions raised in this lecture.