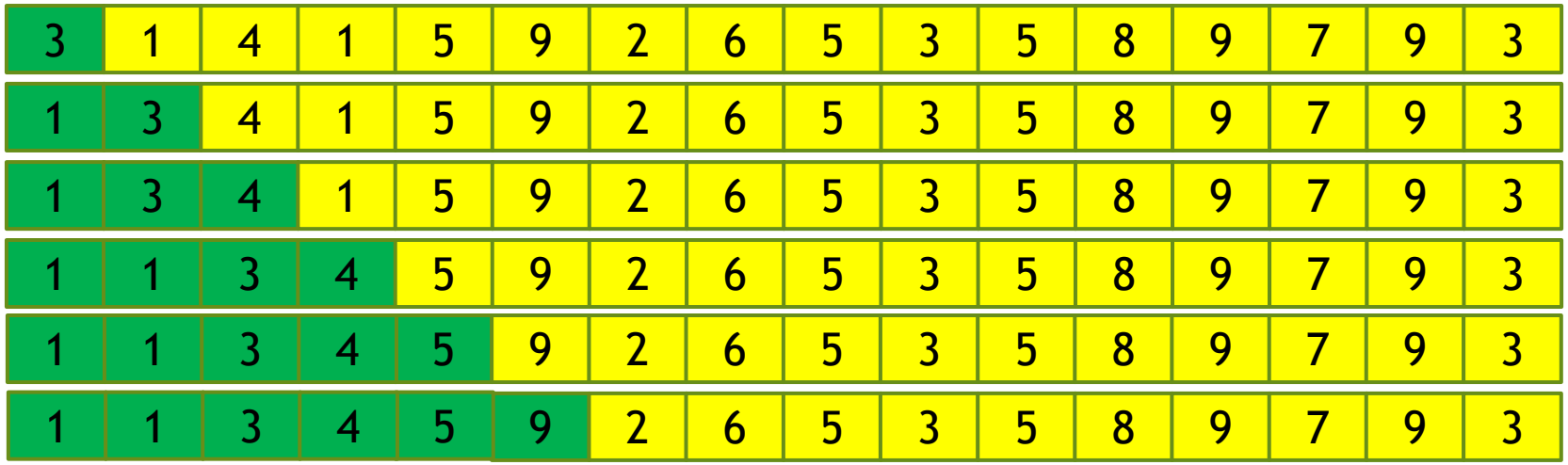
Insertion sort

- Algeng einföld stöðug röðunaraðferð
- Röðunaraðferð byggð á samanburðum
- Alls ekki meðal hraðvirkustu aðferða
  - Góð til að raða stuttum runum
- Stundum notuð sem hluti flóknari röðunaraðferða
- Allir ættu að kunna þessa aðferð

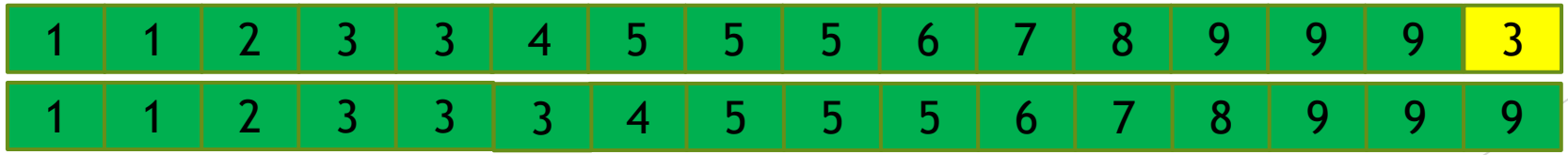
Insertion sort dæmi

- Röðum rununni

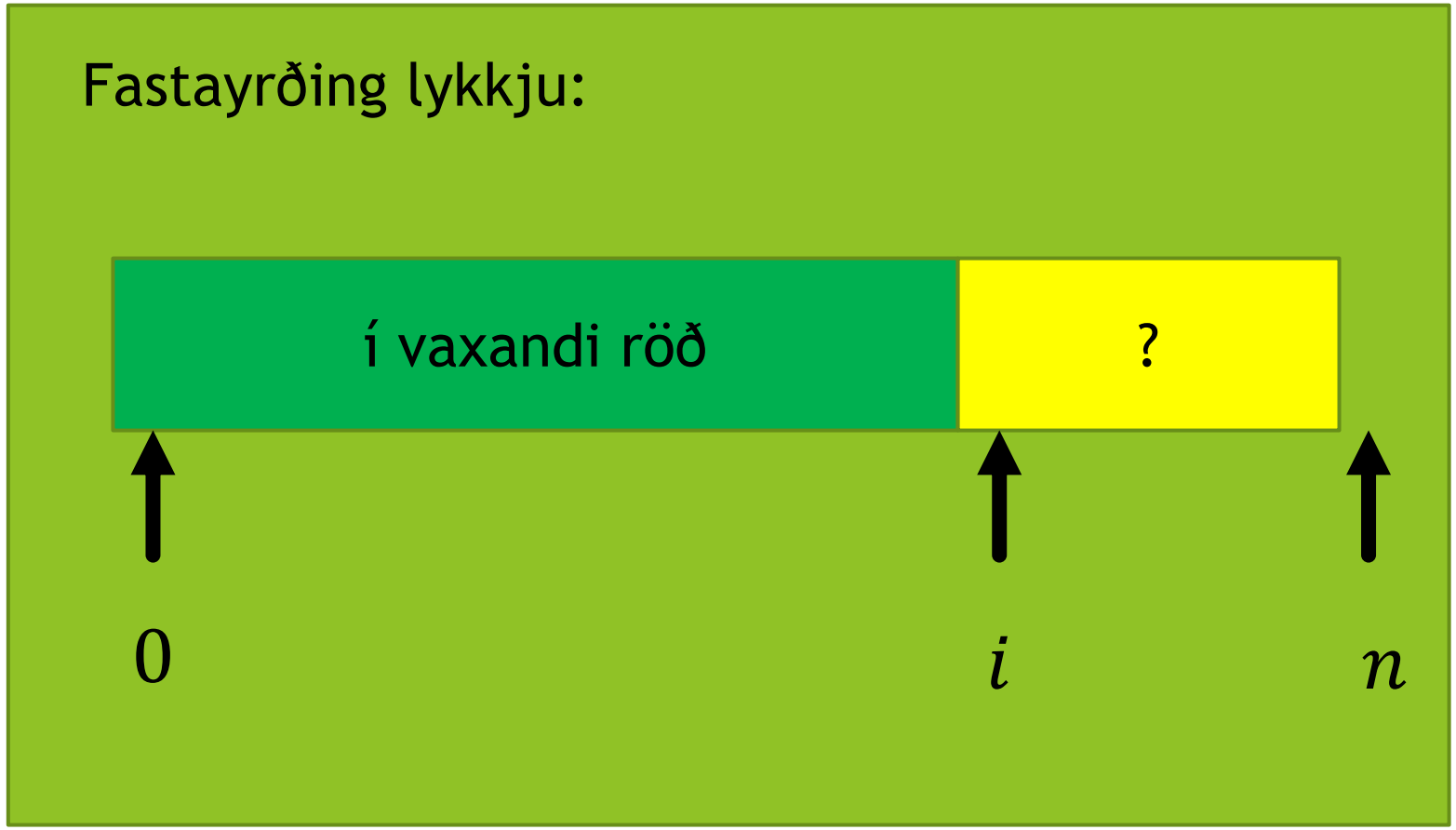
- Ástandið eftir hverja umferð



.....



