


Eötvös Loránd University
Faculty of Informatics

PROGRAMMING

2019/20 1st semester

Lecture 10

ZS. PLUHÁR, H. TORMA

Content

- Program transformations
- Sum in a submatrix
- Sorting


Program transformations

Program transformation

Program transformation: all operation, that takes an algorithm (or computer program) and generates an other algorithm (program) semantically equivalent to the original.

Main aims:

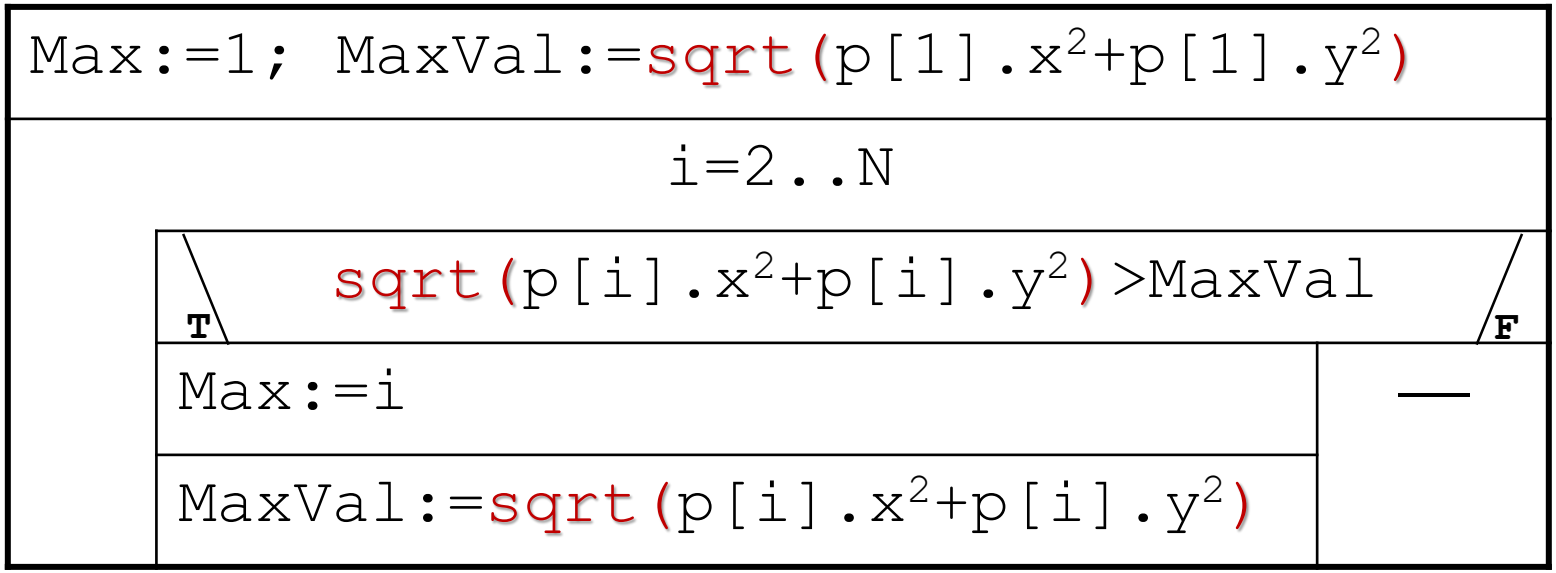
› Increase effectiveness;

› Simplify;

› Make it more feasible.

Program transformation – simplying, increase effectiveness

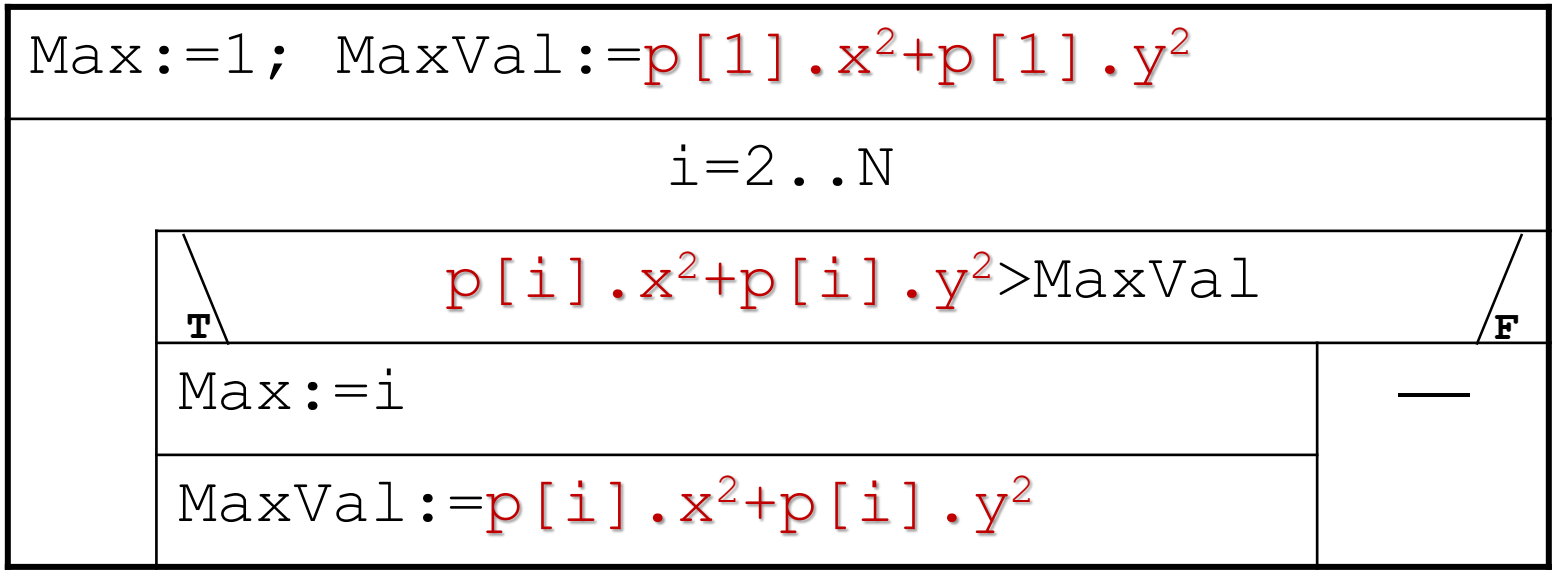
Task: the fartherst point from the origin...



The square root is a monotonuos function – it's not needed to define the maximum.

Program transformation – simplying, increase effectiveness

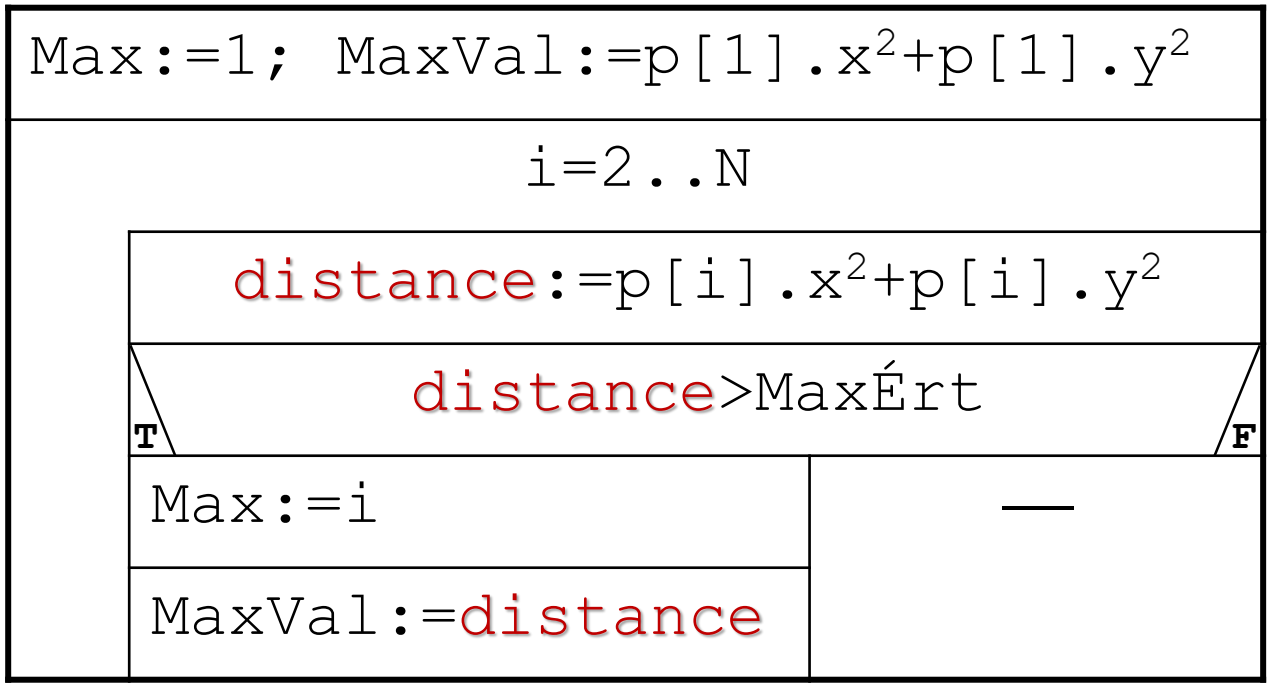
Task: the farthest point from the origin...



