CS 350 Algorithms and Complexity

Winter 2019

Lecture 8: Decrease & Conquer (continued)

Andrew P. Black

Department of Computer Science

Portland State University

Finding the Median

- The Median of an array of numbers is the "middle" number, when sorted.
- We can obviously find the median by sorting the array, and then picking the $\lfloor \frac{n}{2} \rfloor$th element
- How much work is that (in average case)?

A. O(n)

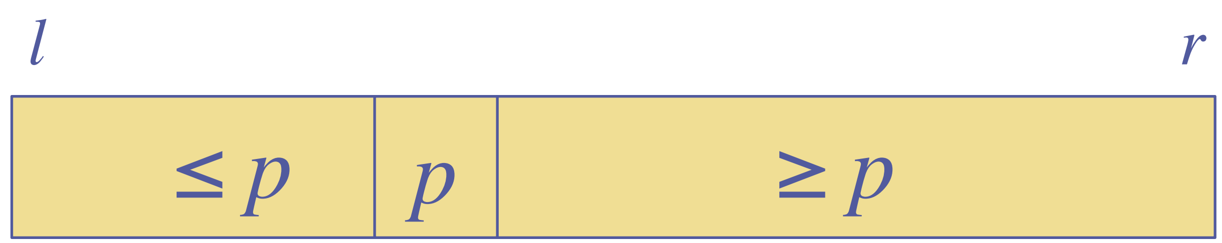
B. O(n lg n)

C. O(n^2)

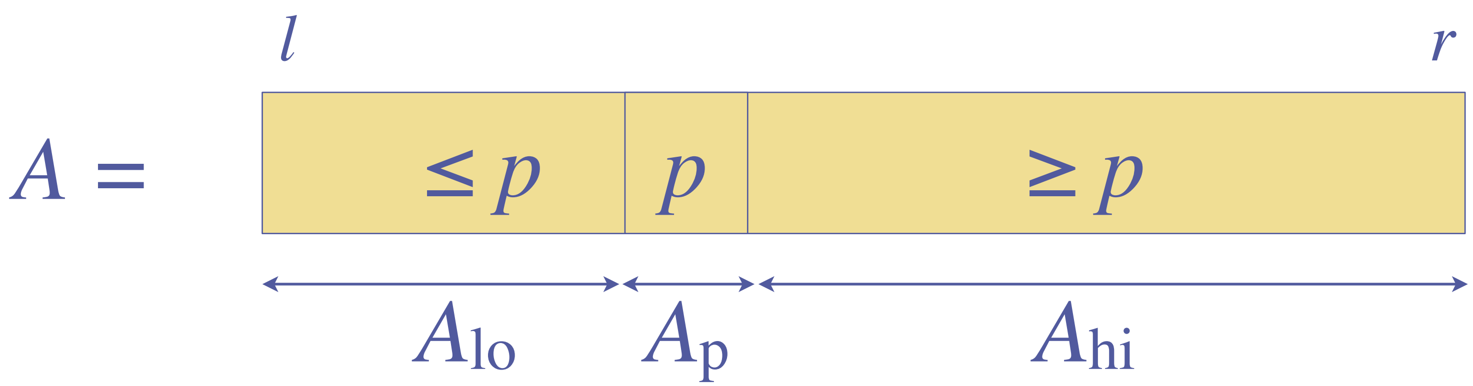
Median in Linear Time?

- Can we do better?
  - After all, sorting the whole array is more work than is needed to find the median
- What <u>smaller problem</u> will help us?
- Key insight: generalize the problem!
  - Rather than seeking an algorithm for the $\lfloor \frac{n}{2} \rfloor$th element, lets look for the kth element, $k \in [1..n]$

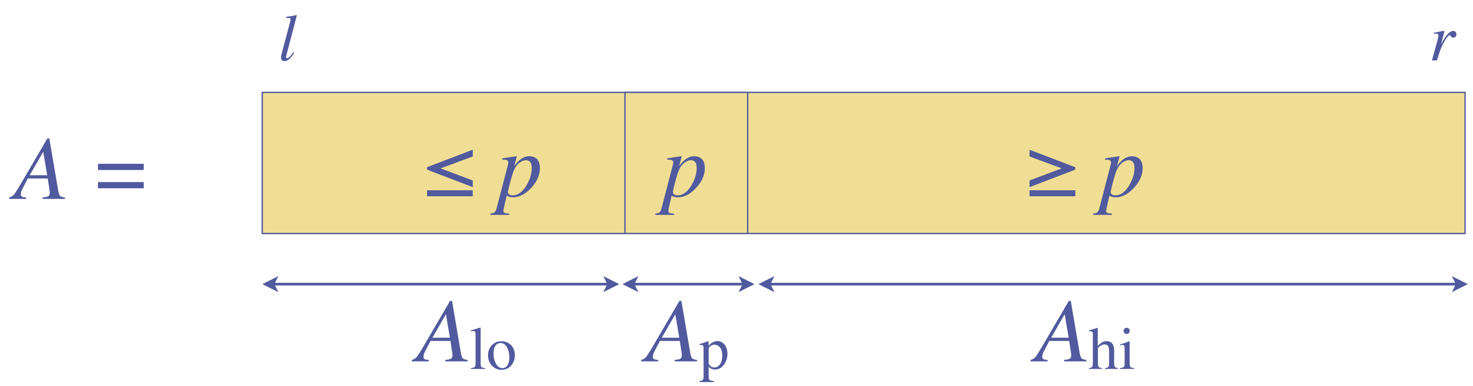
Suppose that we have a way of partitioning the array at element with value p:



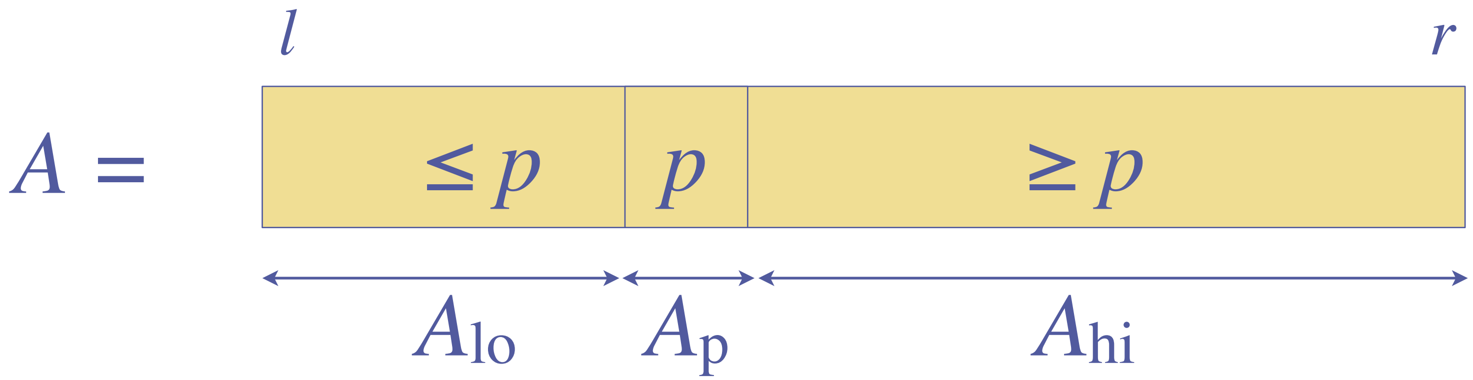
How can this help?



- Suppose that we are looking for the 10th element, and:
  - $|A_{\text{lo}}| = 5$
  - $|A_{\text{p}}| = 1$
  - Then we can seek the 4th element of A_{hi} instead
- We have reduced the problem size by a <u>variable amount</u>, in this case $|A_{\text{lo}}| + |A_{\text{p}}| = 6$



- Suppose that we are looking for the 10th element, and:
  - $|A_{\mathrm{lo}}| = 28$
  - Then we can seek the 10th element of A_{lo} instead
- We have reduced the problem size by a <u>variable amount</u>, in this case $|A_{\mathrm{p}}| + |A_{\mathrm{hi}}|$



- Suppose that we are looking for the 8th element, and:
    - $|A_{\text{lo}}| = 6$
    - $|A_{\text{p}}| = 2$
  - Then we can seek the 2nd element of A_{p} instead.
- We have now solved the problem, because all the elements of A_{p} are p

Variable-size decrease?

Variable-size decrease?

- What's the connection?
  - we would like to be able to find the n^{th} element
  - instead, partitioning lets us
    - pick an element, and, in linear time,
    - find its index (the s such that it is the s^{th} elem)
  - If $n = s$, we win!
  - if $n < s$, we continue in the left part, or
  - if $n > s$, we continue in the right part

Variable-size decrease?

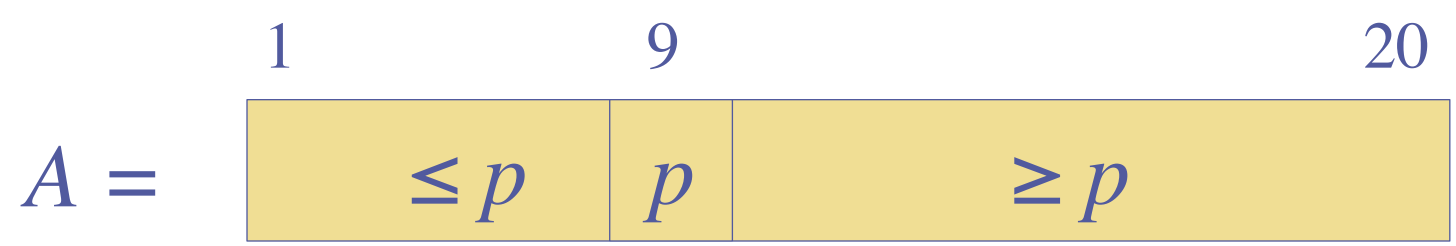
- Example
  - suppose that we have $A[1{:}20]$ and are looking for the 7th-smallest element:
  - run partition, find $s = 9$, say
  - Where do we look for the 7th-smallest element?

A: $A[1..20]$

B: $A[1..8]$

C: $A[1..9]$

D: $A[10..20]$



Variable-size decrease?

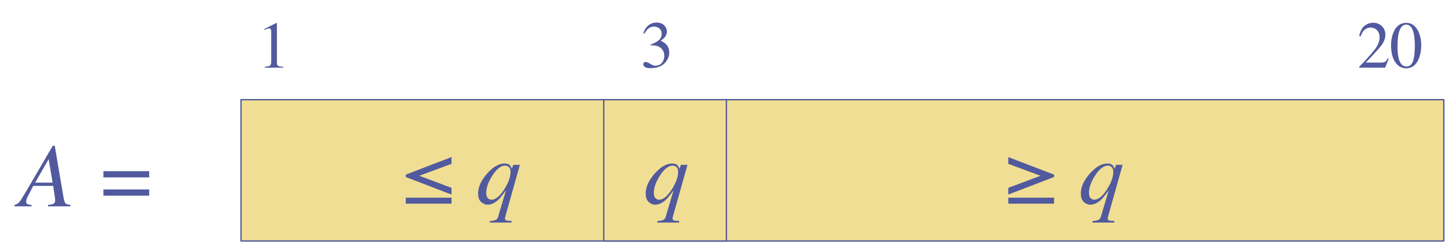
- A different run of the same example:
  - suppose that we have $A[1{:}20]$ and are looking for the 7th-smallest element:
  - run partition, find $s = 3$, say
  - Where do we look for the 7th-smallest element?

    A: $A[1..3]$

    B: $A[1..4]$

    C: $A[3..20]$

    D: $A[4..20]$



What's the Efficiency?