Grunnhugmynd insertion sort



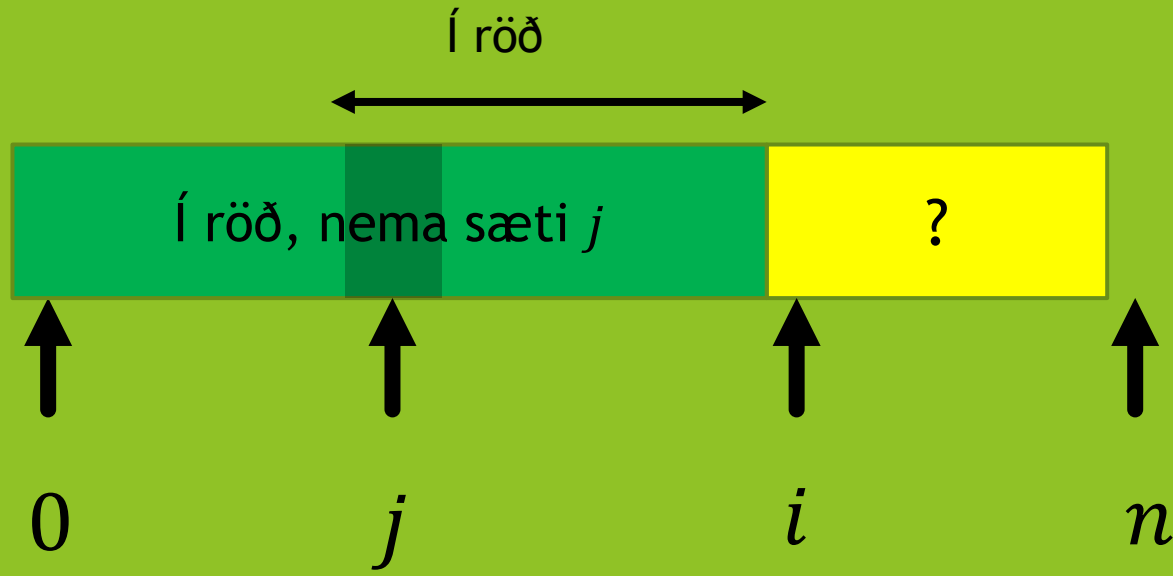
Insertion sort dæmi

- Ástandið fyrir innri lykkju

- Ástandið eftir innri lykkju

Innri lykkja insertion sort

Fastayrðing innri lykkju:



Röðun: Insertion sort

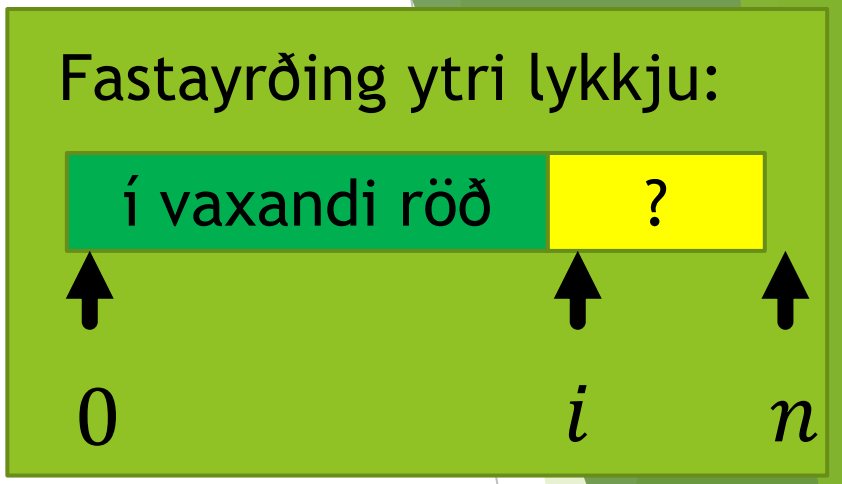
{
Notkun: raða($a_0, a_1, \ldots, a_{n-1}$)
Fyrir: $a_0, a_1, \ldots, a_{n-1}$ er runa af rauntölubreytum
Eftir: Gildunum í rununni hefur verið umraðað svo gildin eru í vaxandi röð
}

stef raða($a_0, a_1, \ldots, a_{n-1}$: runa af rauntölubreytum)
    ???

Röðun: Insertion sort

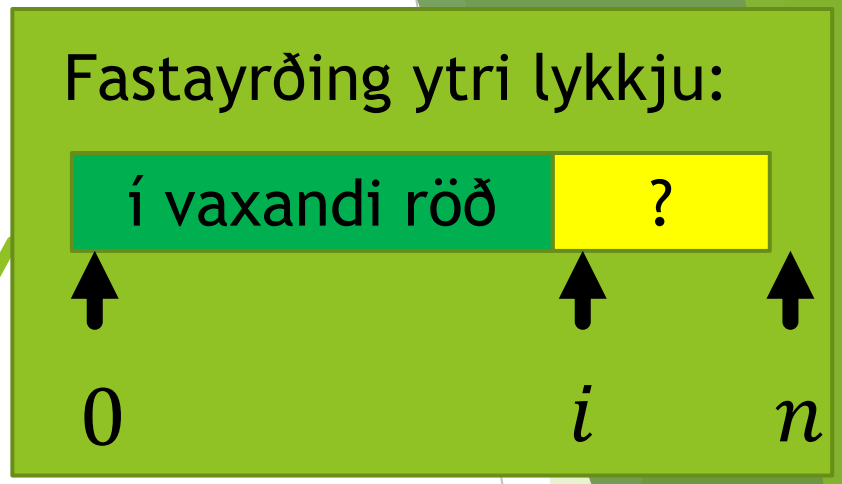
{
Notkun: raða($a_0, a_1, \ldots, a_{n-1}$)
Fyrir: $a_0, a_1, \ldots, a_{n-1}$ er runa af rauntölubreytum
Eftir: Gildunum í rununni hefur verið umraðað svo gildin eru í vaxandi röð
}

stef raða($a_0, a_1, \ldots, a_{n-1}$: runa af rauntölubreytum)
???



Röðun: Insertion sort

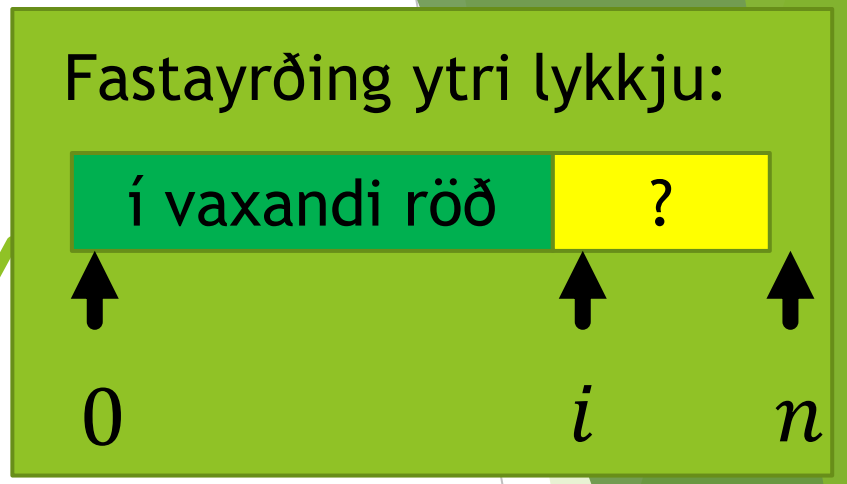
```
{
Notkun:     raða( a_0, a_1, …, a_{n−1} )
Fyrir:      a_0, a_1, …, a_{n−1} er runa af rauntölubreytum
Eftir:      Gildunum í rununni hefur verið umraðað
            svo gildin eru í vaxandi röð
}
stef raða( a_0, a_1, …, a_{n−1}: runa af rauntölubreytum )
    ???
    meðan ???
        { a_0, a_1, …, a_{i−1} er í vaxandi röð, 0 ≤ i ≤ n }
        ???
```



Röðun: Insertion sort

{
Notkun: raða($a_0, a_1, \ldots, a_{n-1}$)
Fyrir: $a_0, a_1, \ldots, a_{n-1}$ er runa af rauntölubreytum
Eftir: Gildunum í rununni hefur verið umraðað svo gildin eru í vaxandi röð
}

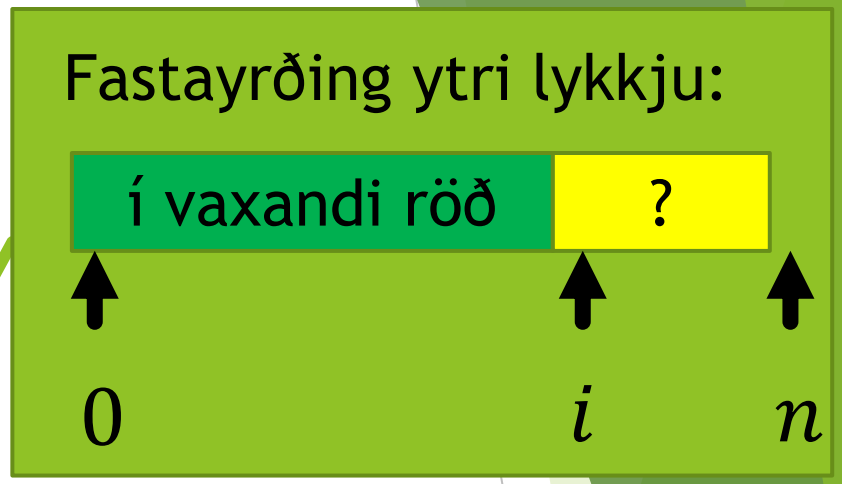
stef raða($a_0, a_1, \ldots, a_{n-1}$: runa af rauntölubreytum)
    ???
    meðan $i \neq n$
        { $a_0, a_1, \ldots, a_{i-1}$ er í vaxandi röð, $0 \leq i \leq n$ }
        ???