Here, we calculate the same expression several times.

Program transformation – to omit recalculation

Task: the farthest point from the origin...



Program transformation – expanding paralell value assignment

```
a,b,c:=f(x),g(x),h(x);
```

› Can be divided into consecutive calculation, if the relation is cycle free

```
a:=f(x); b:=g(x); c:=h(x);
```

Program transformation – expanding paralell value assignment

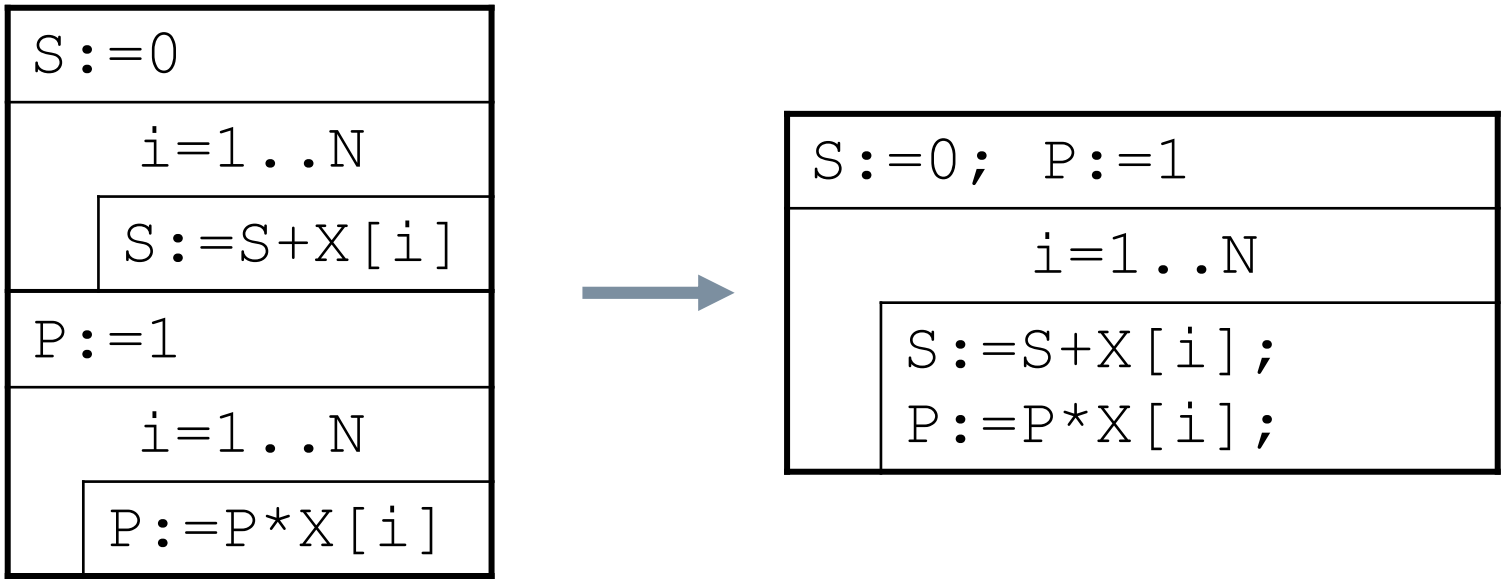
```
a,b,c:=b,c,a;
```

› Can be divided into consecutive calculations with the help of an *auxiliary variable*, if the relation is not cycle free – contains a cycle

```
av:=a; a:=b; b:=c; c:=av;
```

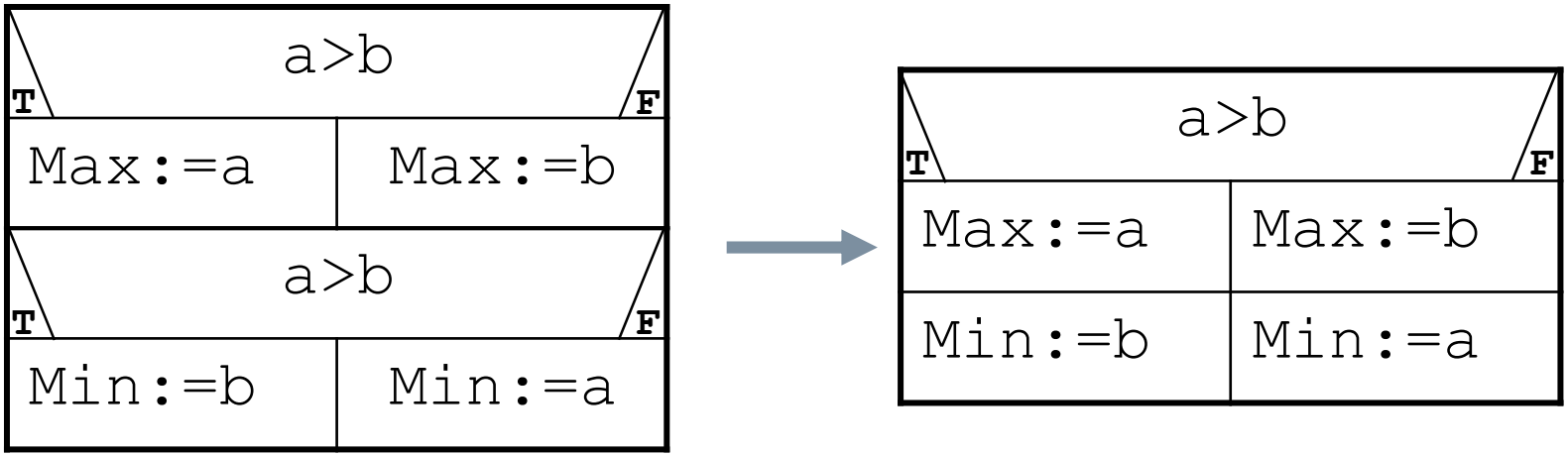
Program transformation – combining loops, loop fusion

› Loops having the same number of loop cycles, can be combined, if they are independent of each other