- Dasgupta's analysis shows that:

  if we can do the partition in O(n) time, and

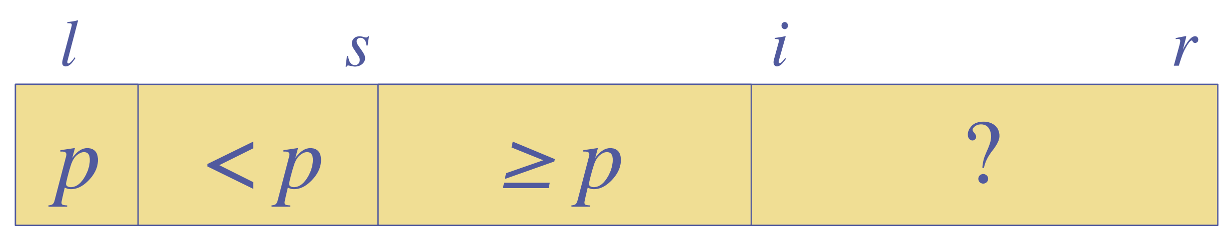
  the two parts are of roughly equal size

  then we can select the k^{th} element in O(n) time

- How can we do partition in O(n) time?
  - Lomuto Partition
  - Hoare Partition

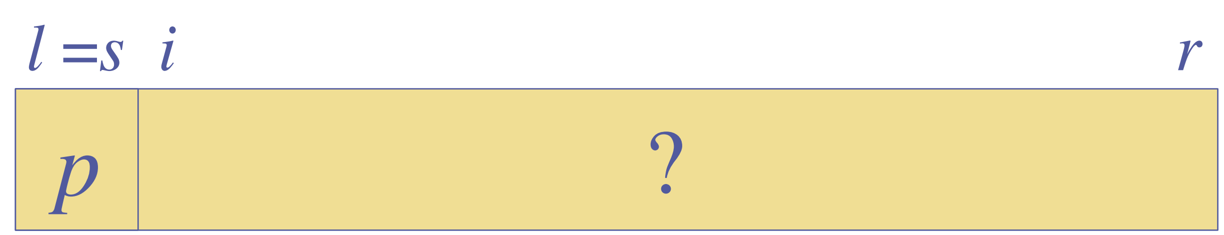
Lomuto Partition

- While algorithm is running:
- Invariant:
  - $A[l] = p \;\wedge\; A[l{+}1 \,..\, s] < p \;\wedge\; A[s{+}1 \,..\, i{-}1] \geq p \;\wedge\; l \leq s < i \leq r$

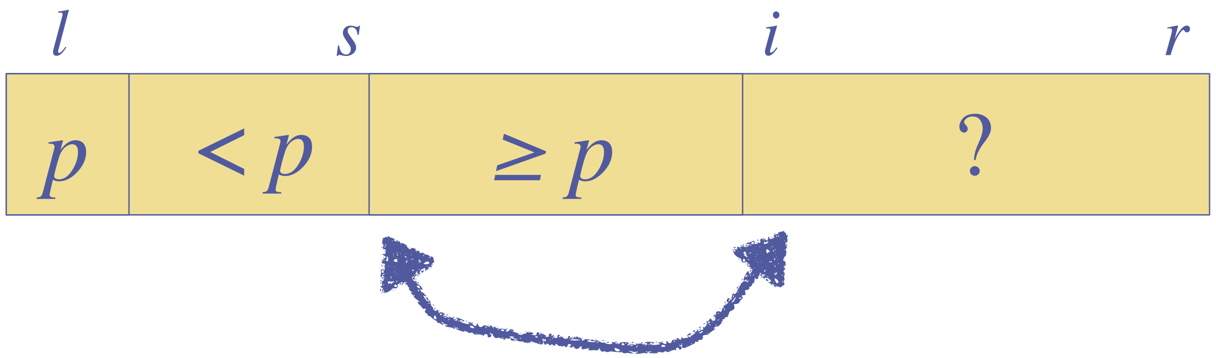


- Establish invariant initially:
  - $p \leftarrow A[l];\;\; s \leftarrow l;\;\; i \leftarrow s{+}1$

    // makes $< p$ interval and $\geq p$ intervals both empty

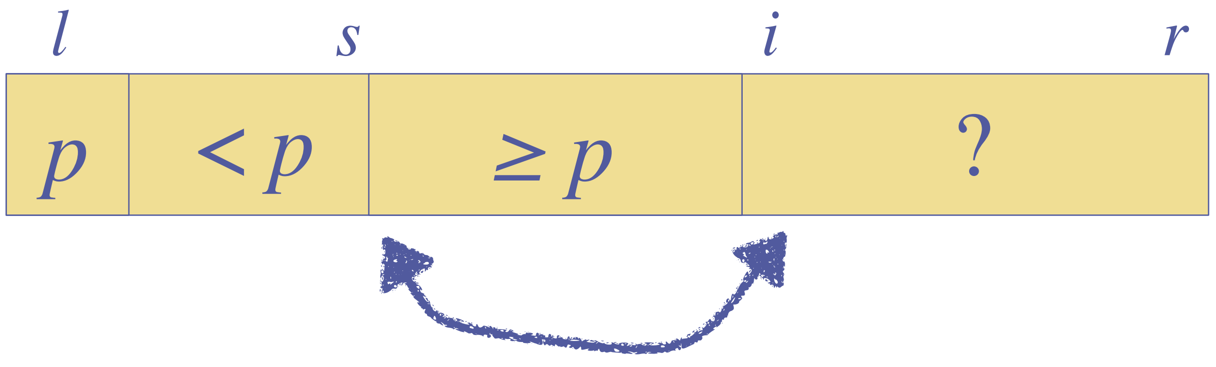


I don't like Lomuto Partition



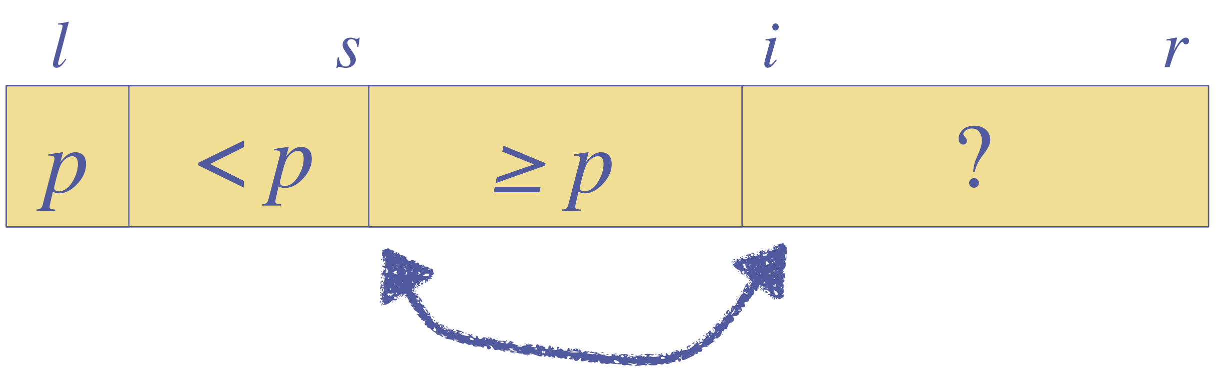
I don't like Lomuto Partition

- It does more swaps than necessary



I don't like Lomuto Partition

- It does more swaps than necessary



  - "half of the swap" is wasted
- It confuses students!
  - Quicksort does not use the Lumuto Partition
- It does not randomize the choice of p

Lomuto Partition:

Just forget about it!

How to pick the pivot?

- The choice is crucial
  - must be picked quickly
  - should shrink the sub-array substantially
    — ideally, $[l..s]$ and $[s..h]$ should be ≅ ½ $[l..h]$
    - if we can guarantee this, then T(n) = T(n/2) + O(n)
    - but that would require that the pivot be the median!
  - Instead, pick the pivot randomly

Efficiency analysis for random pivot

- If we are unlucky, and repeatedly choose the smallest element for the pivot, the array would shrink by just one element (the worst case)
- So we would be performing
  $$n + (n-1) + (n-2) + \ldots + \frac{n}{2} = \Theta(n^2)$$
  operations — but this is unlikely.
- It's also unlikely that we would stumble on the median each time (the best case).
- A "reasonably good" pivot is one between the 25th and 75th percentile. That's <u>half</u> of the available candidates. So we will get one, on average, after two random selections.
- After two partitions, we will shrink the problem to ¾ of its size, so
  $$T(n) \leq T\left(\frac{3n}{4}\right) + O(n)$$

Efficiency analysis for random pivot

- If we are unlucky, and repeatedly choose the smallest element for the pivot, the array would shrink by just one element (the worst case)
- So we would be performing

$$n + (n-1) + (n-2) + ... + \frac{n}{2} = \Theta(n^2)$$

  operations — but this is unlikely.
- It's also unlikely that we would stumble on the median each time (the best case).
- A "reasonably good" pivot is one between the 25th and 75th percentile. That's half of the available candidates. So we will get one, on average, after two random selections.
- After two partitions, we will shrink the problem to ¾ of its size, so

$$T(n) \leq T\left(\frac{3n}{4}\right) + O(n)$$

see reading on Medians

Hoare Partition

- Classic algorithm of computing
- Developed in 1959, published in 1961.
- Not only linear, but peculiarly efficient!
- Tony Hoare won the Turing Award for Quicksort, which is based on this algorithm

  ... and some other things!

Partition: CACM (Vol 4) July 1961

ALGORITHM 63
PARTITION
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  partition (A,M,N,I,J);  value M,N;
             array A;  integer M,N,I,J;
```

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin    real X;  integer F;
         F := random (M,N);  X := A[F];
         I := M;  J := N;
up:      for I := I step 1 until N do
                   if X < A [I] then go to down;
         I := N;
down:    for J := J  step −1 until M do
                   if A[J]<X then go to change;
         J := M;
change:  if I < J then begin exchange (A[I], A[J]);
                             I := I + 1; J := J − 1;
                             go to up
                       end
else     if I < F then begin exchange (A[I], A[F]);
                             I := I + 1
                       end
else     if F < J then  begin exchange (A[F], A[J]);
                             J := J − 1
                       end;
end      partition
```

Partition: CACM (Vol 4) July 1961

ALGORITHM 63
PARTITION
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  partition (A,M,N,I,J);  value M,N;
           array A;  integer M,N,I,J;
```

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin    real X;  integer F;
         F := random (M,N);  X := A[F];
         I := M;  J := N;
up:      for I := I step 1 until N do
                 if X < A [I] then go to down;
         I := N;
down:    for J := J  step −1 until M do
                 if A[J]<X then go to change;
         J := M;
change:  if I < J then begin exchange (A[I], A[J]);
                             I := I + 1; J := J − 1;
                             go to up
                       end
else     if I < F then begin exchange (A[I], A[F]);
                             I := I + 1
                       end
else     if F < J then  begin exchange (A[F], A[J]);
                             J := J − 1
                       end;
end      partition
```

Partition: CACM (Vol 4) July 1961

ALGORITHM 63
PARTITION
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

procedure partition (A,M,N,I,J); **value** M,N;
array A; **integer** M,N,I,J;

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin   real X; integer F;
        F := random (M,N); X := A[F];
        I := M; J := N;
up:     for I := I step 1 until N do
                if X < A [I] then go to down;
        I := N;
down:   for J := J step -1 until M do
                if A[J]<X then go to change;
        J := M;
change: if I < J then begin exchange (A[I], A[J]);
                            I := I + 1; J := J - 1;
                            go to up
                      end
else    if I < F then begin exchange (A[I], A[F]);
                            I := I + 1
                      end
else    if F < J then begin exchange (A[F], A[J]);
                            J := J - 1
                      end;
end     partition
```

Partition: CACM (Vol 4) July 1961

ALGORITHM 63
PARTITION
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

procedure partition (A,M,N,I,J); **value** M,N;
    array A; **integer** M,N,I,J;

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin    real X;  integer F;
         F := random (M,N);  X := A[F];
         I := M;  J := N;
up:      for I := I step 1 until N do
                     if X < A [I] then go to down;
         I := N;
down:    for J := J  step −1 until M do
                     if A[J]<X then go to change;
         J := M;
change:  if I < J then begin exchange (A[I], A[J]);
                               I := I + 1; J := J − 1;
                               go to up
                         end
else     if I < F then begin exchange (A[I], A[F]);
                               I := I + 1
                         end
else     if F < J then  begin exchange (A[F], A[J]);
                               J := J − 1
                         end;
end      partition
```