Röðun: Insertion sort

{
Notkun: raða($a_0, a_1, \ldots, a_{n-1}$)
Fyrir: $a_0, a_1, \ldots, a_{n-1}$ er runa af rauntölubreytum
Eftir: Gildunum í rununni hefur verið umraðað svo gildin eru í vaxandi röð
}

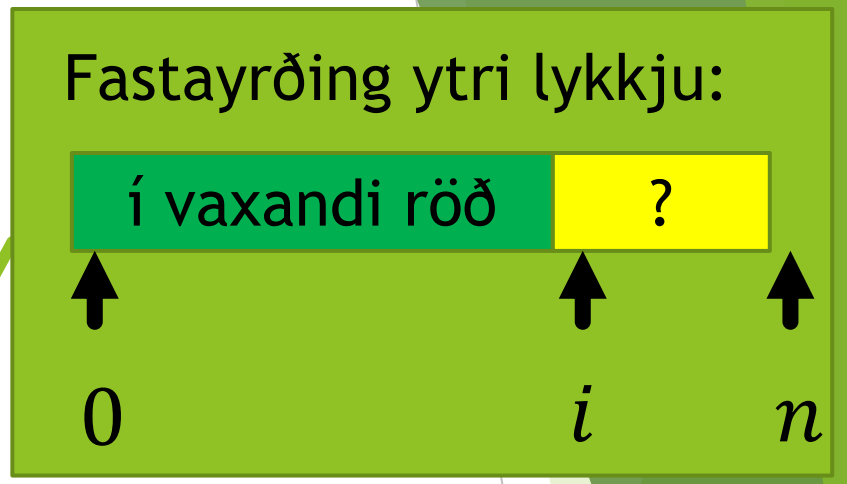
stef raða($a_0, a_1, \ldots, a_{n-1}$: runa af rauntölubreytum)
    i := 0
    meðan $i \neq n$
        { $a_0, a_1, \ldots, a_{i-1}$ er í vaxandi röð, $0 \leq i \leq n$ }
        ???

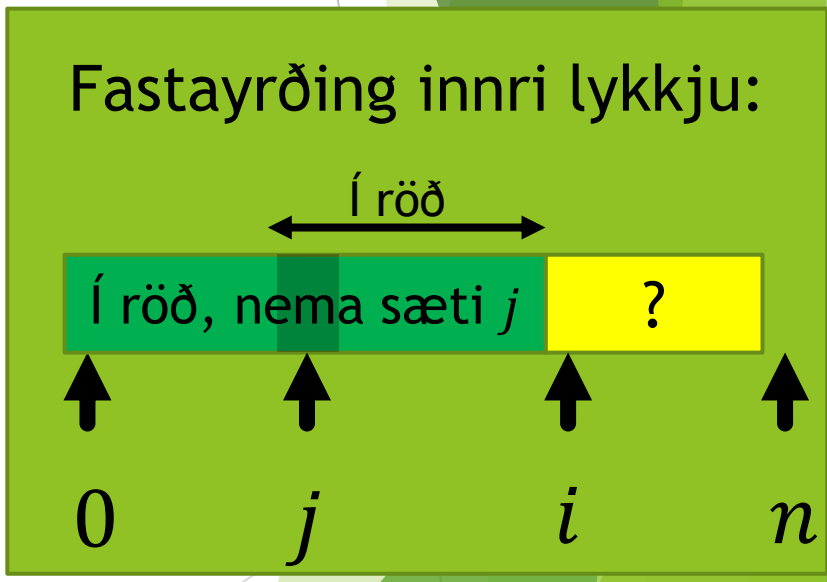


Röðun: Insertion sort

{
Notkun: raða($a_0, a_1, \ldots, a_{n-1}$)
Fyrir: $a_0, a_1, \ldots, a_{n-1}$ er runa af rauntölubreytum
Eftir: Gildunum í rununni hefur verið umraðað svo gildin eru í vaxandi röð
}

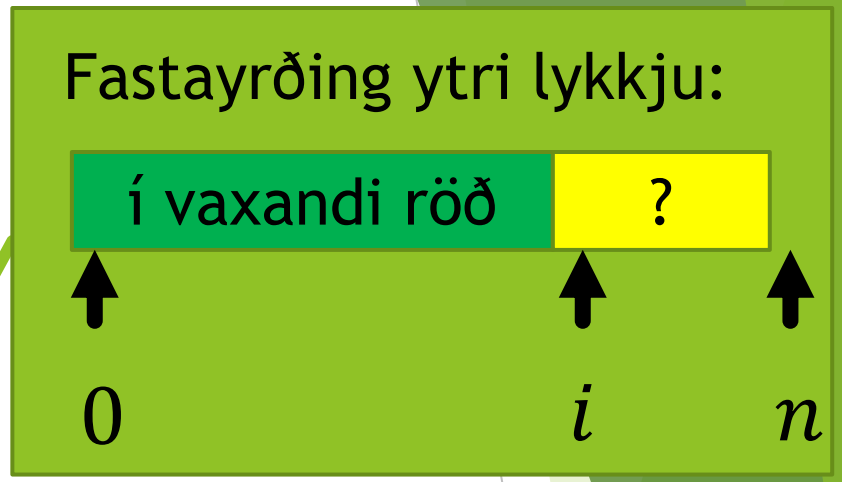
stef raða($a_0, a_1, \ldots, a_{n-1}$: runa af rauntölubreytum)
    i := 0
    meðan $i \neq n$
        { $a_0, a_1, \ldots, a_{i-1}$ er í vaxandi röð, $0 \leq i \leq n$ }
        ???

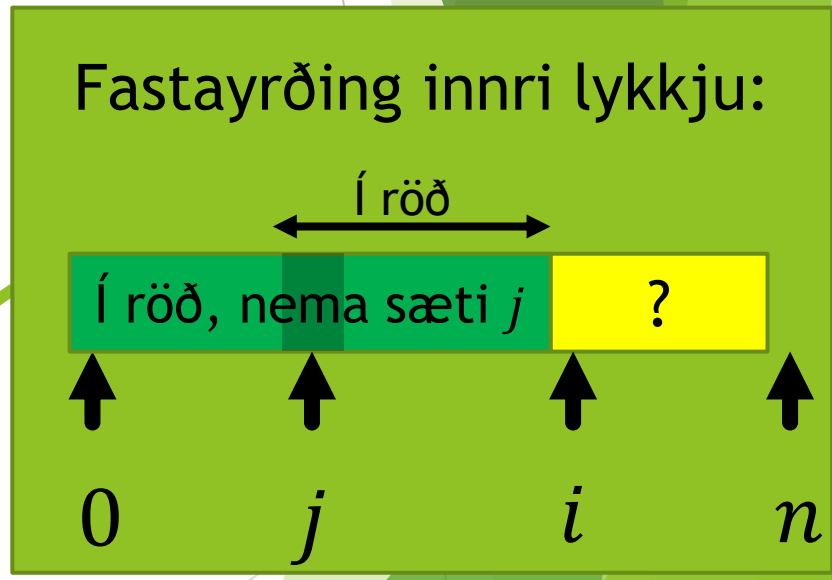




Röðun: Insertion sort

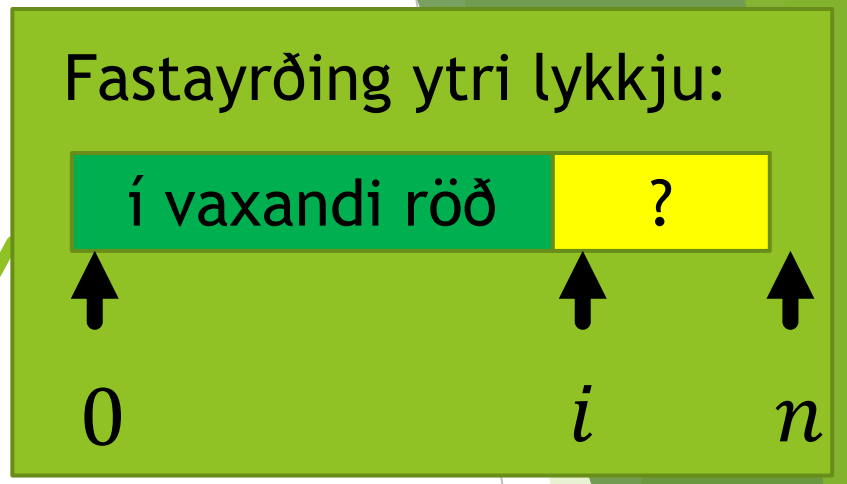
```
{
Notkun:     raða( a_0, a_1, …, a_{n−1} )
Fyrir:      a_0, a_1, …, a_{n−1} er runa af rauntölubreytum
Eftir:      Gildunum í rununni hefur verið umraðað
            svo gildin eru í vaxandi röð
}
stef raða( a_0, a_1, …, a_{n−1}: runa af rauntölubreytum )
    i := 0
    meðan i ≠ n
        { a_0, a_1, …, a_{i−1} er í vaxandi röð, 0 ≤ i ≤ n }
        ???
        meðan ???
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, …, a_{i−1} er í vaxandi röð, }
            { a_0, a_1, …, a_{j−1}, a_{j+1}, …, a_i eru einnig í vaxandi röð. }
            { Gildið í sæti a_j er því ef til vill of aftarlega. }
            ???
```