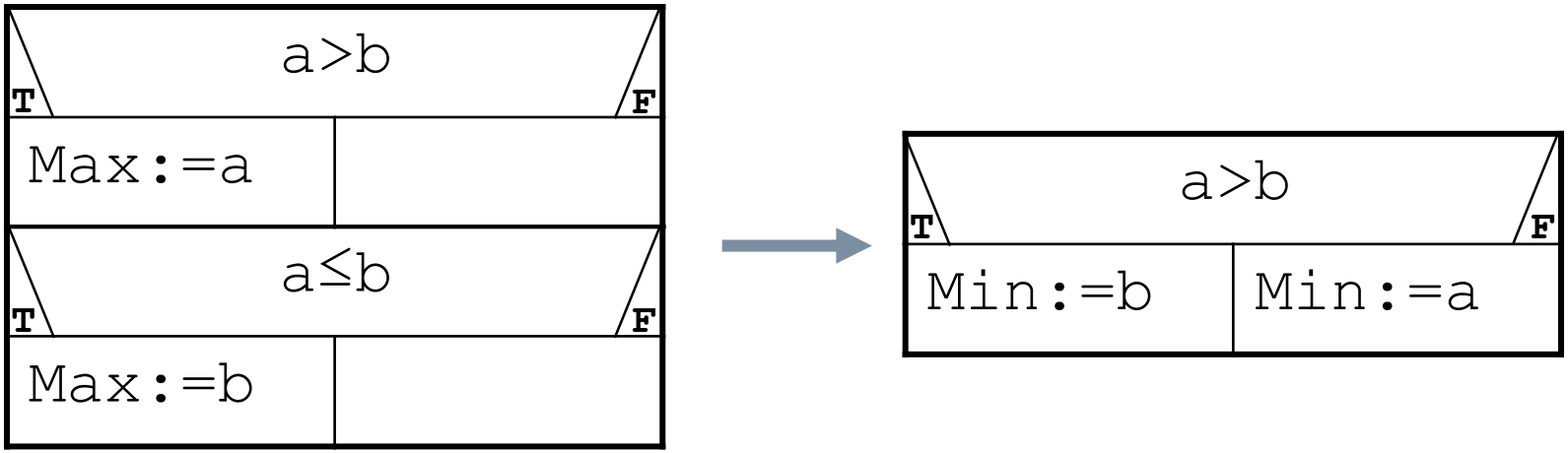
Program transformation – combining conditional statements

› Conditional statements having the same conditions can be combined, if they are independent of each other

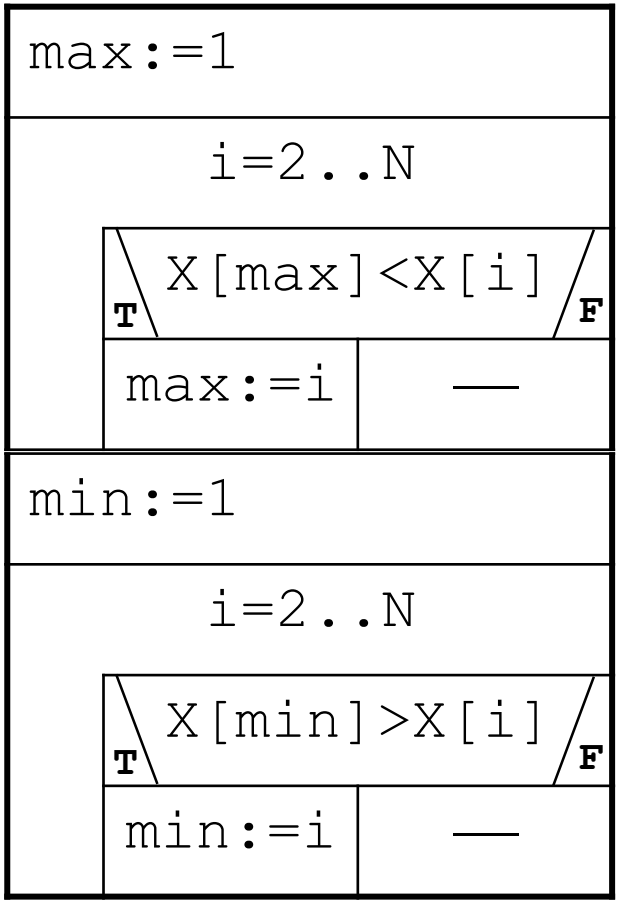


Program transformation – combining conditional statements

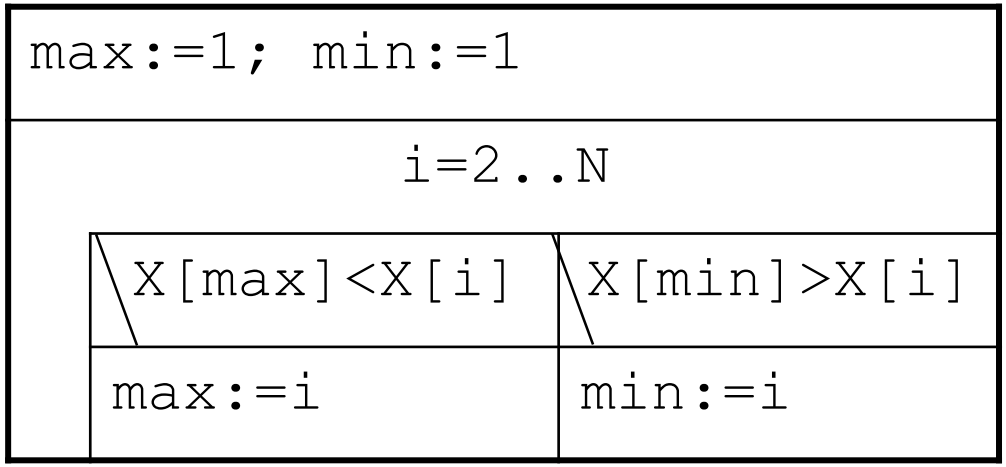
› Combination of conditionals having exclusive, full conditionals, if they are independent of each other



Program transformation – fusion of loops and conditionals

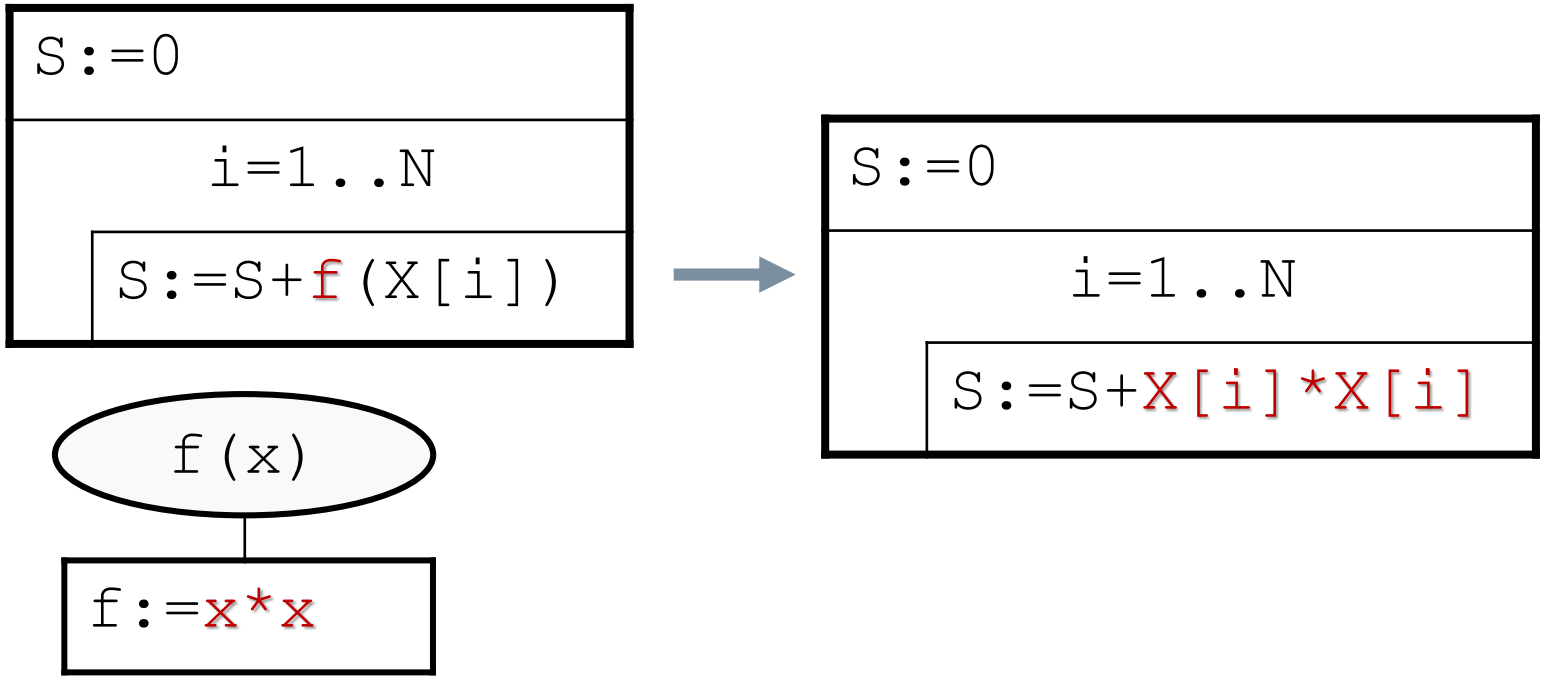


› Loops having the same number of steps and conditional statements having exclusive conditions could also be combined, if they are independent of each other