Partition: CACM (Vol 4) July 1961

ALGORITHM 63
PARTITION
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  partition (A,M,N,I,J);  value M,N;
           array A;  integer M,N,I,J;
```

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin    real X;  integer F;
         F := random (M,N);  X := A[F];
         I := M;  J := N;
up:      for I := I step 1 until N do
                if X < A [I] then go to down;
         I := N;
down:    for J := J  step -1 until M do
                if A[J]<X then go to change;
         J := M;
change:  if I < J then begin exchange (A[I], A[J]);
                             I := I + 1; J := J - 1;
                             go to up
                       end
else     if I < F then begin exchange (A[I], A[F]);
                             I := I + 1
                       end
else     if F < J then  begin exchange (A[F], A[J]);
                             J := J - 1
                       end;
end      partition
```

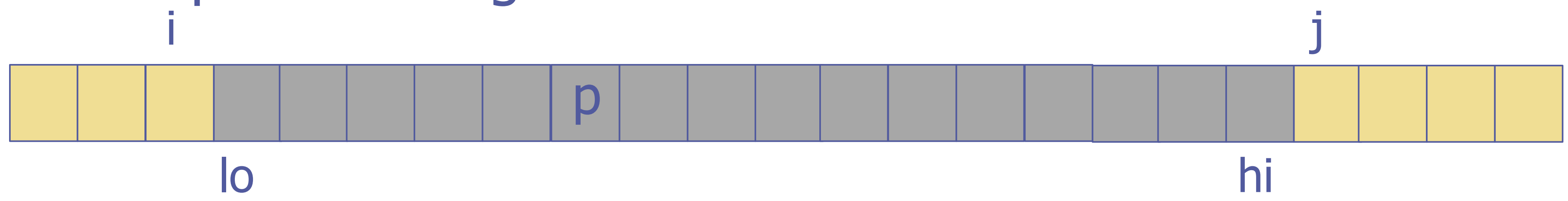
Important features:

1. random pivot
2. double-ended search
3. works in place
4. two outputs

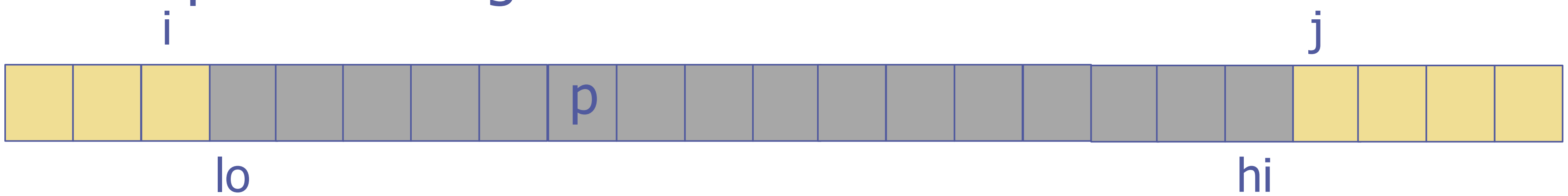
Hoare Partition

```
method partition(A, lo, hi) {
  def pivotIndex = randomBetween(lo)and(hi)
  def pivot = A[pivotIndex]
  var i := lo−1
  var j := hi+1
  while {
    do { i := i + 1 }
      while { (i <= hi).andAlso {A[i] <= pivot} }
    do { j := j − 1 }
      while { (j >= lo).andAlso {A[j] >= pivot} }
    i < j
  } do { exchange(A, i, j) }
  if (i < pivotIndex) then { exchange(A, i, pivotIndex) ;  i := i + 1 }
   elseif (j > pivotIndex) then { exchange(A, pivotIndex, j) ; j := j − 1 }
  list.with(i, j)
}
```

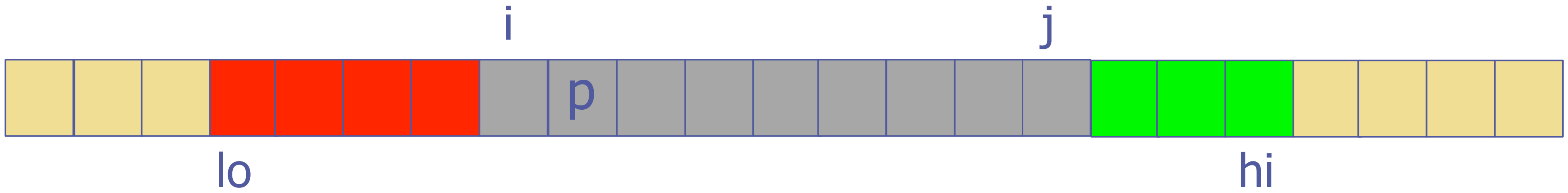
Before partition begins:



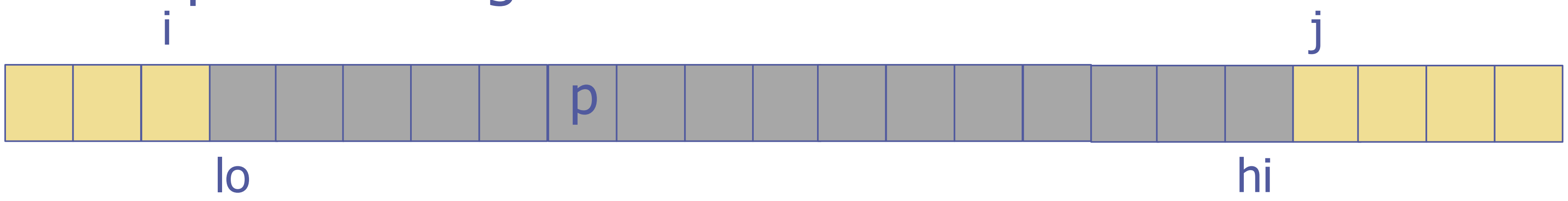
Before partition begins:



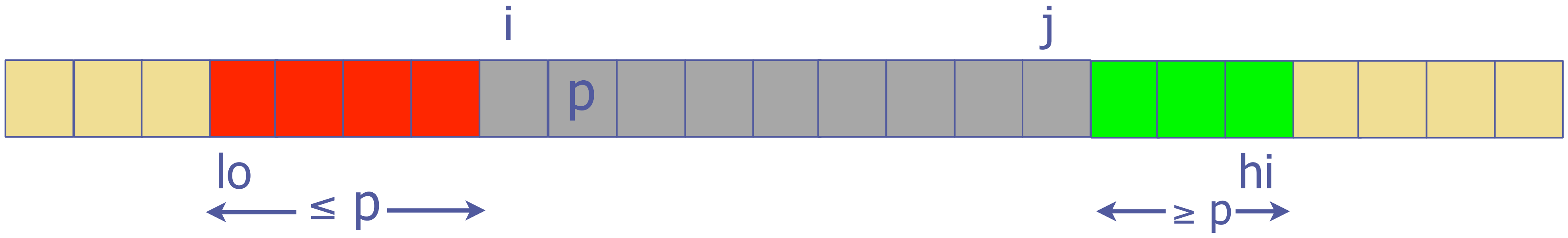
Leave elements that are already in the right place:



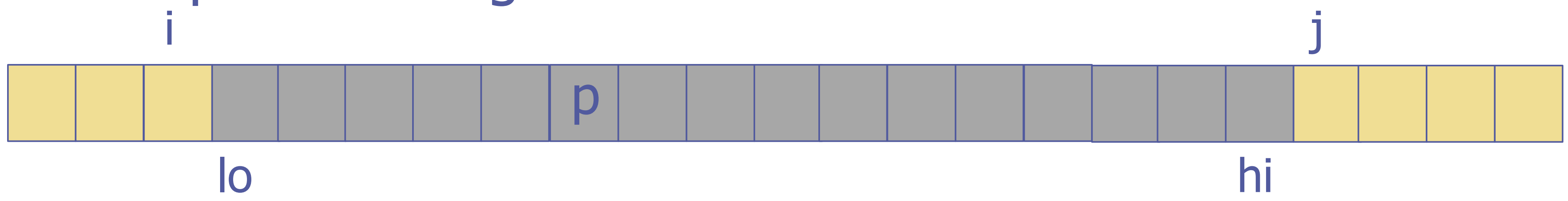
Before partition begins:



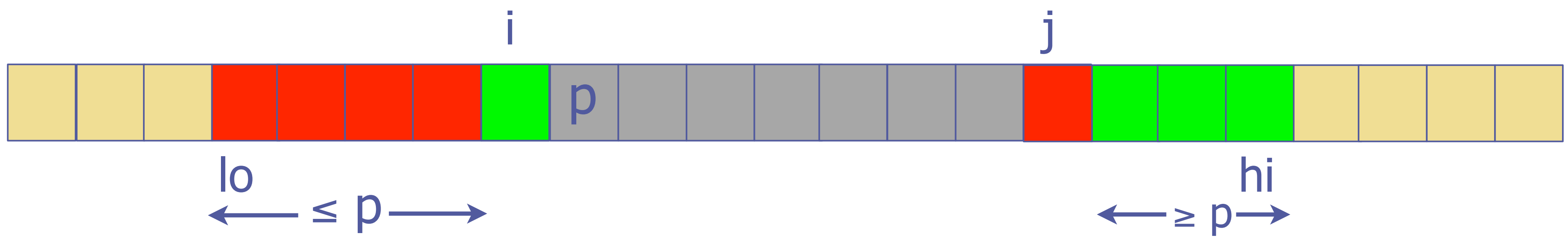
Leave elements that are already in the right place:



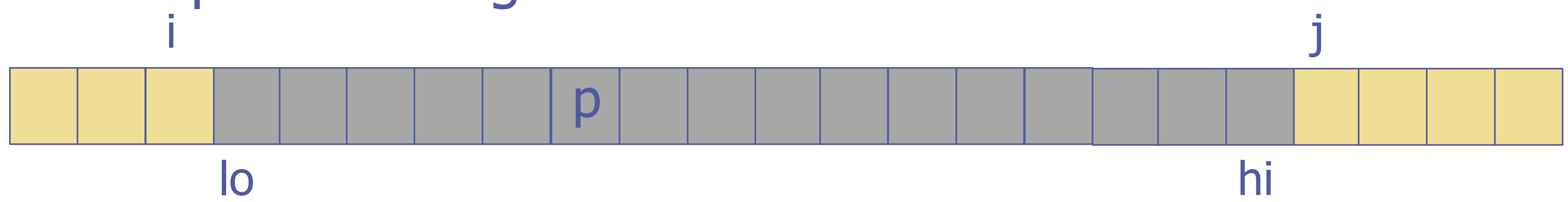
Before partition begins:



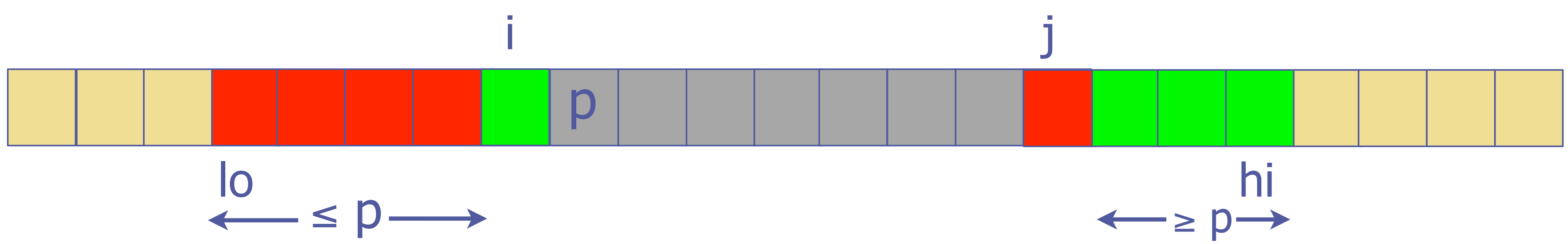
Leave elements that are already in the right place:



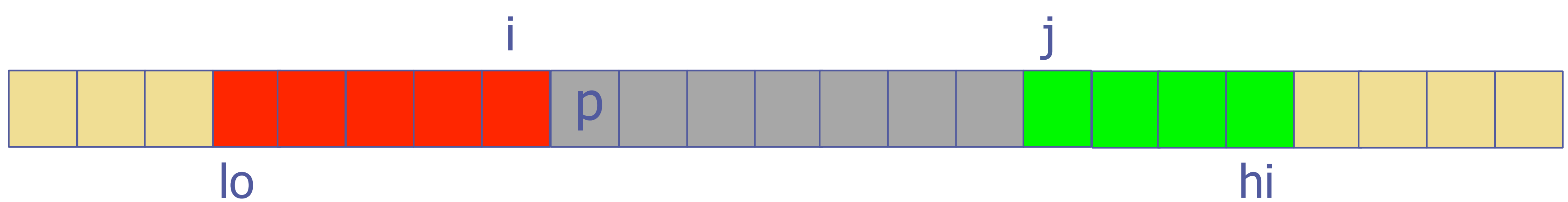
Before partition begins:



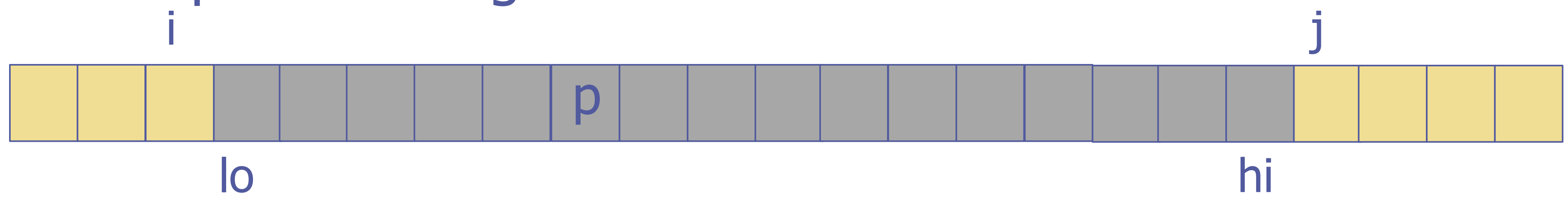
Leave elements that are already in the right place:



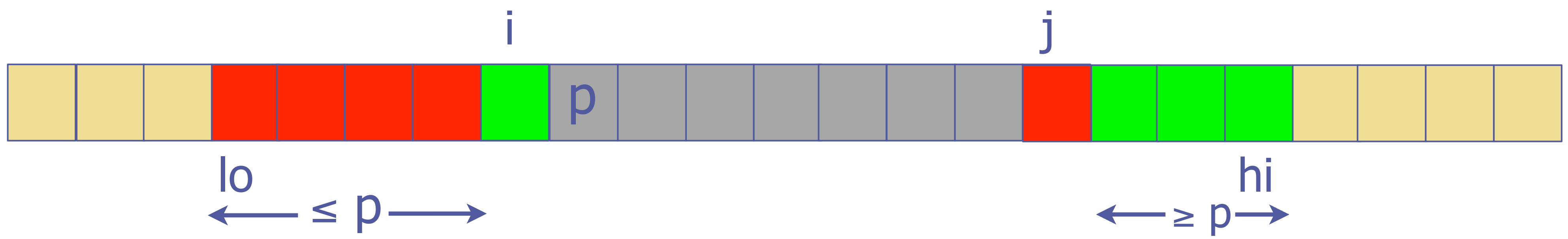
Now a[i] ≥ p ≥ a[j], so swap a[i] and a[j]:



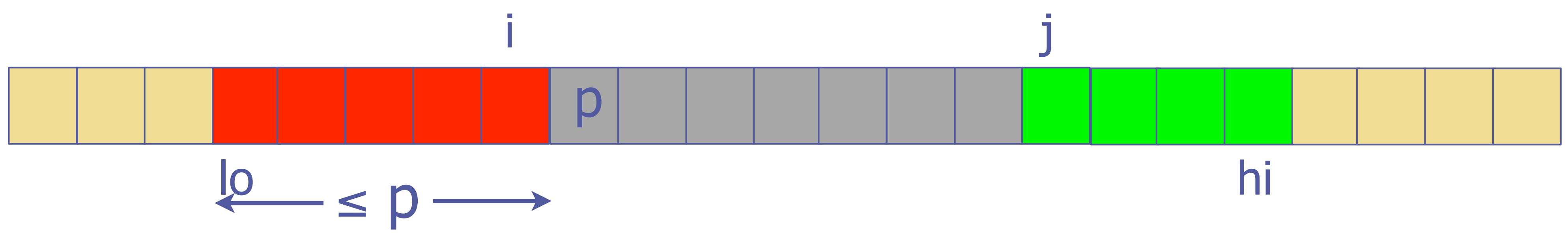
Before partition begins:



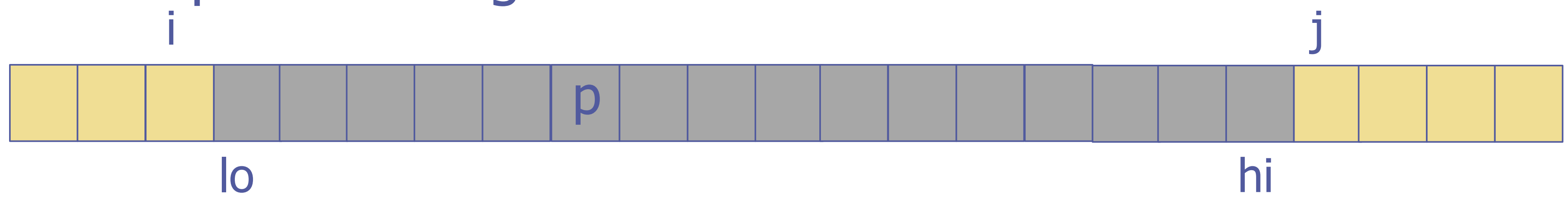
Leave elements that are already in the right place:



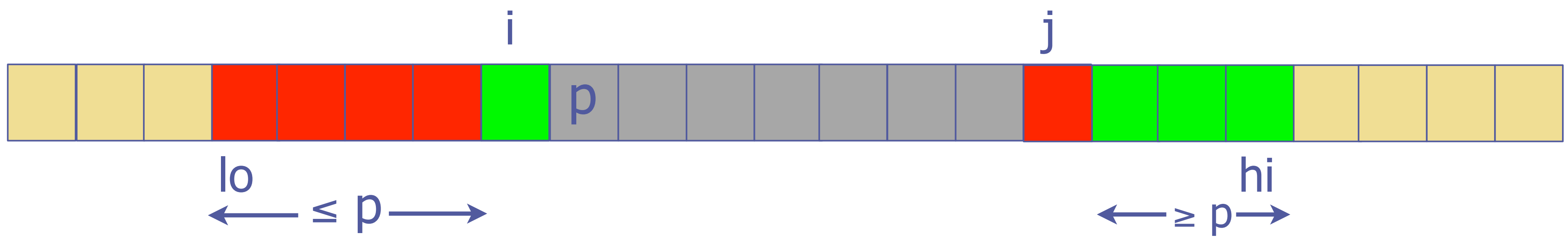
Now a[i] ≥ p ≥ a[j], so swap a[i] and a[j]:



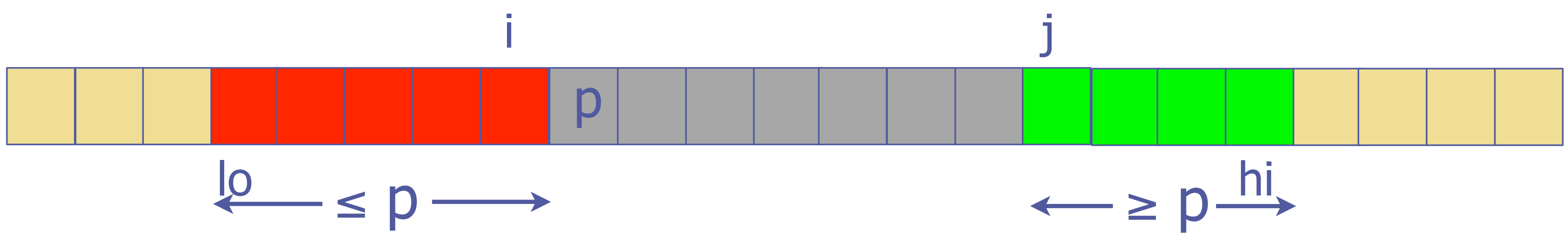
Before partition begins:



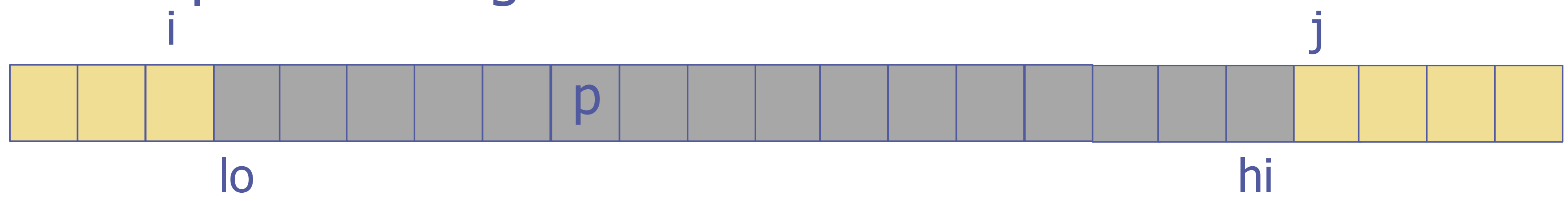
Leave elements that are already in the right place:



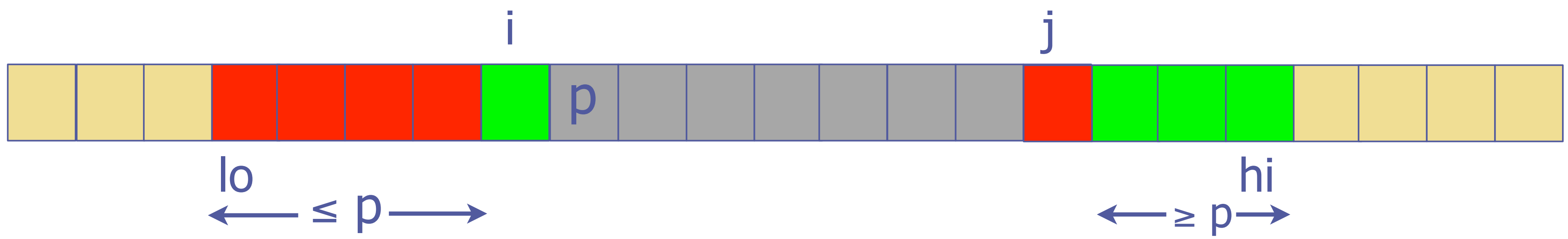
Now a[i] ≥ p ≥ a[j], so swap a[i] and a[j]:



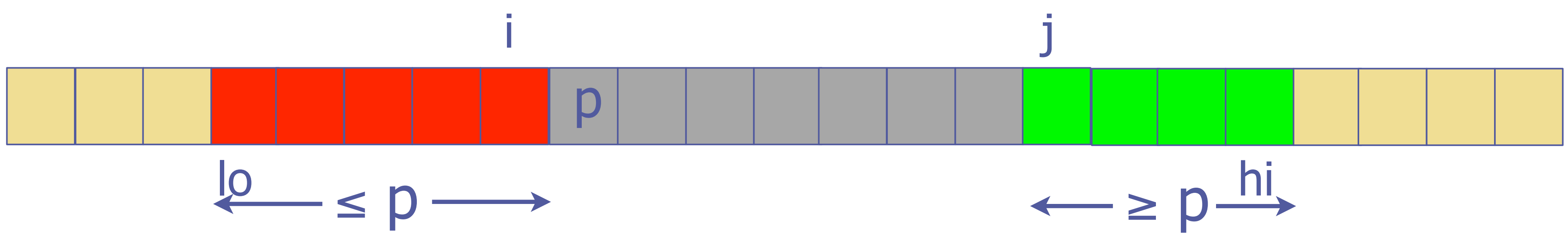
Before partition begins:



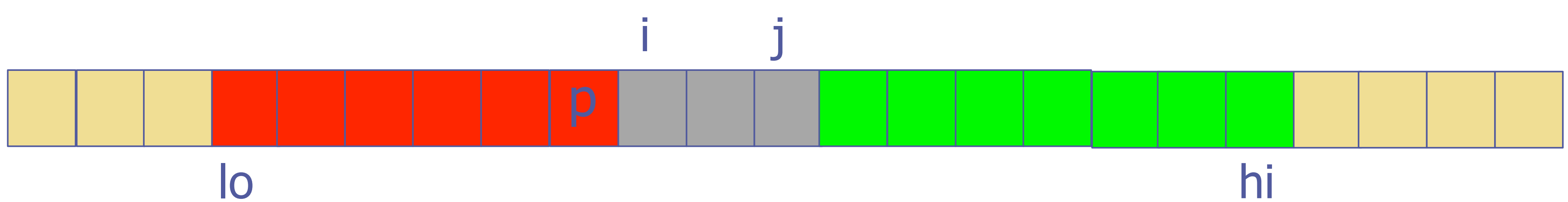
Leave elements that are already in the right place:



Now a[i] ≥ p ≥ a[j], so swap a[i] and a[j]:

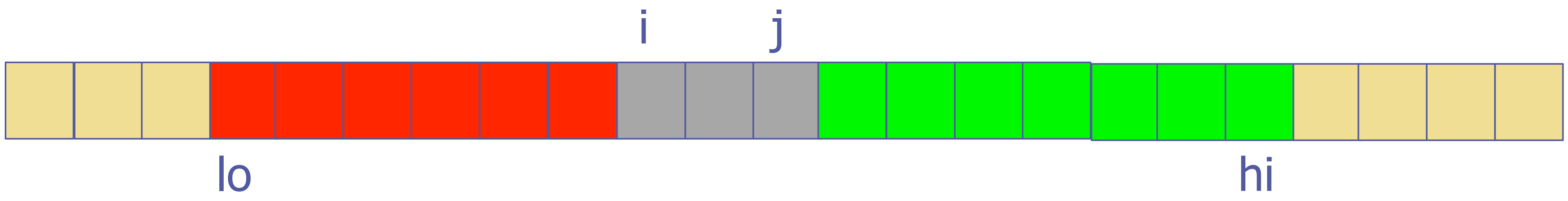


And continue ...

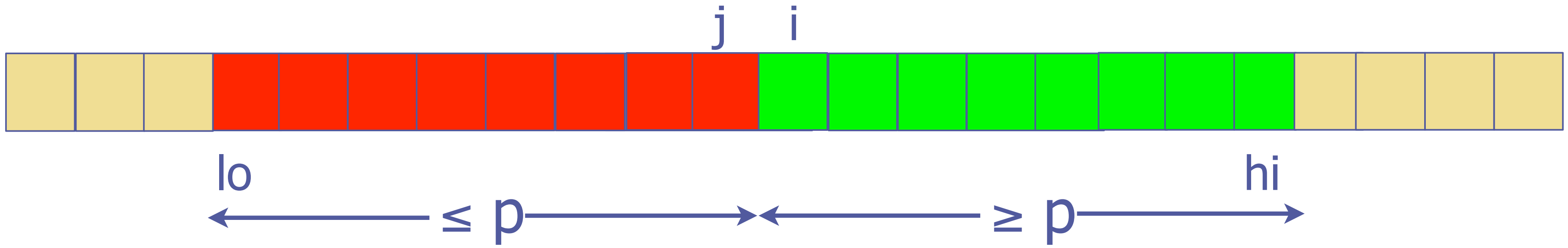


when do we stop?

And continue ...

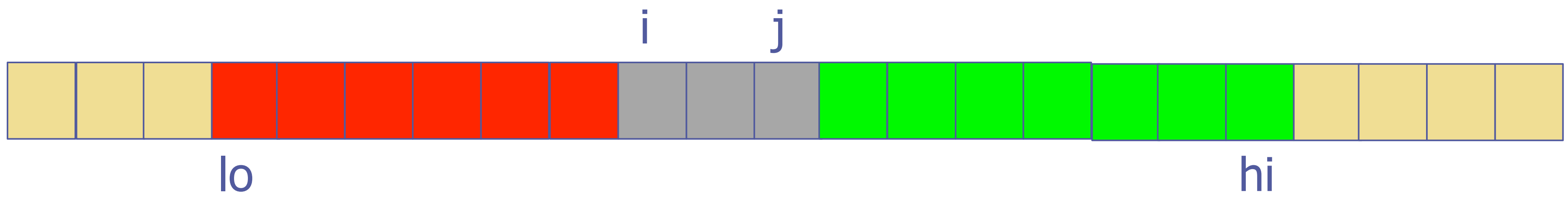


until i and j cross!

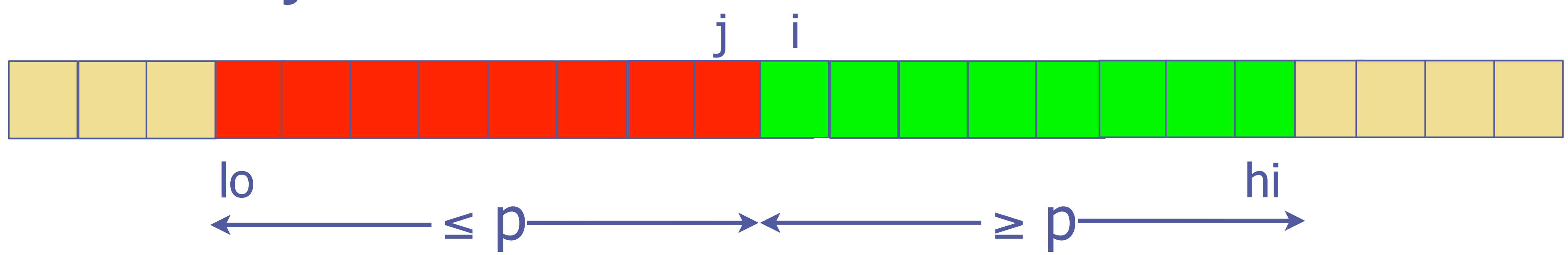


when do we stop?

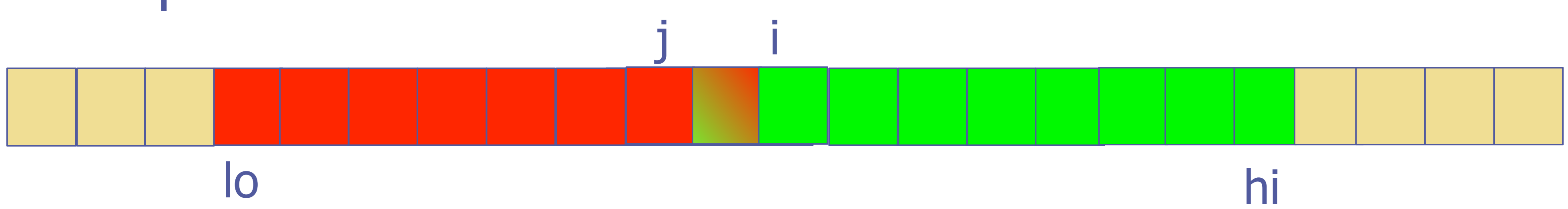
And continue ...



until i and j cross!

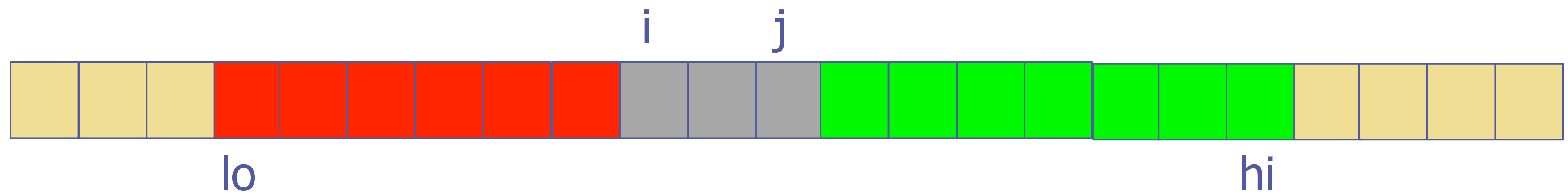


is this possible?

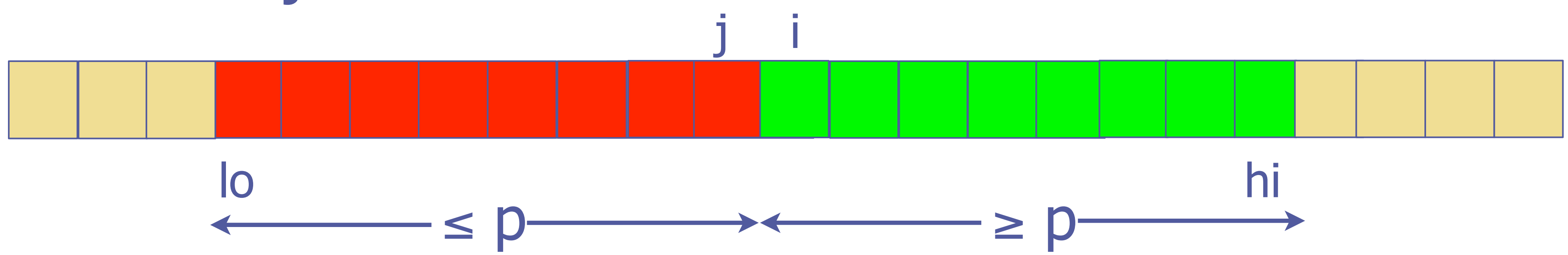


when do we stop?

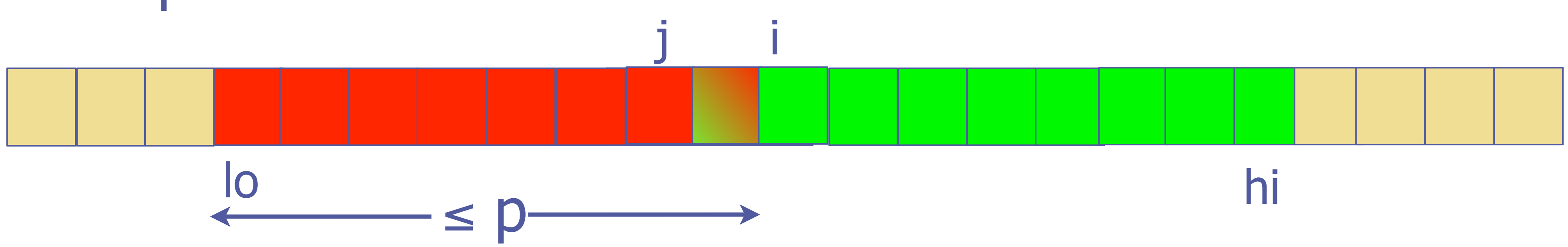
And continue ...



until i and j cross!

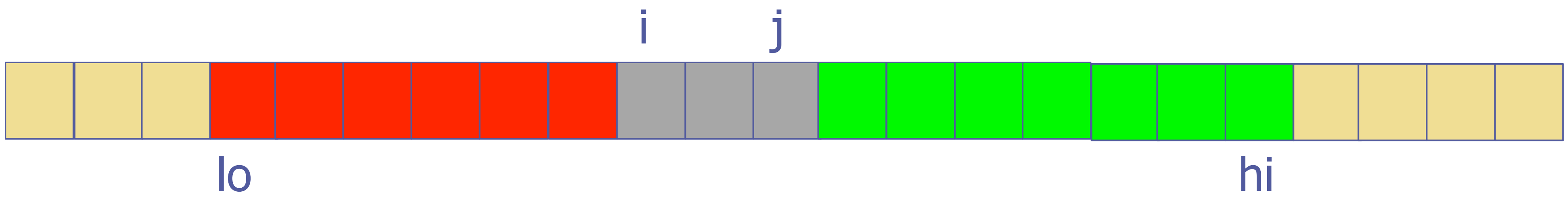


is this possible?