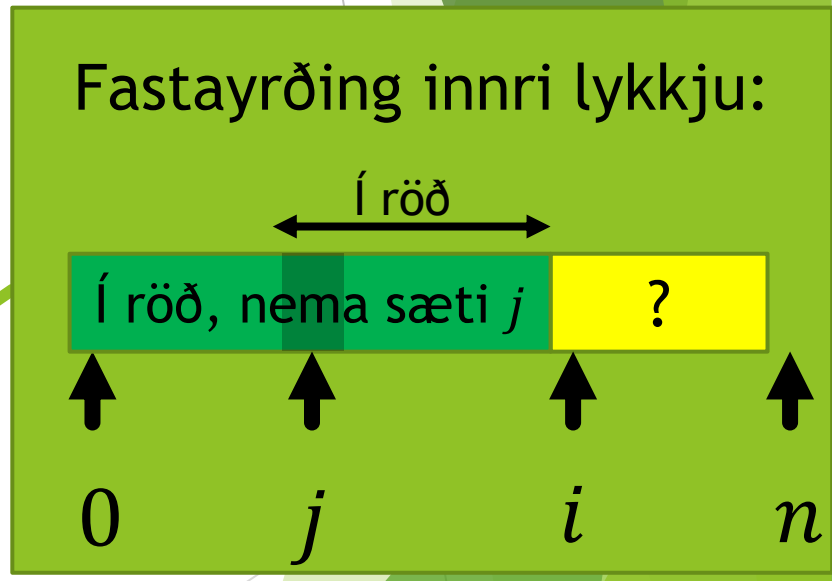




Röðun: Insertion sort

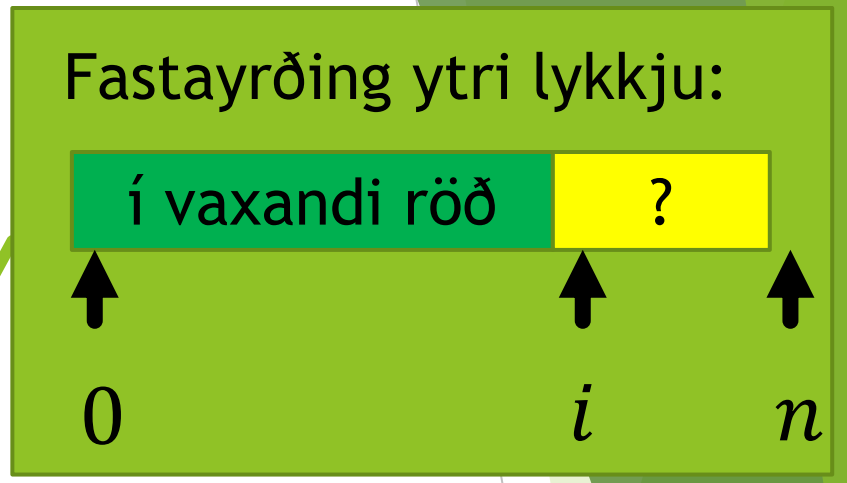
```
{
Notkun:     raða( a_0, a_1, …, a_{n−1} )
Fyrir:      a_0, a_1, …, a_{n−1} er runa af rauntölubreytum
Eftir:      Gildunum í rununni hefur verið umraðað
            svo gildin eru í vaxandi röð
}
stef raða( a_0, a_1, …, a_{n−1}: runa af rauntölubreytum )
    i := 0
    meðan i ≠ n
        { a_0, a_1, …, a_{i−1} er í vaxandi röð, 0 ≤ i ≤ n }
        ???
        meðan j ≠ 0 og a_j < a_{j−1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, …, a_{i−1} er í vaxandi röð, }
            { a_0, a_1, …, a_{j−1}, a_{j+1}, …, a_i eru einnig í vaxandi röð. }
            { Gildið í sæti a_j er því ef til vill of aftarlega. }
            ???
```

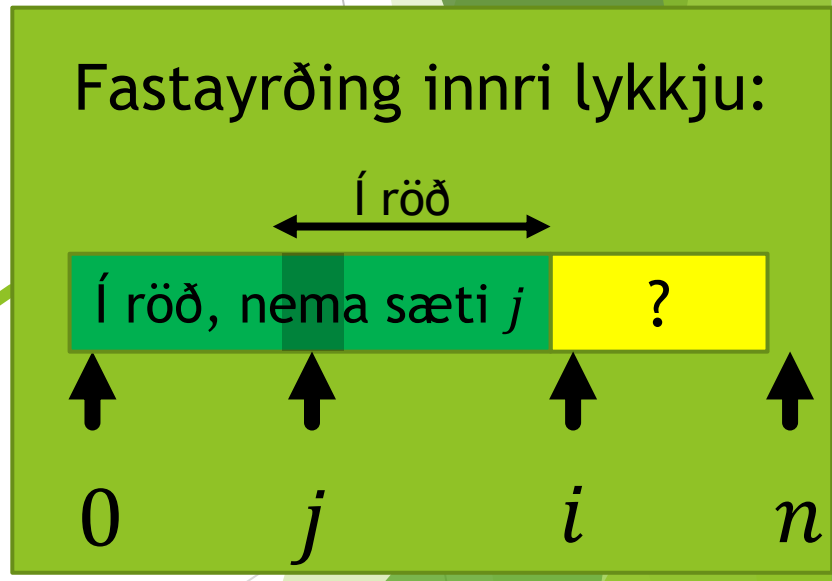




Röðun: Insertion sort

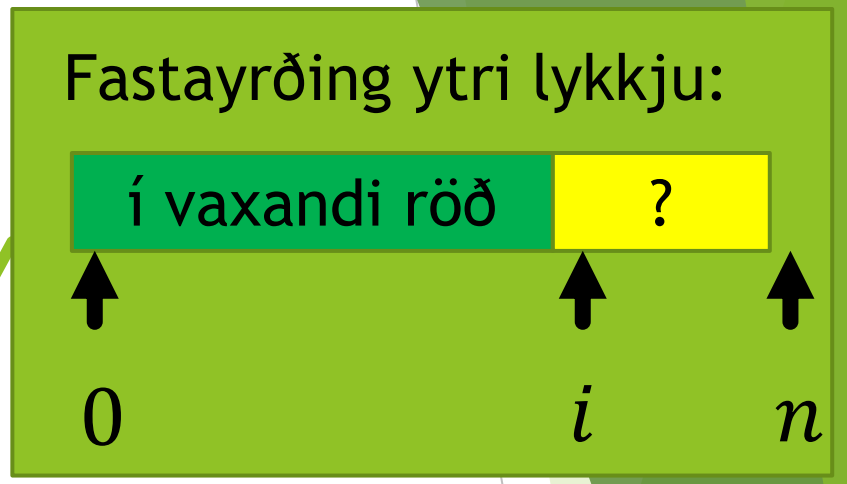
```
{
Notkun:     raða( a_0, a_1, …, a_{n−1} )
Fyrir:      a_0, a_1, …, a_{n−1} er runa af rauntölubreytum
Eftir:      Gildunum í rununni hefur verið umraðað
            svo gildin eru í vaxandi röð
}
stef raða( a_0, a_1, …, a_{n−1}: runa af rauntölubreytum )
    i := 0
    meðan i ≠ n
        { a_0, a_1, …, a_{i−1} er í vaxandi röð, 0 ≤ i ≤ n }
        j := i; i := i + 1
        meðan j ≠ 0 og a_j < a_{j−1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, …, a_{i−1} er í vaxandi röð, }
            { a_0, a_1, …, a_{j−1}, a_{j+1}, …, a_i eru einnig í vaxandi röð. }
            { Gildið í sæti a_j er því ef til vill of aftarlega. }
            ???
```