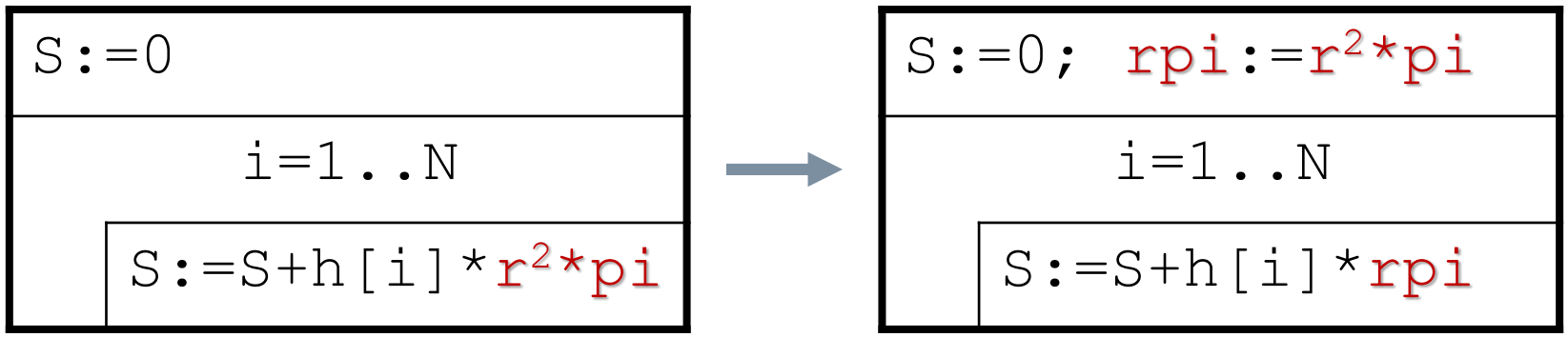
Program transformation – function inlining

› Instead of a function call, the formula of a simple function (the body of the function) can be written.
(C++ compilers can do such optimalization.)



Program transformation – hoisting: moving expression outside loop

› Loop-invariant expressions can be hoisted out of loops, thus improving run-time performance by executing the expression only once rather than at each iteration.



Program transformation

- searching, deciding → selecting
- We put a special element with A attribute at the end of the sequence – certainly will be find one

```
i:=1
  i≤N and A(X[i])
    i:=i+1
Exists:=i≤N
```

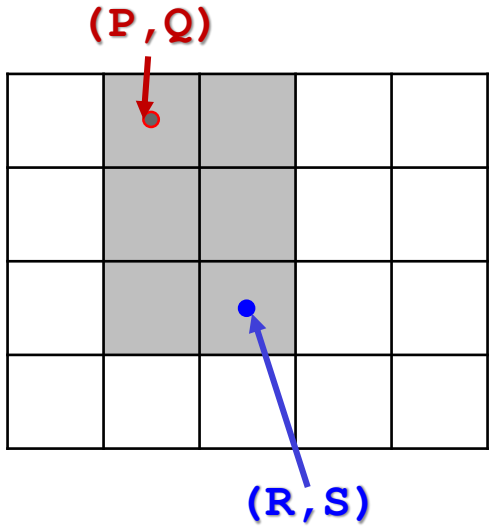
```
i:=1;
X[N+1]:=Titem
        A(X[i])
    i:=i+1
Exists:=i≤N
```

Sum of submatrices

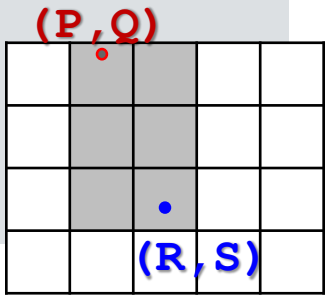
Sum of submatrices

Task: A farmer want to buy a rectangle shaped area in a NxM dimension, rectangle shaped land. He knows about each part of the land how high would be the profit or the deficit to farming it.

Give the rectange shaped area where the farmer can have the highest profit.



Sum of submatrices



Specification:

Input: $N,M \in \mathbb{N}$, $T_{1..N,1..M} \in \mathbb{Z}^{N \times M}$

Output: $P,Q,R,S \in \mathbb{Z}$

Precondition: –

Postcondition: $1 \le P \le R \le N$ and $1 \le Q \le S \le M$ and

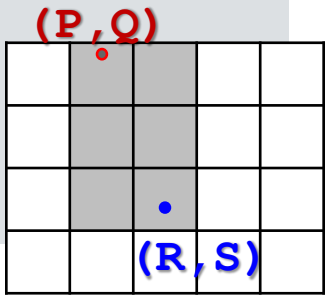
$\forall i,j,k,l\,(1 \le i \le k \le N, 1 \le j \le l \le M)$:
$Profit(P,Q,R,S) \ge Profit(i,j,k,l)$

Definition: $Profit: \mathbb{N}^4 \to \mathbb{Z}$

$$Profit(a,b,c,d) = \sum_{x=a}^{b} \sum_{y=c}^{d} T_{x,y}$$

We need loops for i,j,k,l and x, y → 6 nested loops – **too much!**

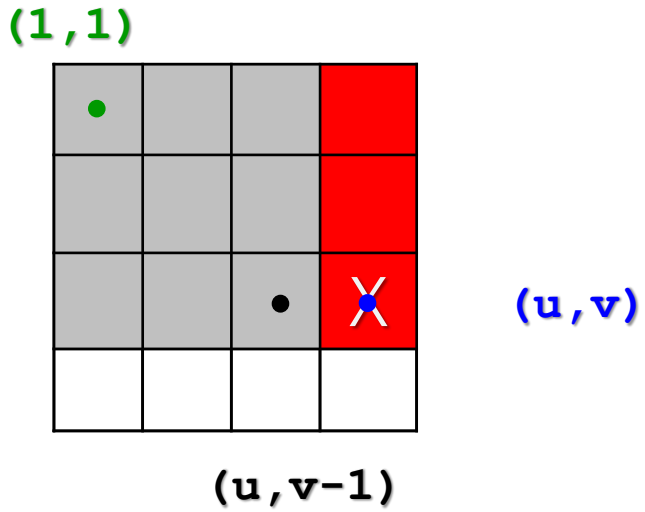
Sum of submatrices



Defining `Profit` function:

› Try to propose a subgoal:

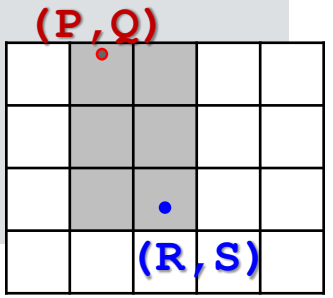
Let's calculate the sum of the rectangle with coordinates **(1,1)** and **(u,v)**



X = Profit value of the gray rectangle + sum of the red rectangle

$X \rightarrow E[u,v]$

Sum of submatrices

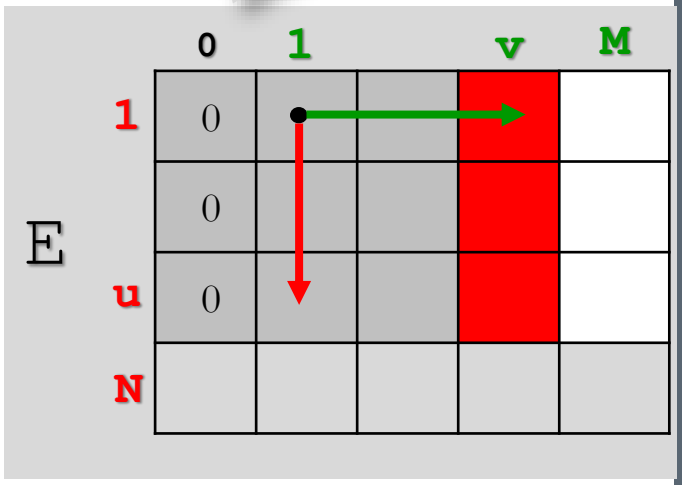


Defining `Profit` function (cont.):