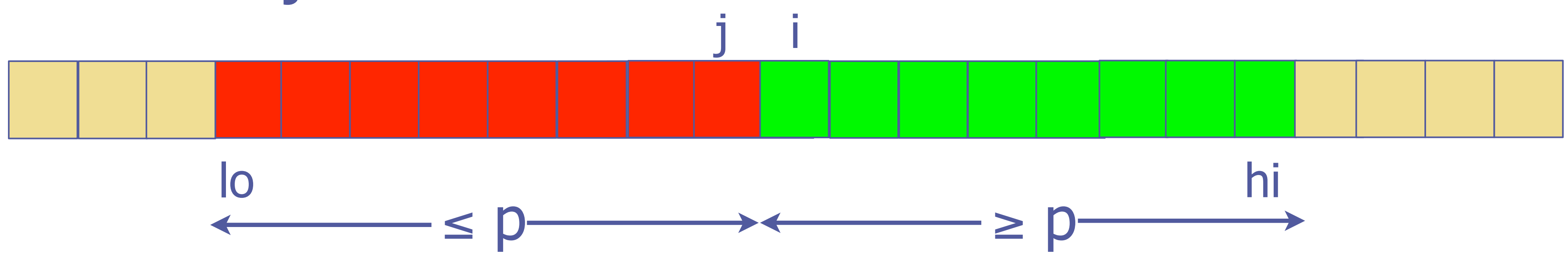


when do we stop?

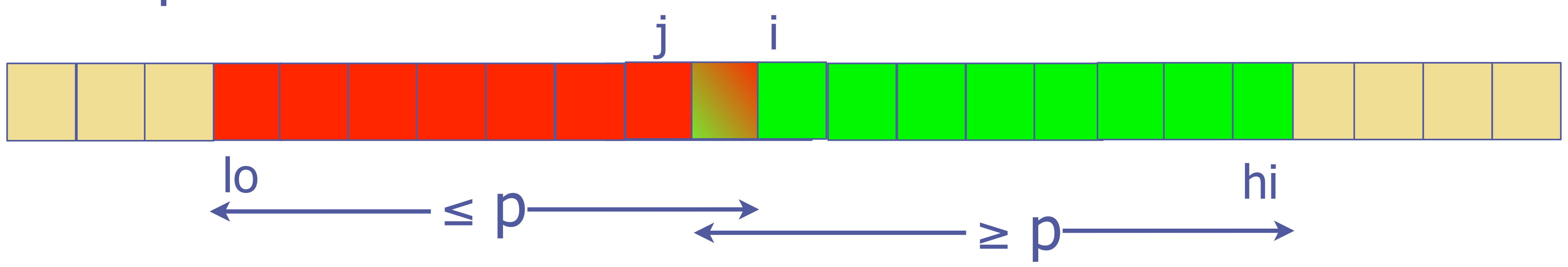
And continue ...



until i and j cross!

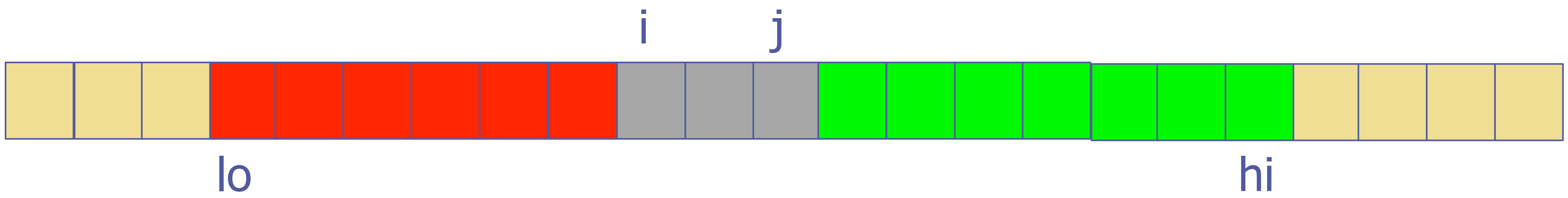


is this possible?

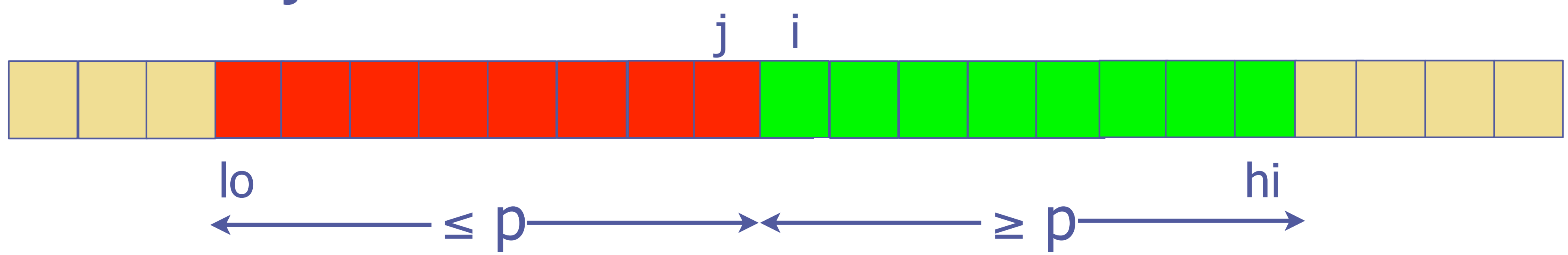


when do we stop?

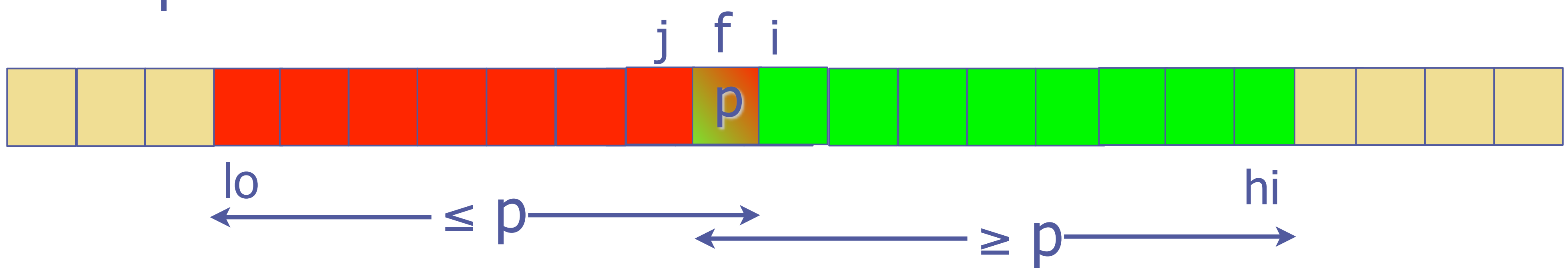
And continue ...



until i and j cross!

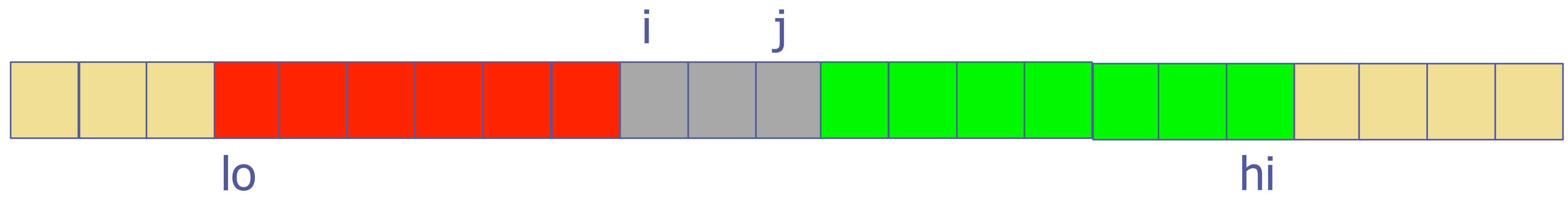


is this possible?

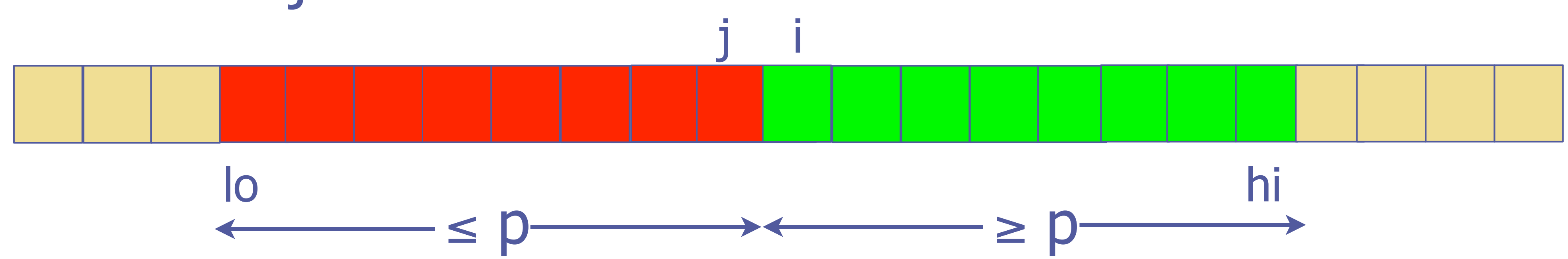


when do we stop?

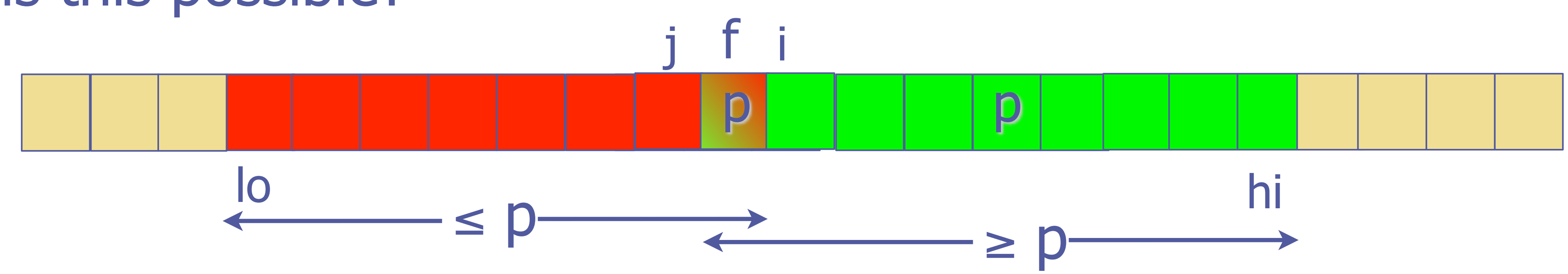
And continue ...



until i and j cross!

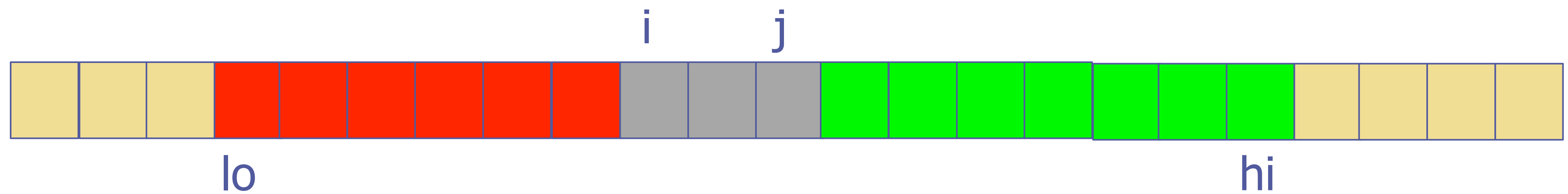


is this possible?

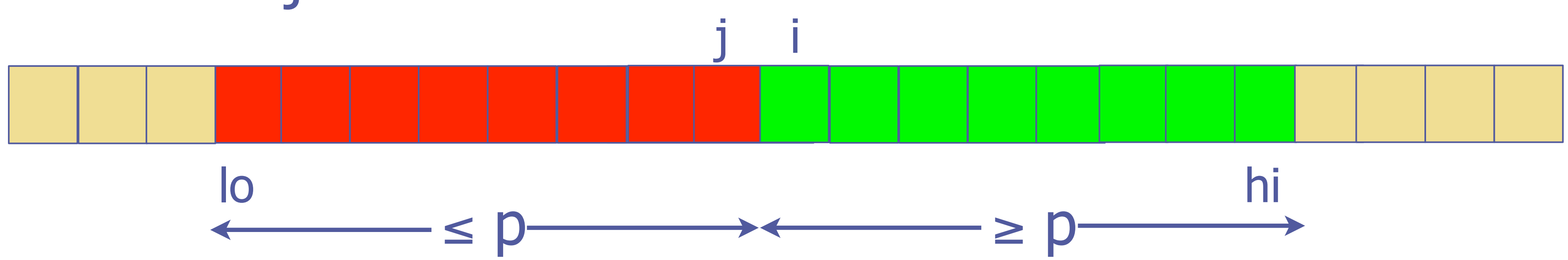


when do we stop?