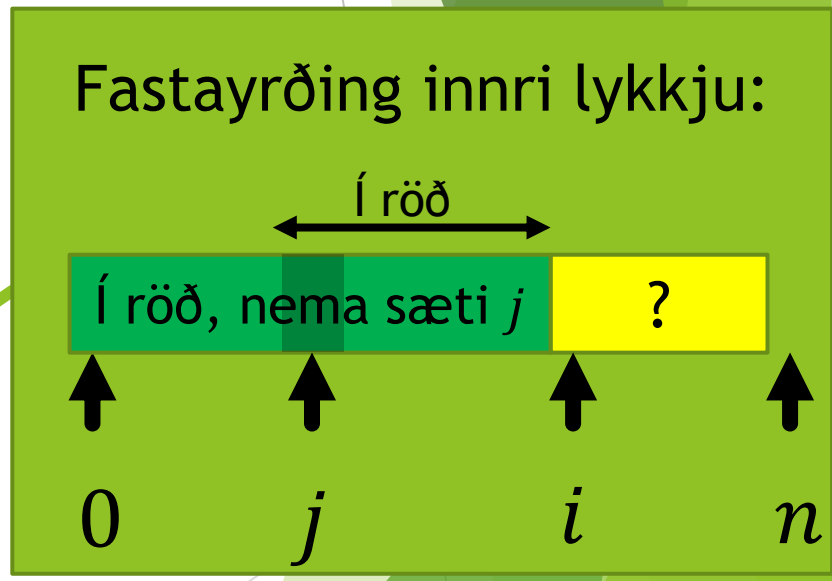




Röðun: Insertion sort

```
{
Notkun:     raða( a_0, a_1, …, a_{n−1} )
Fyrir:      a_0, a_1, …, a_{n−1} er runa af rauntölubreytum
Eftir:      Gildunum í rununni hefur verið umraðað
            svo gildin eru í vaxandi röð
}
stef raða( a_0, a_1, …, a_{n−1}: runa af rauntölubreytum )
    i := 0
    meðan i ≠ n
        { a_0, a_1, …, a_{i−1} er í vaxandi röð, 0 ≤ i ≤ n }
        j := i; i := i + 1
        meðan j ≠ 0 og a_j < a_{j−1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, …, a_{i−1} er í vaxandi röð, }
            { a_0, a_1, …, a_{j−1}, a_{j+1}, …, a_i eru einnig í vaxandi röð. }
            { Gildið í sæti a_j er því ef til vill of aftarlega. }
            m := a_j;   a_j := a_{j−1};   a_{j−1} := m;   j := j − 1
```

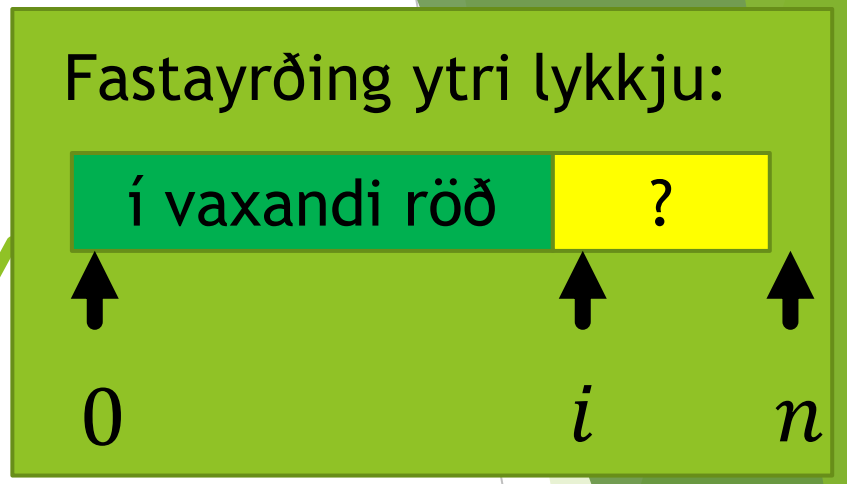


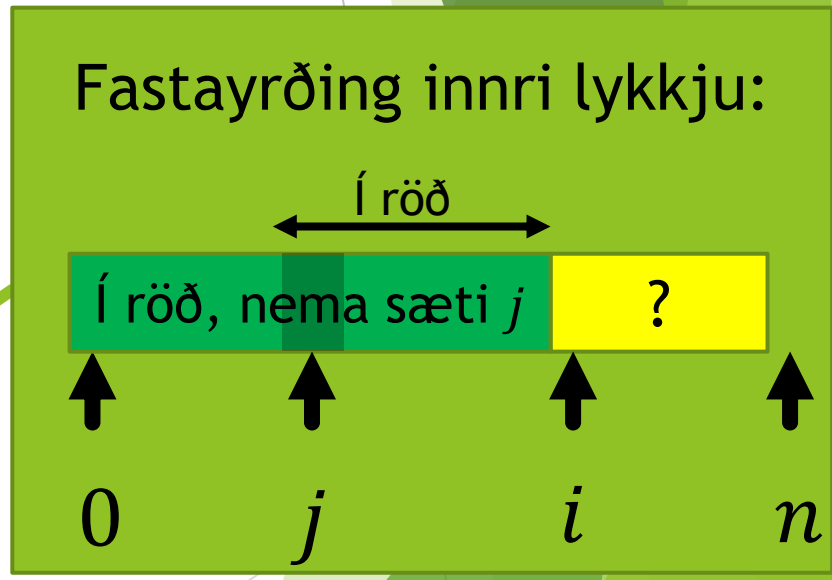


Víxlum gildunum í a_j og a_{j-1}
Færir gildið framar í rununni

Röðun: Insertion sort

```
{
Notkun:     raða( a_0, a_1, …, a_{n−1} )
Fyrir:      a_0, a_1, …, a_{n−1} er runa af rauntölubreytum
Eftir:      Gildunum í rununni hefur verið umraðað
            svo gildin eru í vaxandi röð
}
stef raða( a_0, a_1, …, a_{n−1}: runa af rauntölubreytum )
    i := 0
    meðan i ≠ n
        { a_0, a_1, …, a_{i−1} er í vaxandi röð, 0 ≤ i ≤ n }
        j := i; i := i + 1
        meðan j ≠ 0 og a_j < a_{j−1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, …, a_{i−1} er í vaxandi röð, }
            { a_0, a_1, …, a_{j−1}, a_{j+1}, …, a_i eru einnig í vaxandi röð. }
            { Gildið í sæti a_j er því ef til vill of aftarlega. }
            m := a_j;   a_j := a_{j−1};   a_{j−1} := m;   j := j − 1
```





```
method InsertionSort( m: multiset<int> ) returns ( r: seq<int> )
    ensures multiset(r) == m;
    ensures forall p,q | 0 <= p < q < |r| :: r[p] <= r[q];
{
    r := [];
    var rest := m;
    while rest != multiset{}
        decreases rest;
        invariant m == multiset(r)+rest;
        invariant forall p,q | 0 <= p < q < |r| :: r[p] <= r[q];
    {
        var x :| x in rest;
        rest := rest - multiset{x};
        var p := |r|;
        while p!=0 && r[p-1] > x
            invariant 0 <= p <= |r|;
            decreases p;
            invariant forall s | p <= s < |r| :: r[s] > x;
        {
            p := p-1;
        }
        assert forall s | 0 <= s < p :: r[s] <= x;
        assert r == r[..p]+r[p..];
        r := r[..p]+[x]+r[p..];
    }
}
```

Nauðsynlegar vísbendingar fyrir Dafny

Insertion sort

- Common simple stable sorting method
- Sorting based on comparisons
- Not at all among the fastest methods
  - Good for sorting short sequences
- Sometimes used as part of more complex sorting methods
- Everyone should know this method