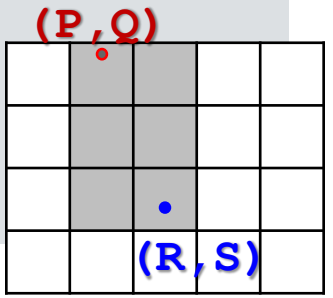
› Calculating the matrix E: (E[1..N,0]:=0):

```
v=1..M
    x:=0
        u=1..N
            x:=x+T[u,v]
            E[u,v]:=E[u,v-1]+x
...
```

Because of under-indexing problem

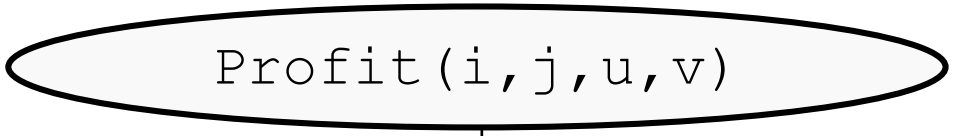


Sum of submatrices



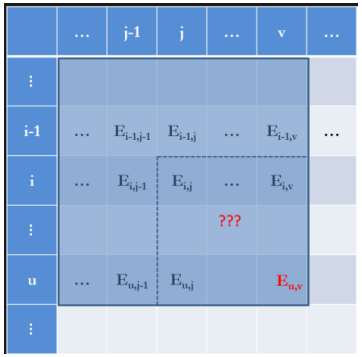
Defining `Profit` function (cont.):

› Defining the `Profit(i,j,u,v)` function with `matrix E[u,v]`:

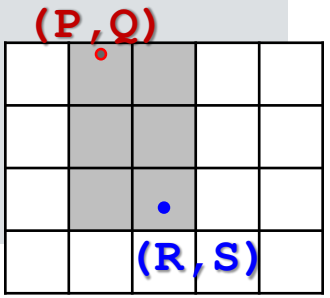



```
Profit:=E[u,v]-E[u,j-1]-E[i-1,v]+E[i-1,j-1]
```

The method: cummulative sum



Sum of submatrices



› select the rectangle with the maximum sum

```
P,Q,R,S:=1,1,1,1
MaxVal:=T[1,1]
i=1..N
  j=1..M
    k=i..N
      l=j..M
        Profit(i,j,k,l)>MaxVal
        T: P:=i; Q:=j; R:=k; S:=l
           MaxVal:=Profit(i,j,k,l)
        F: —
```

Sorting

Sorting - specification

Input: $N \in \mathbb{N}$, $X_{1..N} \in \mathbb{S}^N$

$\leq$: $\mathbb{S}^2 \to \mathbb{L}$ *(operation for comparing)*

Output: $X_{1..N}' \in \mathbb{S}^N$

Precondition: $Sorting(\leq)$ and $Sorted?_{\leq}(\mathbb{S})$

Postcondition: $Sorted?_{\leq}(X')$ and $X' \in Perm(X)$

Notations:

- X': value of X on output (at the end)
- $Sorted?(\mathbb{S})$: is the $\mathbb{S}$ sorted for the $\leq$ operation?
- $X' \in \mathbf{Perm}(S)$: X' is a permutation of the items of X

Permutation: arranging all the members of a set into some sequence or order

Sorting – terms and notations

› **Apostrophe** (’) in the specification:

If an input data appear on the output, post-condition must refer to both value. To distinguish the two values we mark the output data with an apostrophe.

e.g.: Z’:=value of Z on **output** (at the end).

› **Relation** ≤ means **sort**, if

1. *reflexive:* $\forall a \in \mathbb{S}: a \leq a$ (every element of $\mathbb{S}$ is related to itself)
2. *antisymmetric:* $\forall a, b \in \mathbb{S}: a \leq b \text{ and } b \leq a \rightarrow a = b$
3. *transitive:* $\forall a,b,c \in \mathbb{S}: a \leq b \text{ and } b \leq c \rightarrow a \leq c$

Sorting – terms and notations

- $\mathbb{S}$ (completely) **sorted set**:
  $\texttt{Sorted?}(\mathbb{S}) := \forall a,b \in \mathbb{S}: \; a \leq b \text{ or } b \leq a$

- **Sorted sequence**:

  $\texttt{Sorted?}(Z) := \forall i (1 \leq i \leq N-1): \; Z[i] \leq Z[i+1]$

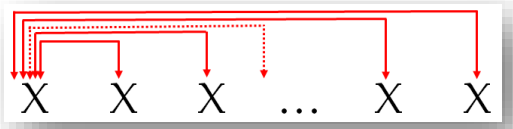
- **Permutation set**:

  $\texttt{Perm}(Z)$:=a *set* which contains *all permutation* of the elements from the sequence $Z \in \mathbb{S}^N$; so an element from this is the required sorted sequence

Simple exchange sorting

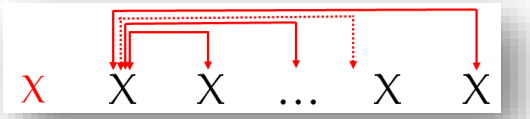
Idea:

› Compare the first element with all elements behind it and swap them if needed!



The minimum will go to the first position.

› Do the same for the second element!

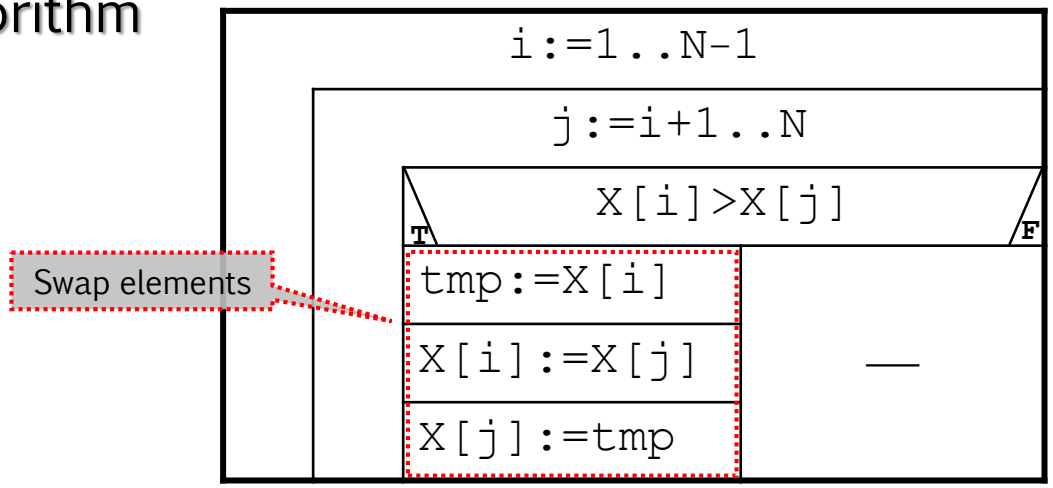


› …

› Finally do it for the last 2 elements!

Simple exchange sorting

Algorithm



Number of comparisons: $1 + 2 + .. + N - 1 = N \cdot \frac{N - 1}{2}$

Number of moving: $0 \ldots 3 \cdot N \cdot \frac{N - 1}{2}$

Selection sort

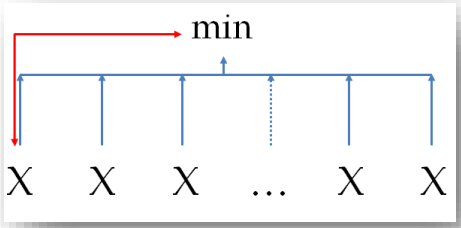
Idea:

› Determine the minimum of elements (**1..N),** swap it with the **1. one**!

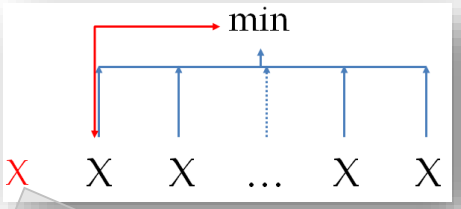
› Next do the same with elements (**2..N)**!

…

› Finally the last two elements (**N–1..N**)!