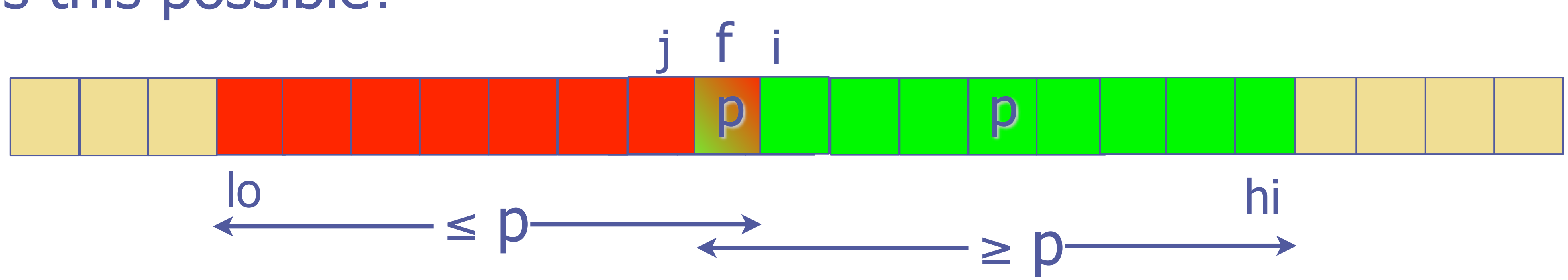
And continue ...



until i and j cross!



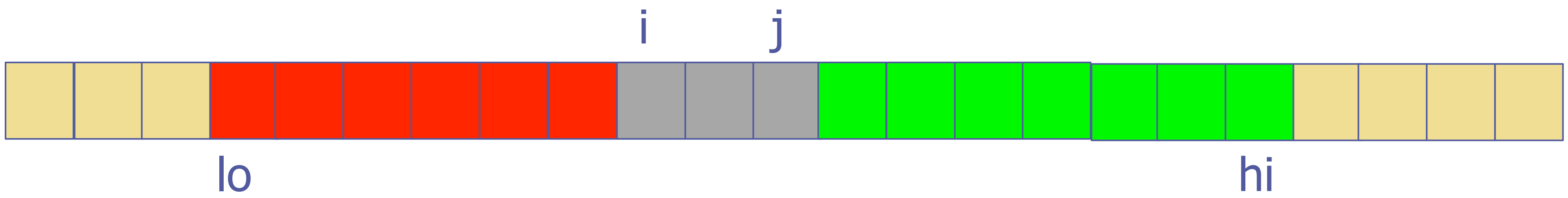
is this possible?



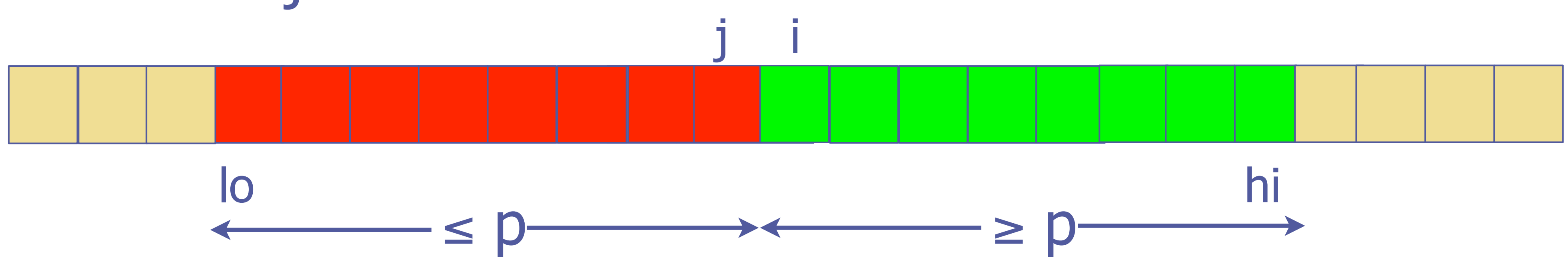
if i < f, exchange elements at i and f and increment i

when do we stop?

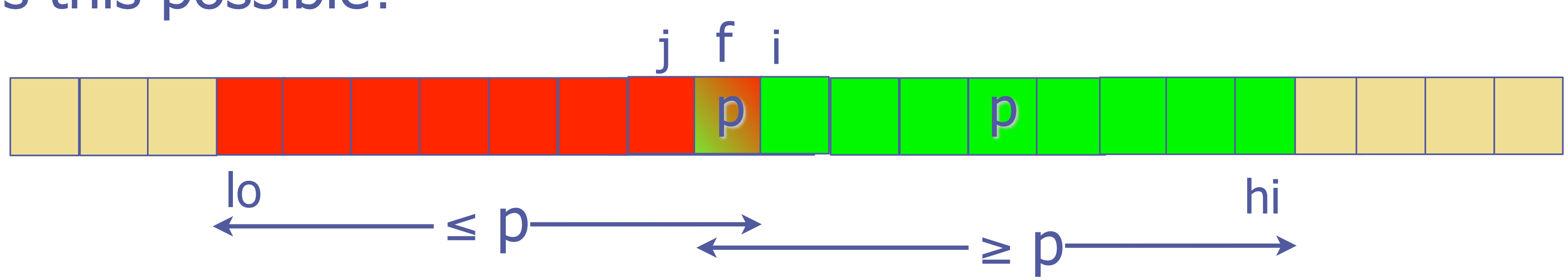
And continue ...



until i and j cross!



is this possible?



if i < f, exchange elements at i and f and increment i

if j > f, exchange elements at j and f and decrement j

Hoare's Partition

- Classic algorithm!
- beautiful and peculiarly efficient
- It can (and has been) improved upon
- To understand it better:
  - code it up
  - watch animations

12 Coins

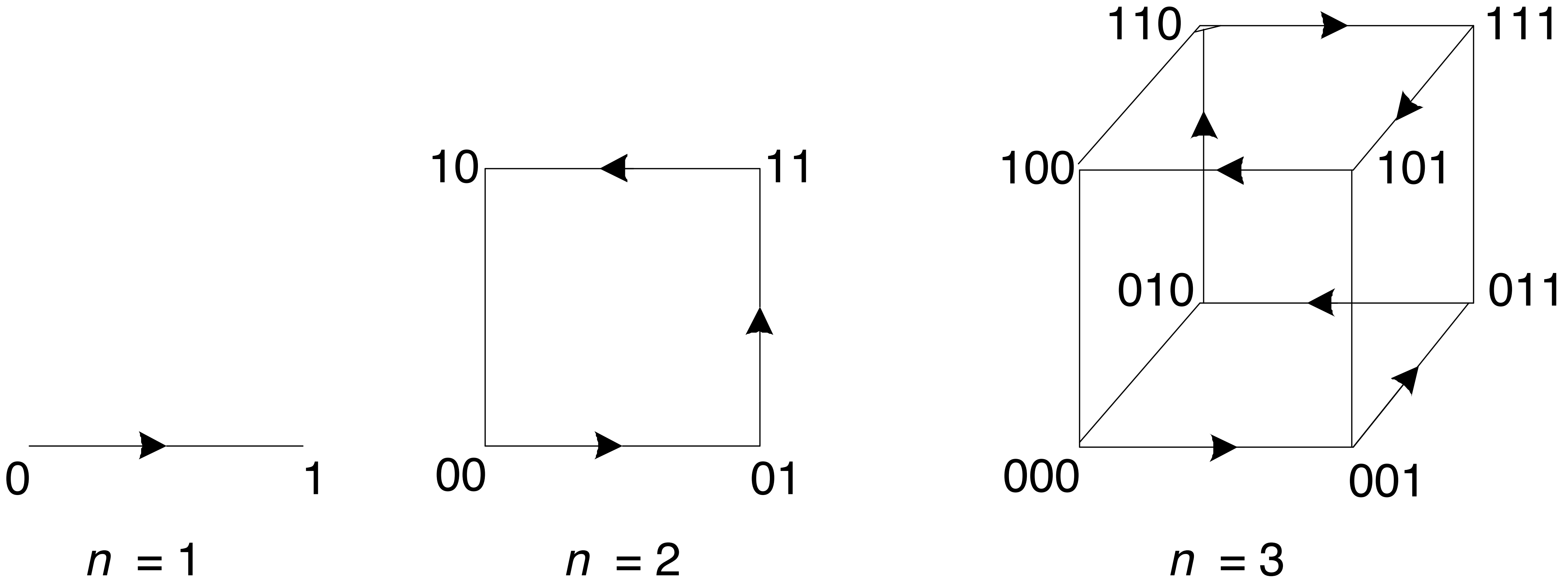
- This problem is originally stated as:
  - You have a balance scale and 12 coins, 1 of which is counterfeit. The counterfeit weigh less or more than the other coins. Can you determine the counterfeit in 3 weightings, and tell if it is heavier or lighter?
- A harder and more general problem is:
  - For some given n > 1, there are $(3^n - 3)/2$ coins, 1 of which is counterfeit. The counterfeit weigh less or more than the other coins. Can you state a priori n weighting experiments with a balance, with which you determine the counterfeit coin, and tell if it is heavier or lighter?

Problem: Gray Code

Use the decrease-by-one technique (*Algorithm BRGC*) to generate the binary reflected Gray code for n = 4.

Problem: Gray Code

Use the decrease-by-one technique (*Algorithm BRGC*) to generate the binary reflected Gray code for n = 4.



Problem: Gray Code Algorithm

- Trace the following algorithm for generating the Binary Gray Code of order 4.

```
Start with code = 0000
    output code
for i = 1 to 15 do:
    b ← position* of least significant 1 in binary rep of i
    code ← code XOR (bit b)
    output code
```

**least significant bit is 1*

Nim

- 1 pile of n chips
- Players take turns removing $1 \le k \le m$ chips
- The player removing the last chip wins

$$m = 4$$

Nim

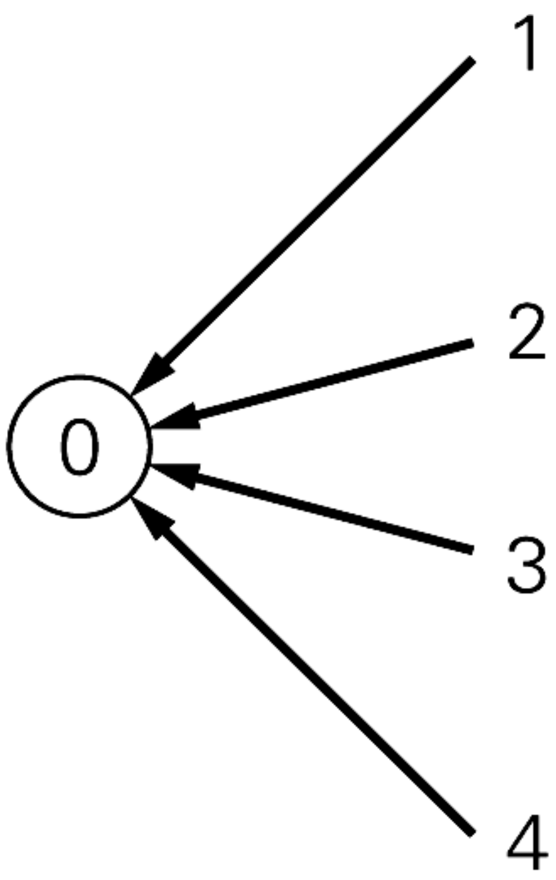
- 1 pile of n chips
- Players take turns removing $1 \leq k \leq m$ chips
- The player removing the last chip wins

$$m = 4$$

Nim

- 1 pile of n chips
- Players take turns removing $1 \le k \le m$ chips
- The player removing the last chip wins