Insertion sort example

- Sort the sequence

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3

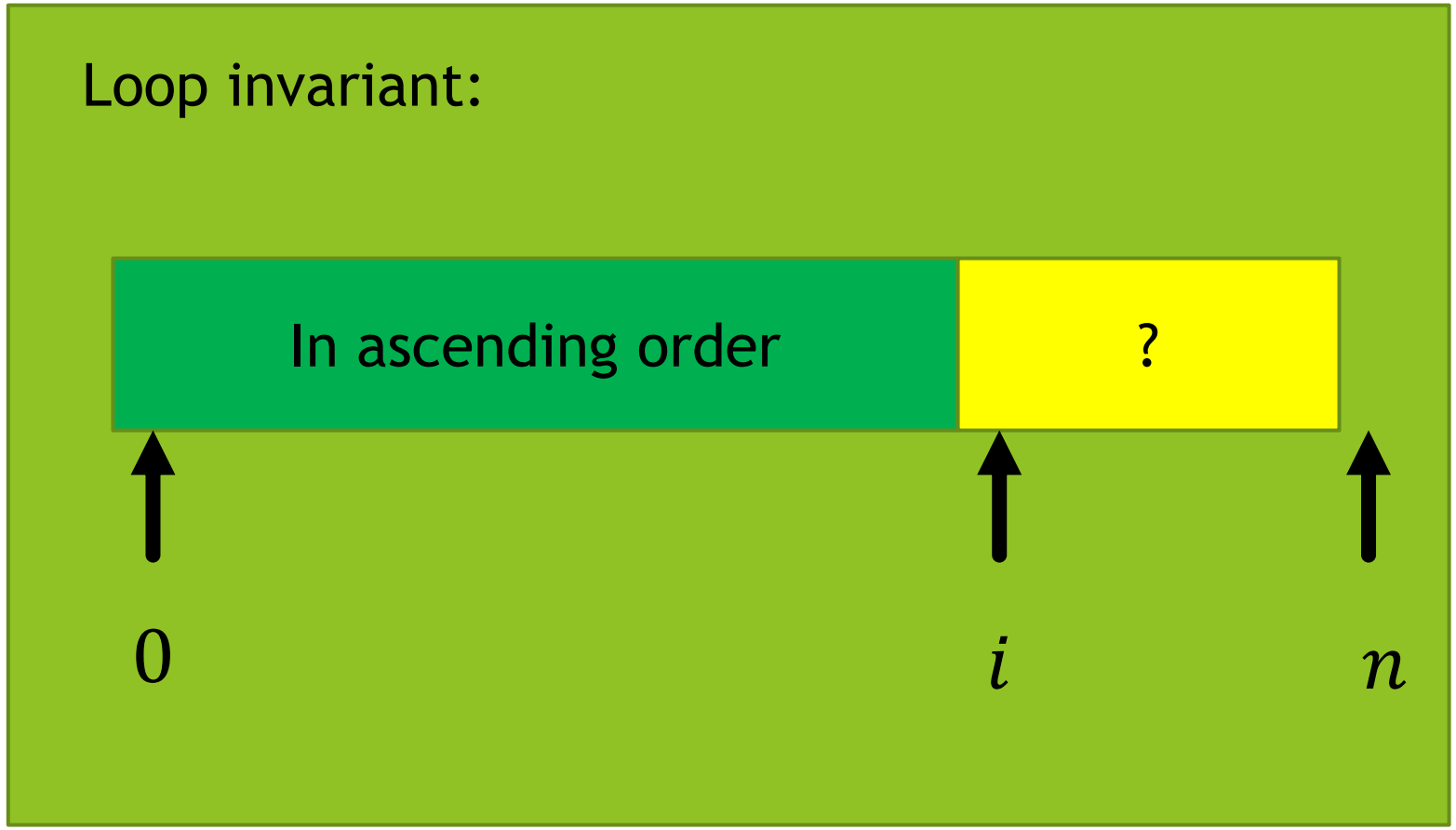
- The state after each pass through loop

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	1	3	4	5	9	2	6	5	3	5	8	9	7	9	3
1	1	3	4	5	9	2	6	5	3	5	8	9	7	9	3
1	1	3	4	5	9	2	6	5	3	5	8	9	7	9	3

.....

1	1	2	3	3	4	5	5	5	6	7	8	9	9	9	3
1	1	2	3	3	3	4	5	5	5	6	7	8	9	9	9

Fundamental idea of insertion sort



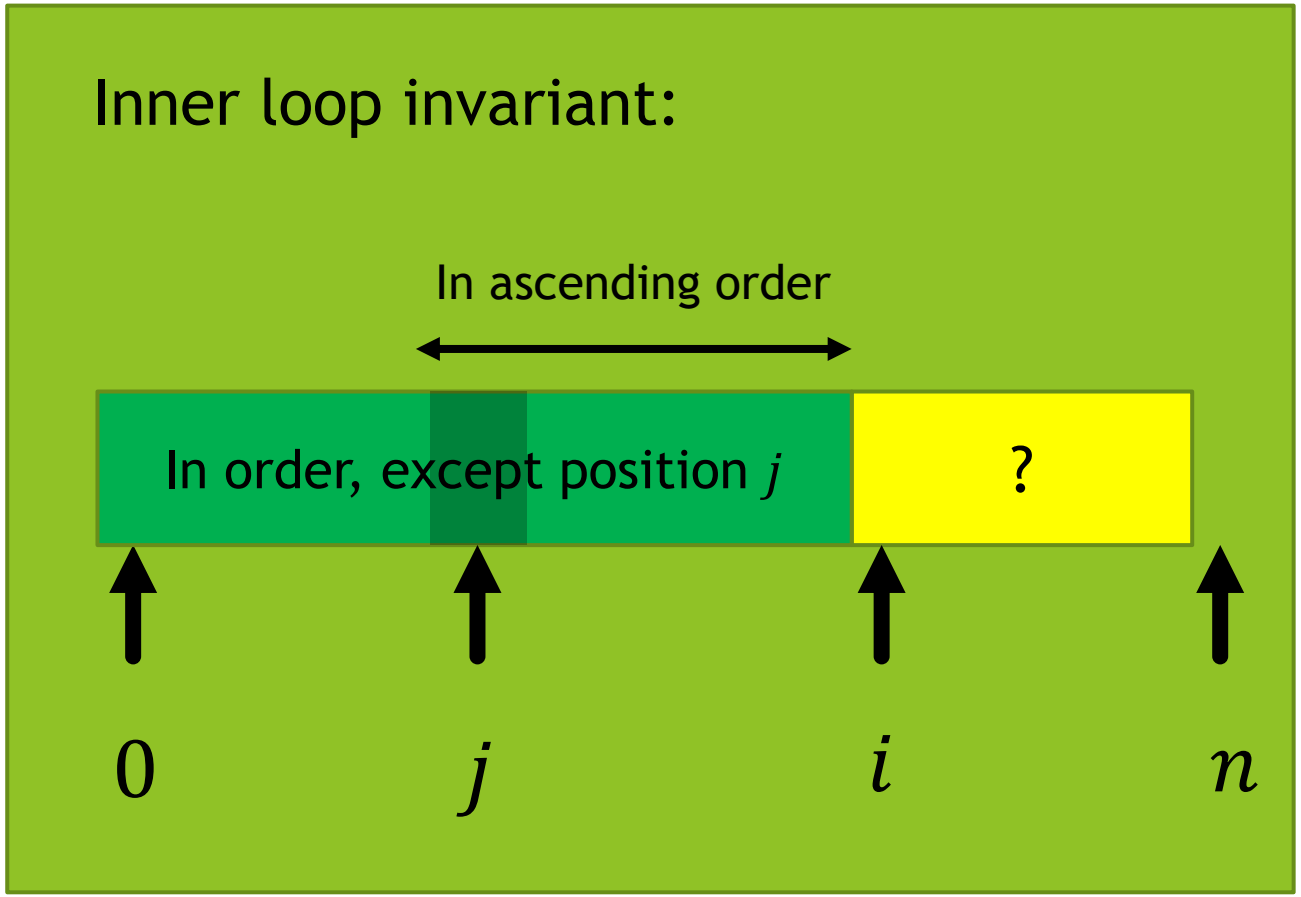
Insertion sort example

- The state before inner loop

- The state before and after each pass through inner loop

- The state after the inner loop

Inner loop of insertion sort



Sorting: Insertion sort

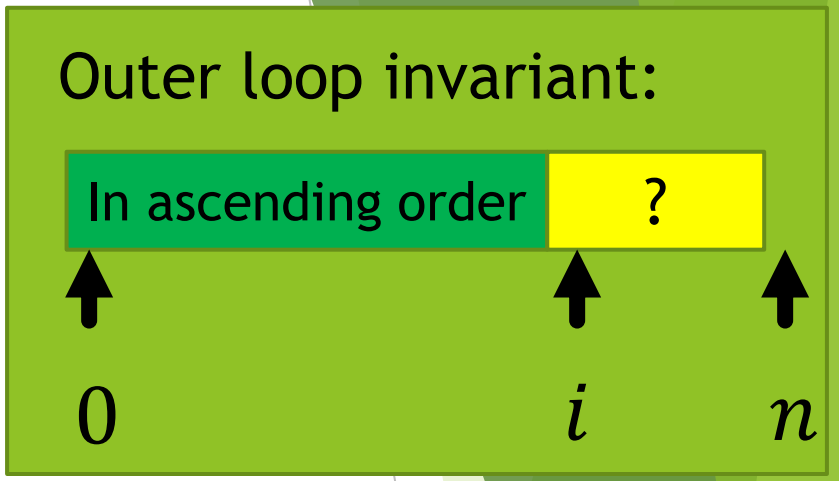
{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)
    ???

Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

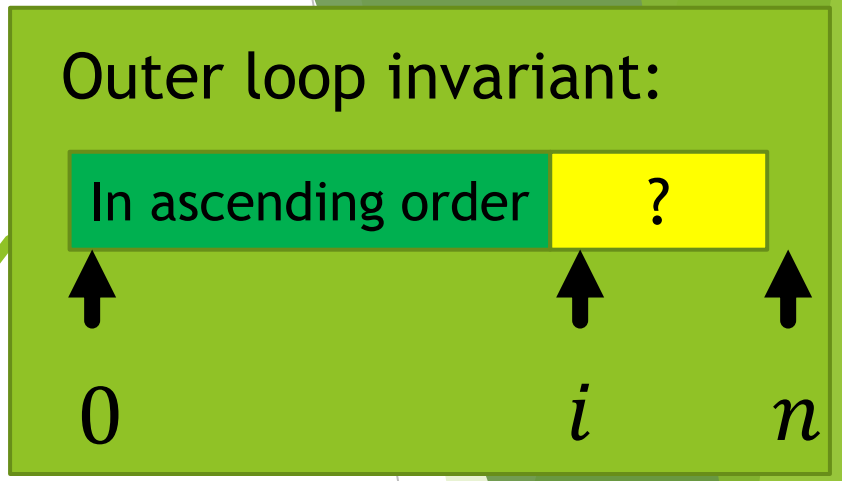
function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)
???



Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

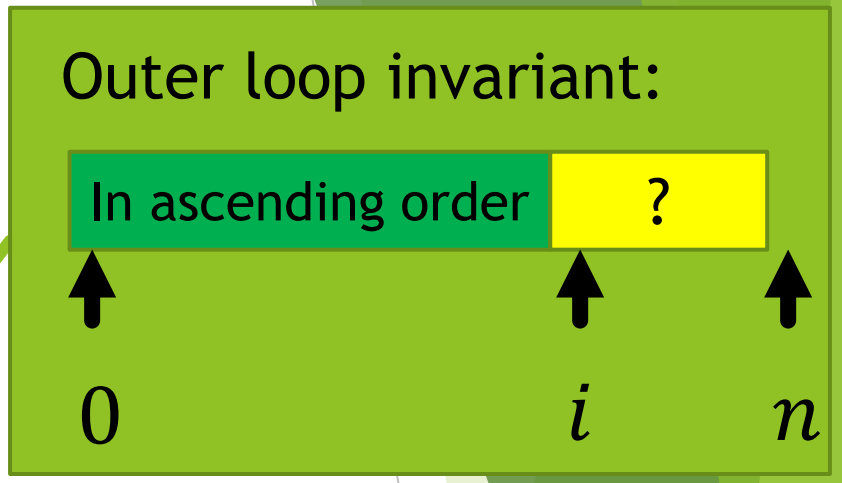
function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)
    ???
    while ???
        { $a_0, a_1, \ldots, a_{i-1}$ are in ascending order, $0 \leq i \leq n$ }
        ???



Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

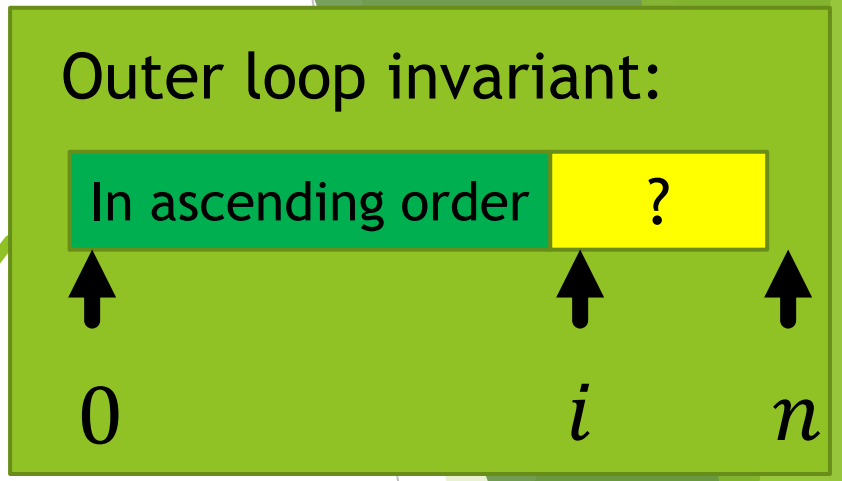
function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)
    i := 0
    while $i \neq n$
        { $a_0, a_1, \ldots, a_{i-1}$ are in ascending order, $0 \leq i \leq n$ }
        ???

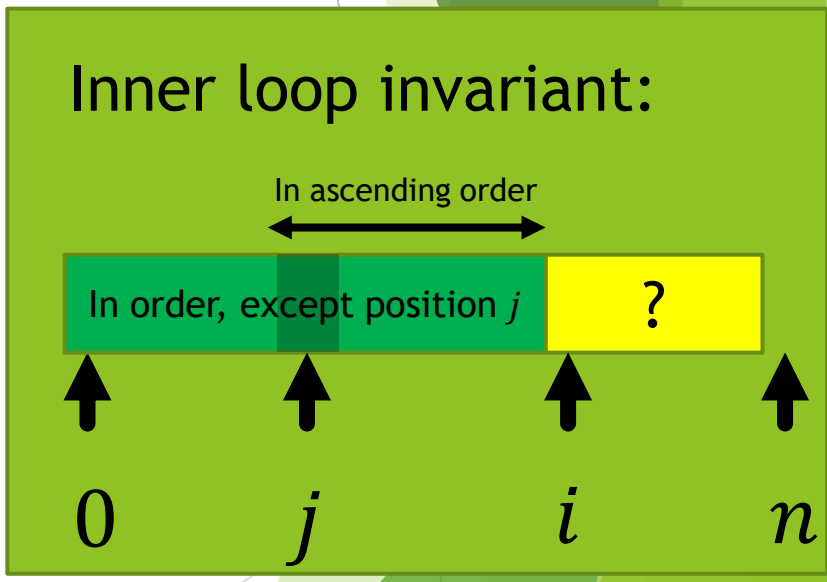


Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)
    i := 0
    while $i \neq n$
        { $a_0, a_1, \ldots, a_{i-1}$ are in ascending order, $0 \leq i \leq n$ }
        ???



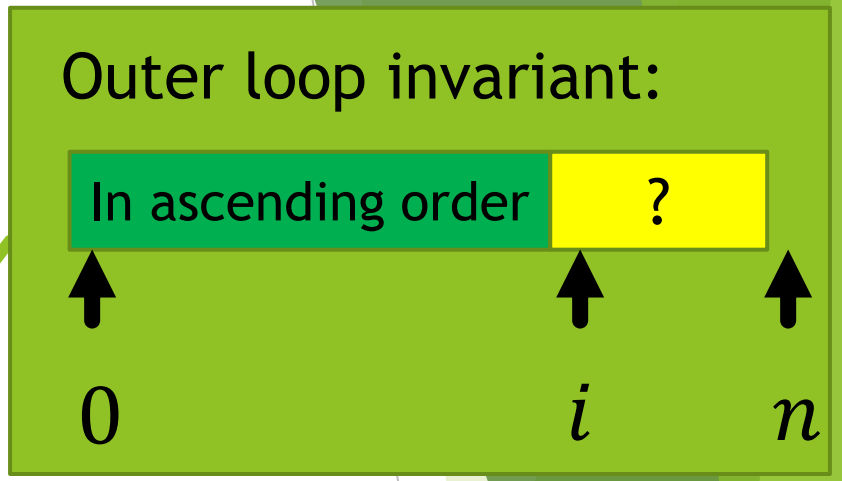


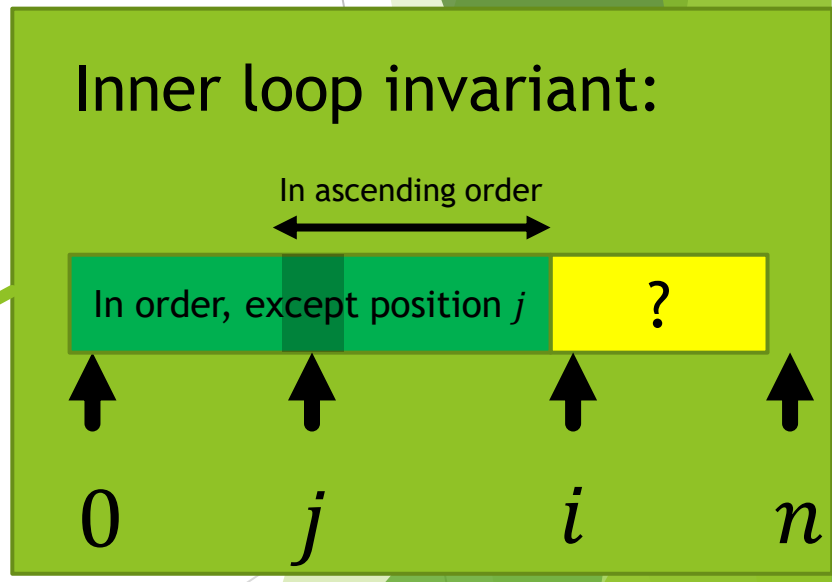
Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)

i := 0
while $i \neq n$
{ $a_0, a_1, \ldots, a_{i-1}$ are in ascending order, $0 \leq i \leq n$ }
???
while ???
{ $0 \leq j < i \leq n$, $a_j, a_{j+1}, \ldots, a_{i-1}$ are ascending, }
{ $a_0, a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_i$ are also ascending. }
{ The value in position a_j is perhaps too far right. }
???