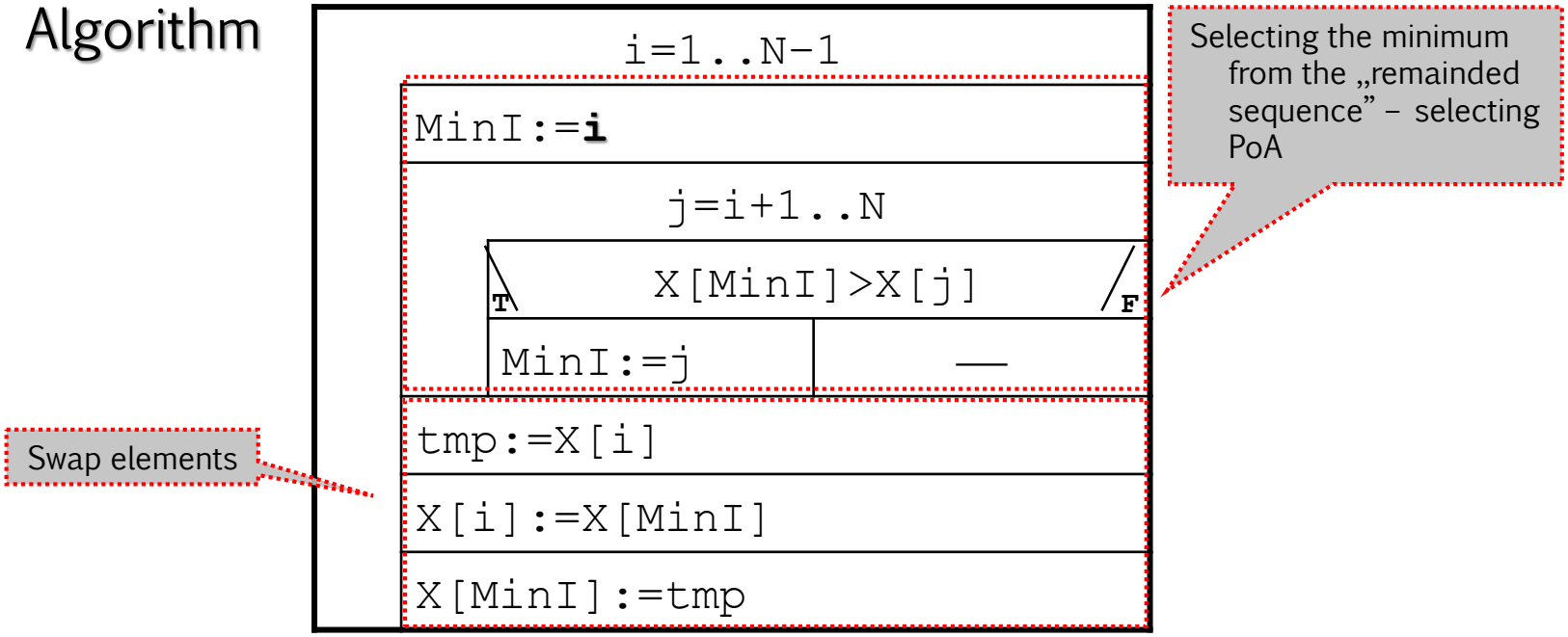
Minimum element goes to the first position.



Red elements are at the right position

Selection sort

Algorithm



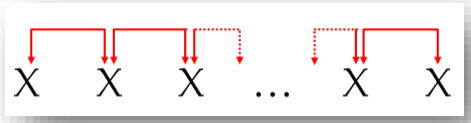
Number of comparisons: $1 + 2 + .. + N - 1 = N \cdot \frac{N - 1}{2}$

Number of moving: $3 \cdot (N - 1)$

Bubble sort

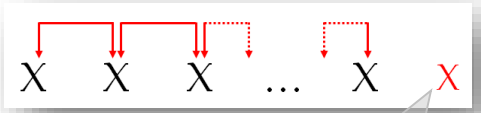
Idea:

› Compare all elements with the next one, if required, swap them!



The maximum one goes to the last place.

› Next do the same without the last element!



Others are going to their place, too.

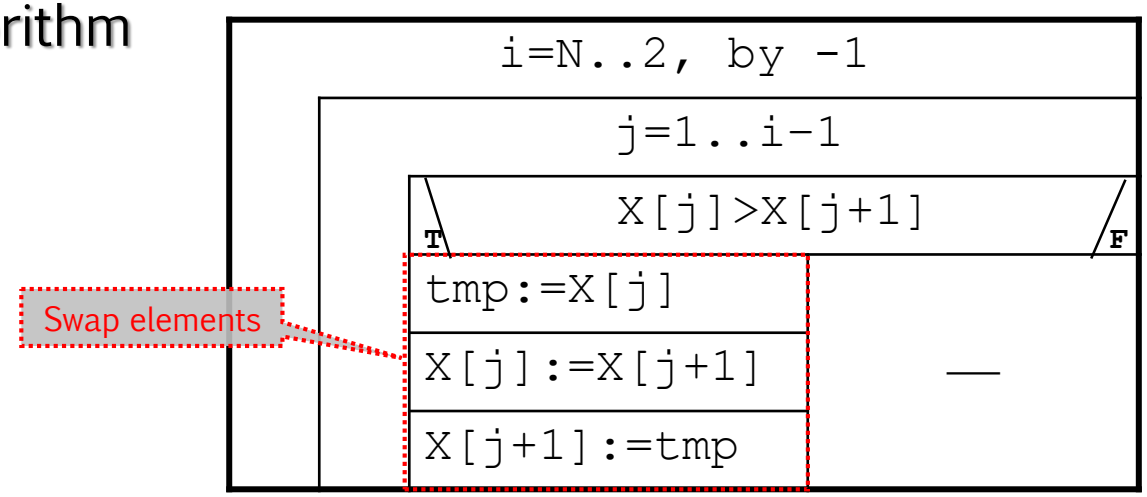
…

› Finally for the first two elements!

Red elements are at the right position

Bubble sort

Algorithm



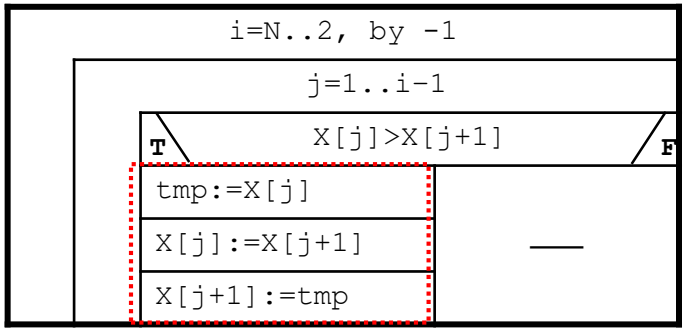
Number of comparisons: $1 + 2 + \ldots + N - 1 = N \cdot \frac{N-1}{2}$

Number of moving: $0 \ldots 3 \cdot N \cdot \frac{N-1}{2}$

Improved bubble sort

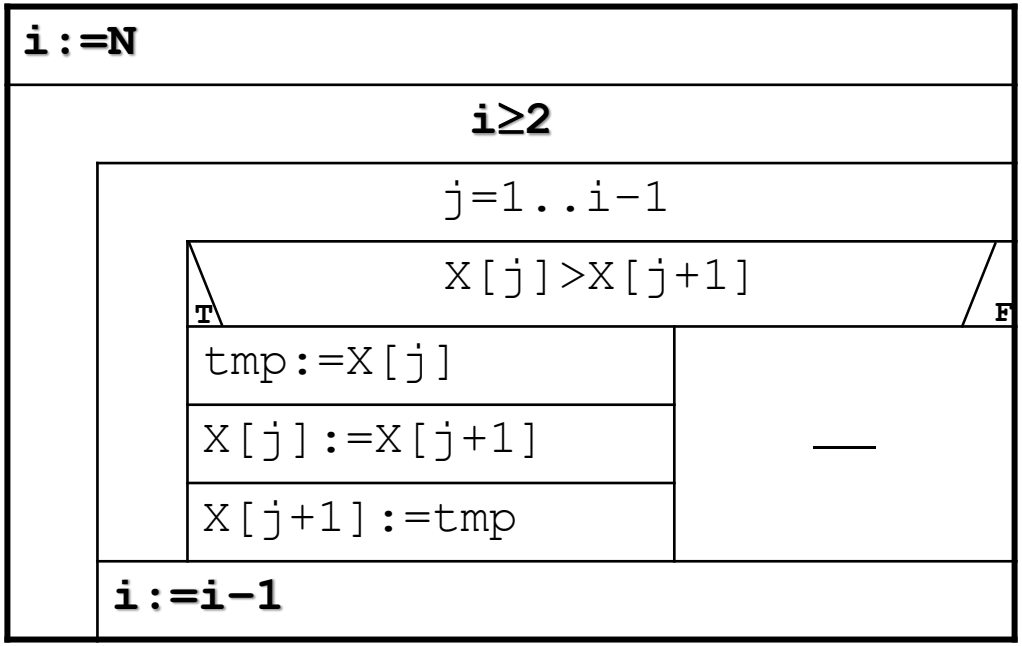
Observations:

- If the **nested loop** does not run the swap, sorting is finished yet.
- If the **nested loop**'s last swap was at position K, there are sorted elements from position K+1, outer loop can skip more then one step.