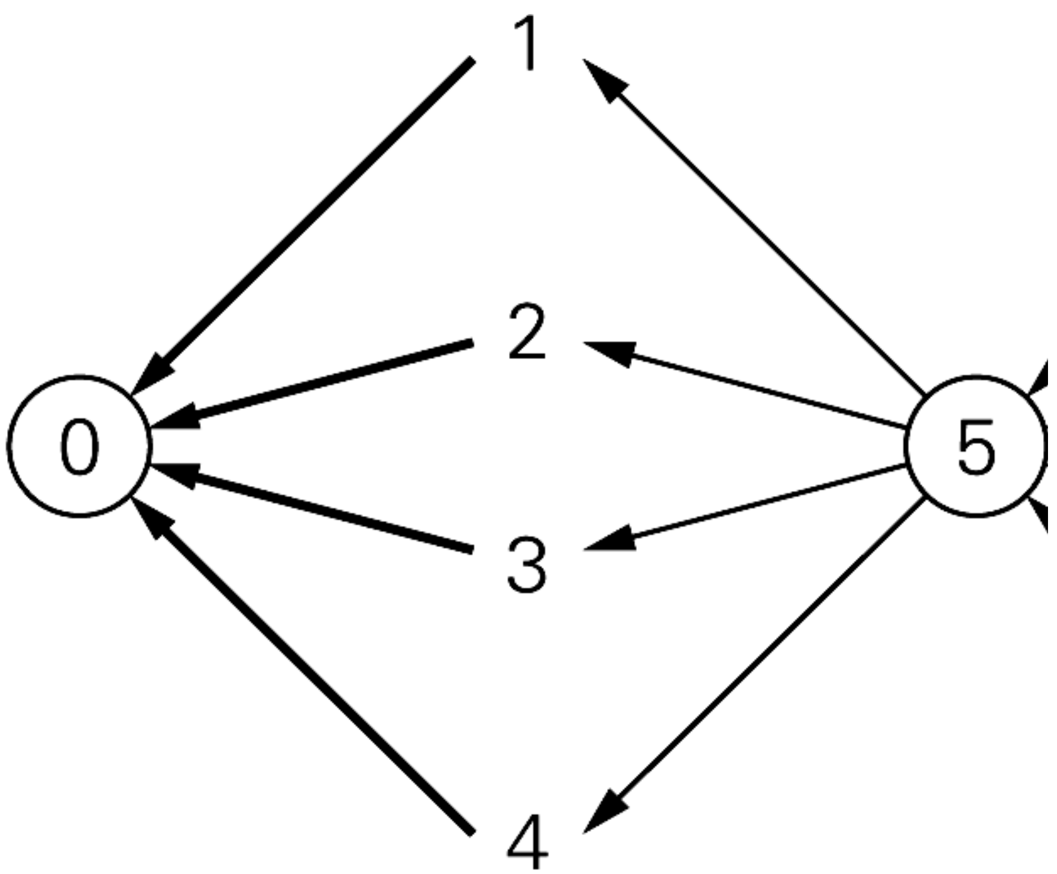
$$m = 4$$



Nim

- 1 pile of n chips
- Players take turns removing $1 \leq k \leq m$ chips
- The player removing the last chip wins

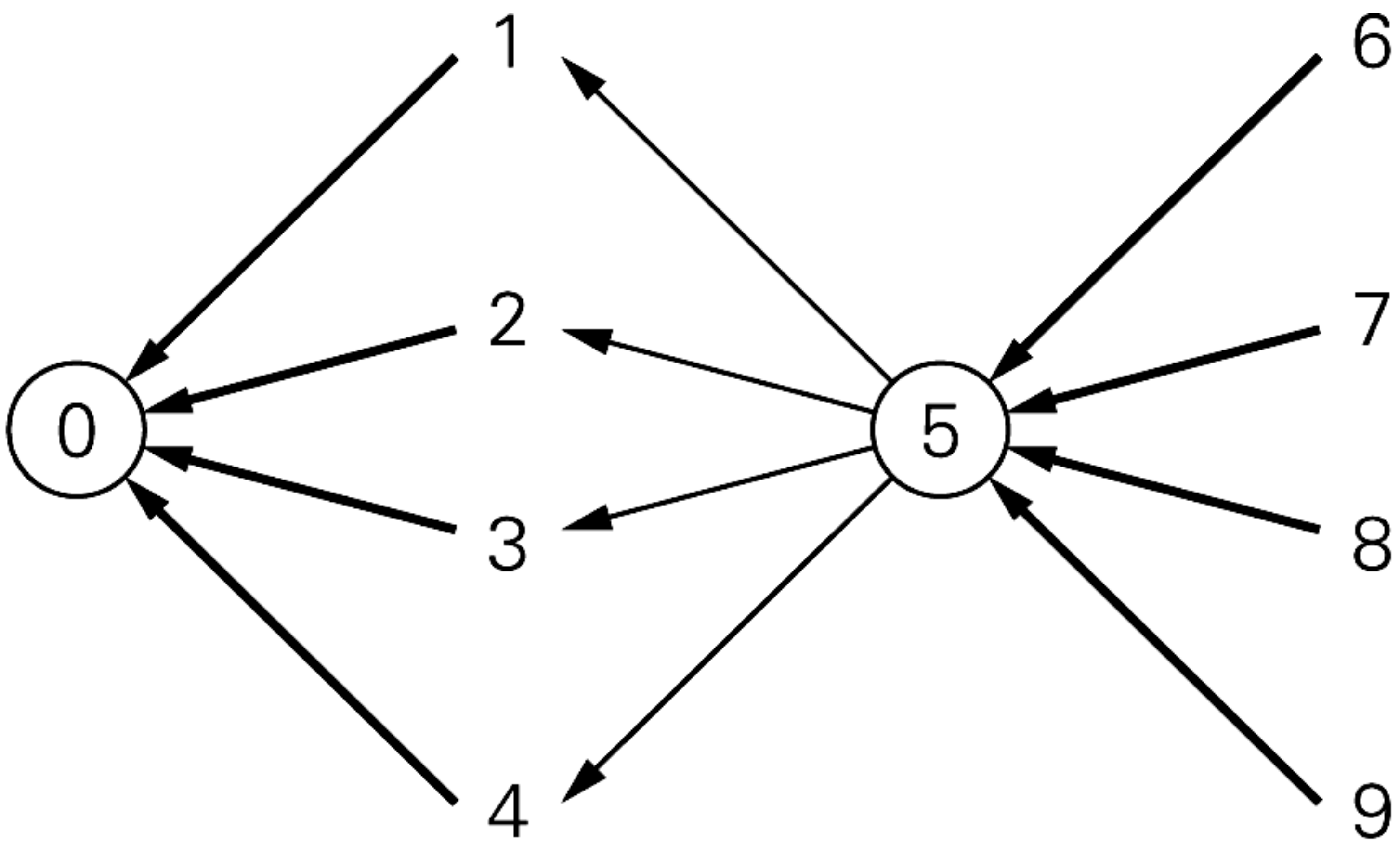
$$m = 4$$



Nim

- 1 pile of n chips
- Players take turns removing $1 \leq k \leq m$ chips
- The player removing the last chip wins

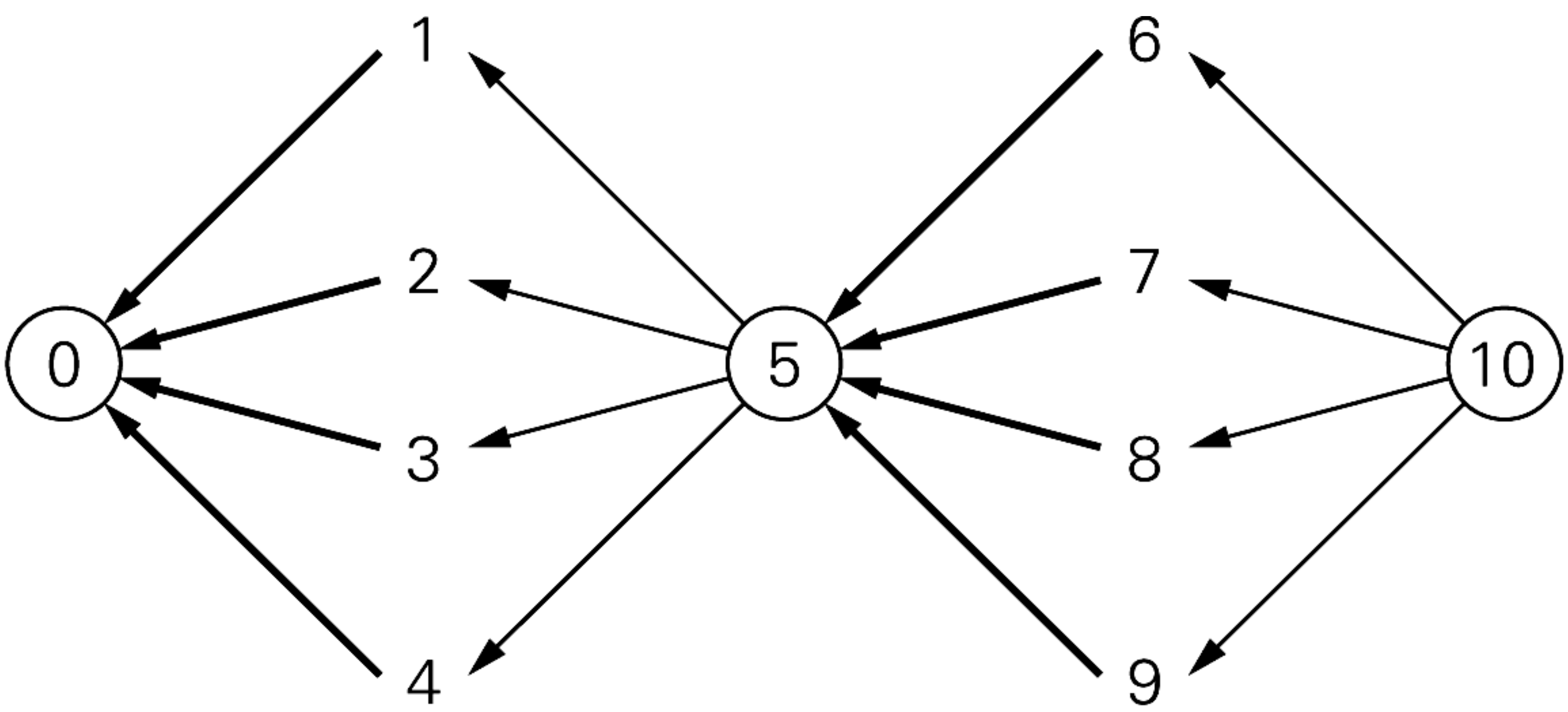
$m = 4$



Nim

- 1 pile of n chips
- Players take turns removing $1 \leq k \leq m$ chips
- The player removing the last chip wins

$m = 4$



Multiplication à la russe

$$n \cdot m = \begin{cases} \frac{n}{2} \cdot 2m & \text{if } n \text{ is even} \\ \frac{n-1}{2} \cdot 2m + m & \text{if } n \text{ is odd} \end{cases}$$

n	m	
50	65	
25	130	
12	260	(+130)
6	520	
3	1, 040	
1	2, 080	(+1040)
	2, 080	+(130 + 1040) = 3, 250

Multiplication à la russe

$$n \cdot m = \begin{cases} \frac{n}{2} \cdot 2m & \text{if } n \text{ is even} \\ \frac{n-1}{2} \cdot 2m + m & \text{if } n \text{ is odd} \end{cases}$$

n	m	
50	65	
25	130	
12	260	(+130)
6	520	
3	1, 040	
1	2, 080	(+1040)
	2, 080	+(130 + 1040) = 3, 250

n	m	
50	65	
25	130	130
12	260	
6	520	
3	1, 040	1, 040
1	2, 080	2, 080
		3, 250

You try it!

- multiply 37×67

n	m	
37	67	

You try it!

- multiply 37 × 67

n	m
37	67
18	

You try it!

- multiply 37 × 67

n	m
37	67
18	134

You try it!

- multiply 37 × 67

n	m	
37	67	
18	134	+ 67