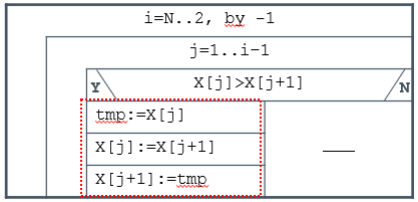


Improved bubble sort

Algorithm:(changed to conditional loop)

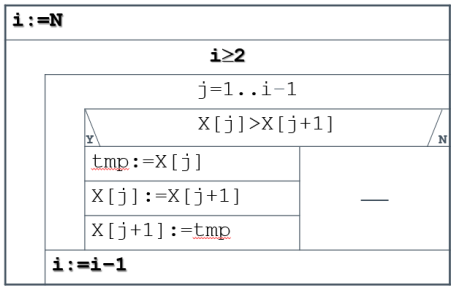


Improved bubble sort

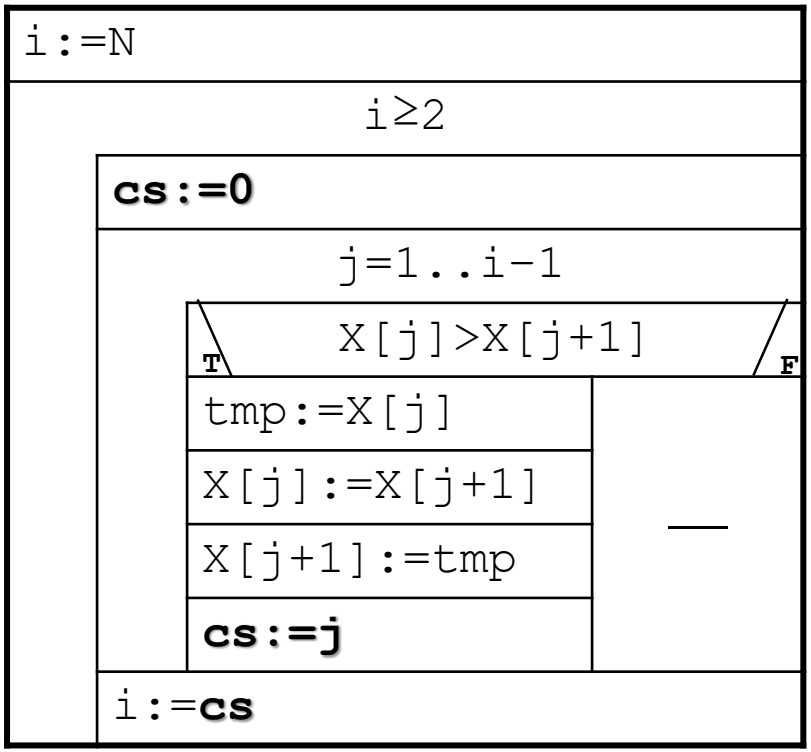


Algorithm

Rewrite it with conditional loop



Remember the last position of swap



Insertion sort

Idea:

› One element is sorted.

› The second one goes to before or after it, so now they two are sorted.

…

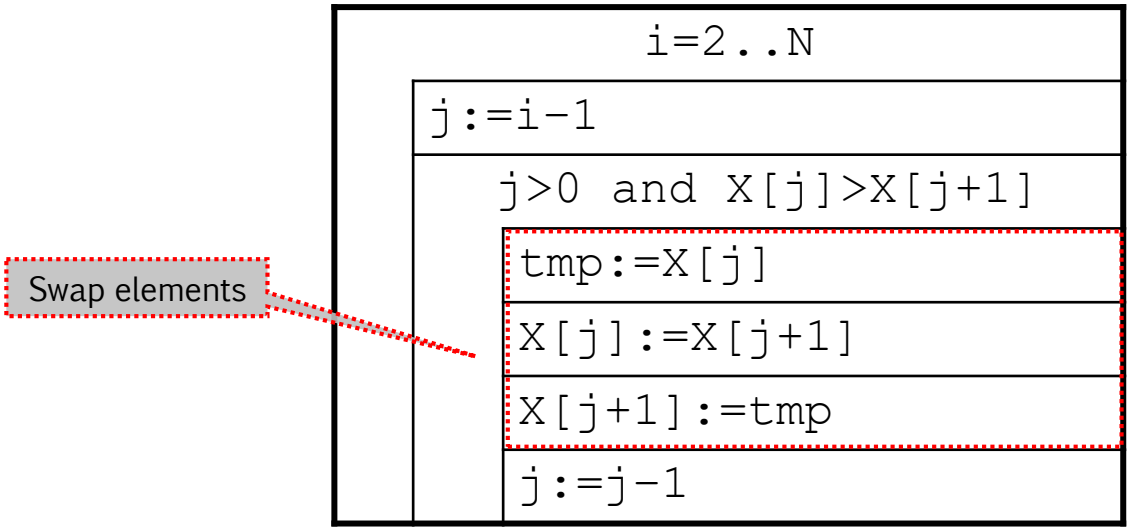
› The i^{th} one is checked and swapped in the sorted i–1 sequence while it gets its right position; so now these i elements are sorted.

› …

› Do this with the last one, too!

Insertion sort

Algorithm



Number of comparisons: $N-1 \ldots N \cdot \frac{N-1}{2}$

Number of moving: $0 \ldots 3 \cdot N \cdot \frac{N-1}{2}$

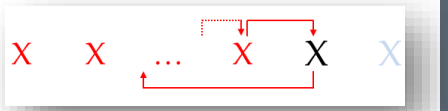
Improved insertion sort

Idea:

- One element is sorted.
- The second one goes to before or after it, so now they two are sorted.

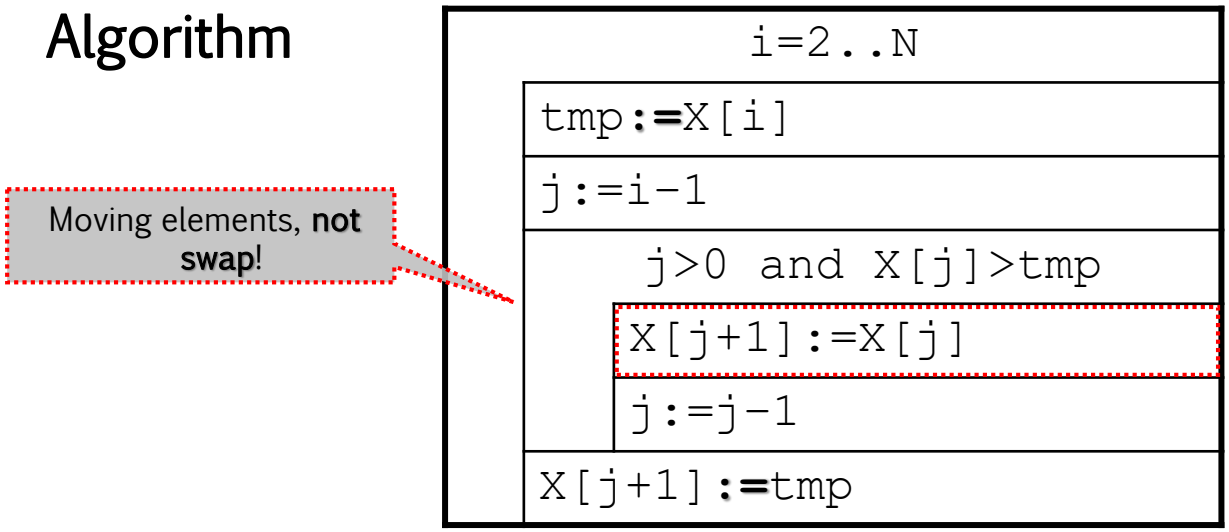
...

- Do **shift** back the greather elements than the i^{th} element and paste it (the i^{th} element) before them, so now these i elements are sorted.
- ...
- Do this with the last one, too!



Improved insertion sort

Algorithm



Number of comparisons: $N-1 \ldots N \cdot \frac{N-1}{2}$

Number of moving: $2 \cdot (N-1) \ldots (N+4) \cdot \frac{N-1}{2}$

Special sorting algorithms

Idea:

If we have special information about the series, we can use other methods.

Specification – sorting in N steps:

Input: $N \in \mathbb{N}$, $X_{1..N} \in \mathbb{Z}^N$

Elements are numbers

Output: $\mathbf{Y_{1..N}} \in \mathbb{Z}^N$

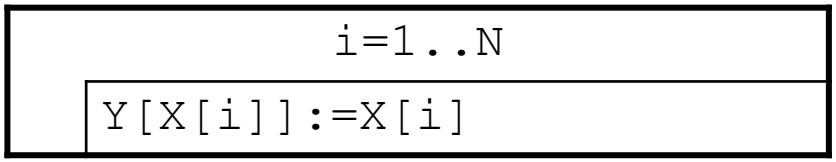
Precond.: $\mathbf{X \in Perm(1,\ldots,N)}$

All items occour only once!

Postcond.: `Sorted?(Y) and Y∈Perm(X)`

Sorting with deviding

Algorithm

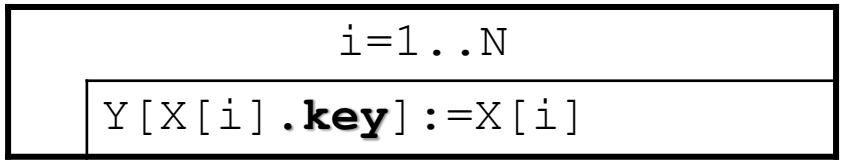


It could be: `Y[i]:=i` ☺

The task is interesting if `X[i]` is a record with a key field: a number between 1 and N:

```
X,Y: Array[1..N: Record(key: 1..N, …)]
```

Algorithm



Sorting with deviding and counting

With less resctriction in the precondition:

› the values are numbers between 1 and M ($M \geq N$),

› they can be repeated.

Specification:

Input: $N, M \in \mathbb{N}$, $X_{1..N} \in \mathbb{Z}^N$

Output: $Y_{1..N} \in \mathbb{Z}^N$

Precond.: $M \geq 1$ and $\forall i (1 \leq i \leq N): 1 \leq X_i \leq M$

Postcond.: $\text{Sorted?}(Y)$ and $Y \in \text{Perm}(X)$

Sorting with deviding and counting

Idea:

- First, count the **numbers** (how many times..) of **each value** (every „i”) in the sequence! (=**counting**)
- Next, define the **position** of „**first i**” 's value: this is the sum of the numbers of the less elements then „i” in the sequence + 1 (recursive counting)!
- Finally, check all elements in the sequence, put to the right **position** the element (with „i” value) and modify the position, because the next „first i” value must be shifting to the next position. (=**copy**)

Sorting with deviding and counting

Algorithm

Cnt[i]: how many occurance is available from i?

First[i]: where is the first of i^{th}?

```
Cnt[1..M]:=0
        i=1..N
    Cnt[X[i]]:=Cnt[X[i]]+1
First[1]:=1
        i=2..M
    First[i]:=First[i-1]+Cnt[i-1]
        i=1..N
    Y[First[X[i]]]:=X[i]
    First[X[i]]:=First[X[i]]+1
```

Number of comparisons: N

Number of additive operations: 3·M–3+2·N

Sorting with deviding and counting

Algorithm

Base set is $\mathbb{Z}$, so we count other assignments, too!

```
Cnt[1..M]:=0
                    i=1..N
    Cnt[X[i]]:=Cnt[X[i]]+1
First[1]:=1
                    i=2..M
    First[i]:=First[i-1]+Cnt[i-1]
                    i=1..N
    Y[First[X[i]]]:=X[i]
    First[X[i]]:=First[X[i]]+1
```

Number of moving: **N+1+M+2·N=M+3·N**

Number of additive op.s: 3·M–3+2·N

Counting sort

Idea:

- If deviding is not usable (unknown M), than first let's **count** *(define the order)*, **devide** after it only *(put it to the right place...)*!
- Use the simpliest **principle of repleacement**.
- `Cnt[i]` means the **count of** the **less** or **equal on the left side elements** of the i^{th} element!

  ↓

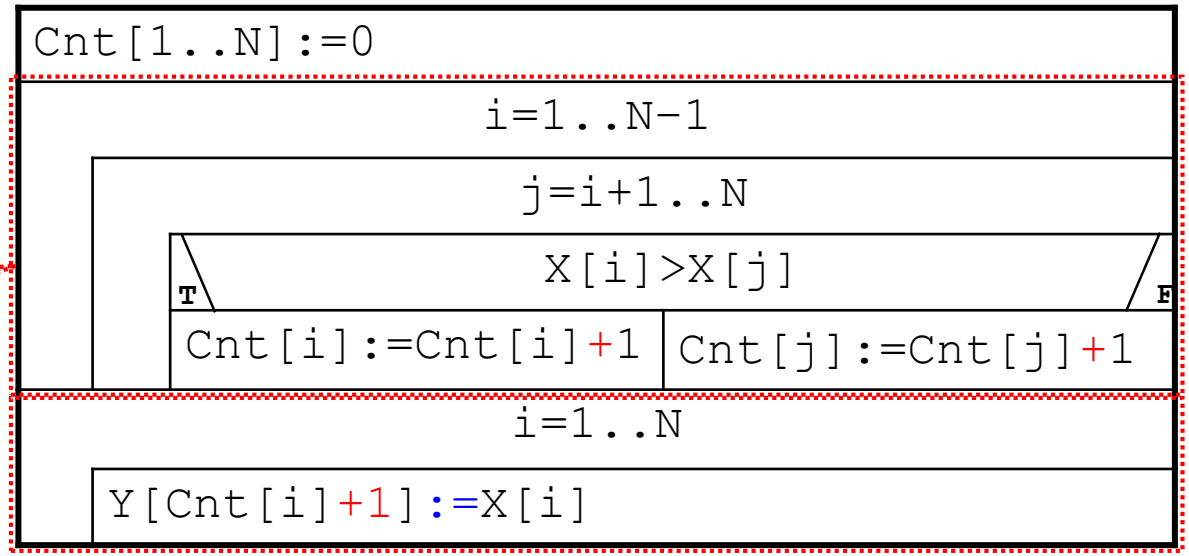
  `Cnt[i]+1` is useable as an **index of i^{th} element in the sorted sequence**.

Counting sort

Algorithm

Based on the principle of simple exchange sorting

copy



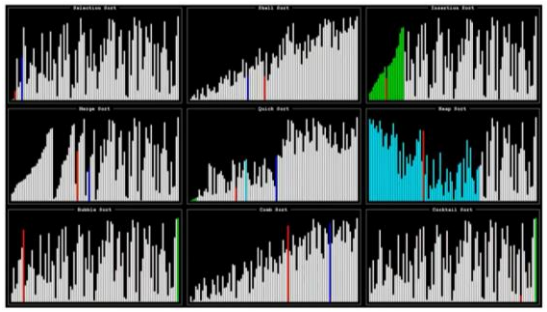
Number of comparisons: $1 + 2 + \ldots + N - 1 = N \cdot \frac{N-1}{2}$

Number of moving: N

Number of additive op.s: ~number of comparisons

Effectiveness of sorting algorithms

https://www.youtube.com/watch?v=ZZuD6iUe3Pc



https://www.toptal.com/developers/sorting-algorithms/