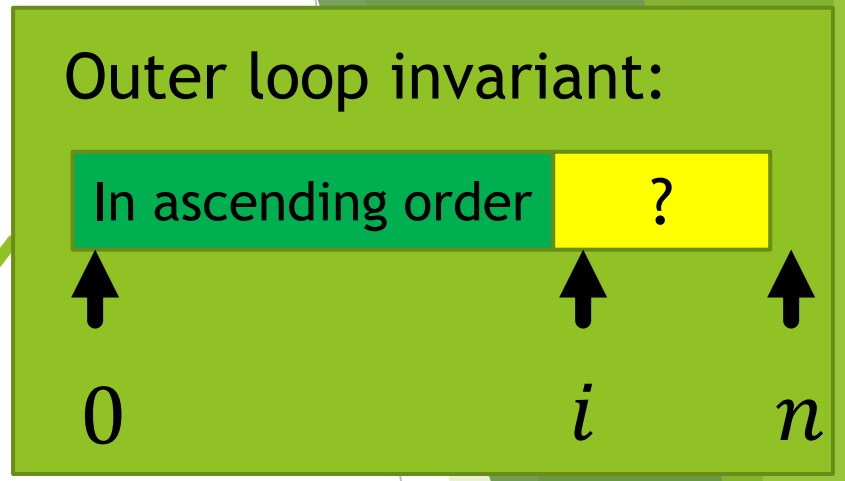


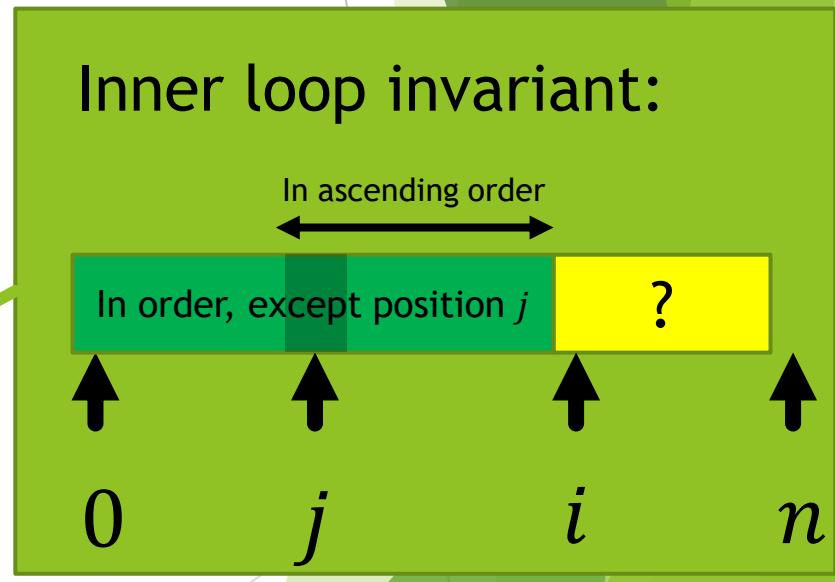


Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

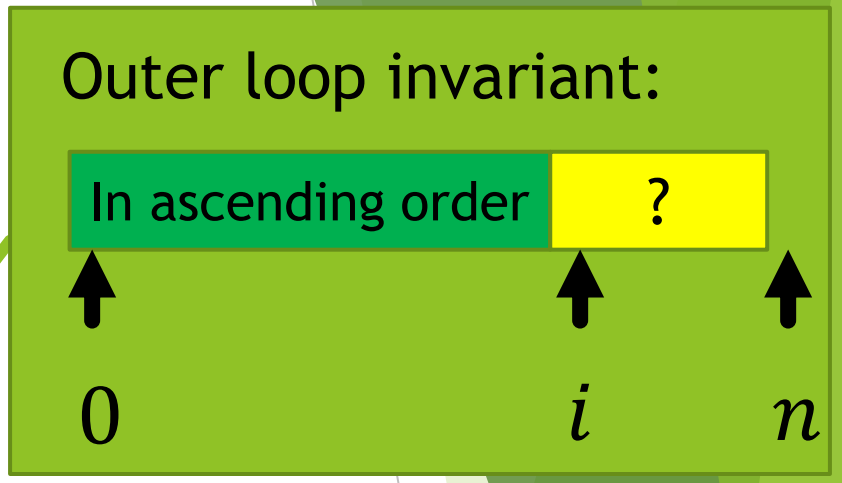
```
function sort( a_0, a_1, ..., a_{n-1}: sequence of numeric variables )
    i := 0
    while i ≠ n
        { a_0, a_1, ..., a_{i-1} are in ascending order, 0 ≤ i ≤ n }
        j := i; i := i + 1
        while j ≠ 0 and a_j < a_{j-1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, ..., a_{i-1} are ascending, }
            { a_0, a_1, ..., a_{j-1}, a_{j+1}, ..., a_i are also ascending. }
            { The value in position a_j is perhaps too far right. }
            ???
```

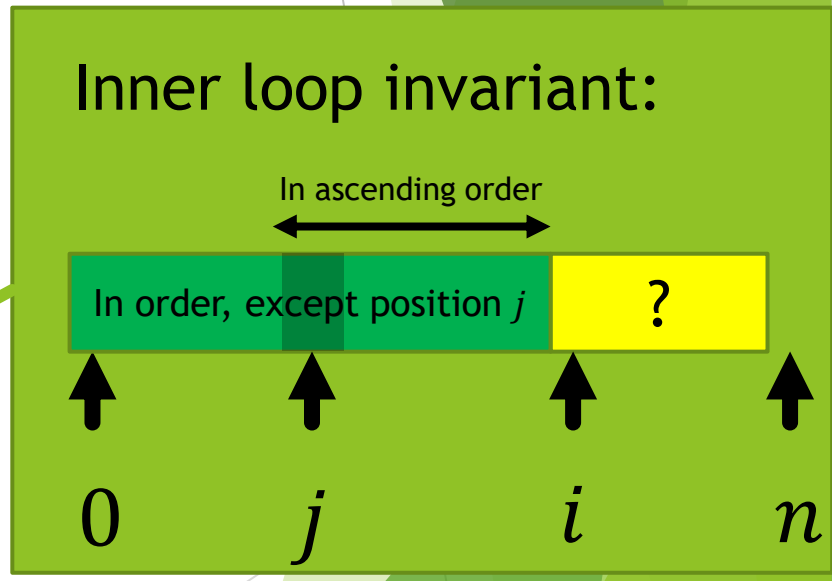




Sorting: Insertion sort

```
{
Usage:      sort( a_0, a_1, ..., a_{n-1} )
Pre:        a_0, a_1, ..., a_{n-1} is a sequence of numeric variables
Post:       The values in the sequence have been permuted
            such that the values are in ascending order
}
function sort( a_0, a_1, ..., a_{n-1}: sequence of numeric variables )
    i := 0
    while i ≠ n
        { a_0, a_1, ..., a_{i-1} are in ascending order, 0 ≤ i ≤ n }
        j := i; i := i + 1
        while j ≠ 0 and a_j < a_{j-1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, ..., a_{i-1} are ascending, }
            { a_0, a_1, ..., a_{j-1}, a_{j+1}, ..., a_i are also ascending. }
            { The value in position a_j is perhaps too far right. }
            m := a_j;   a_j := a_{j-1};   a_{j-1} := m;   j := j − 1
```





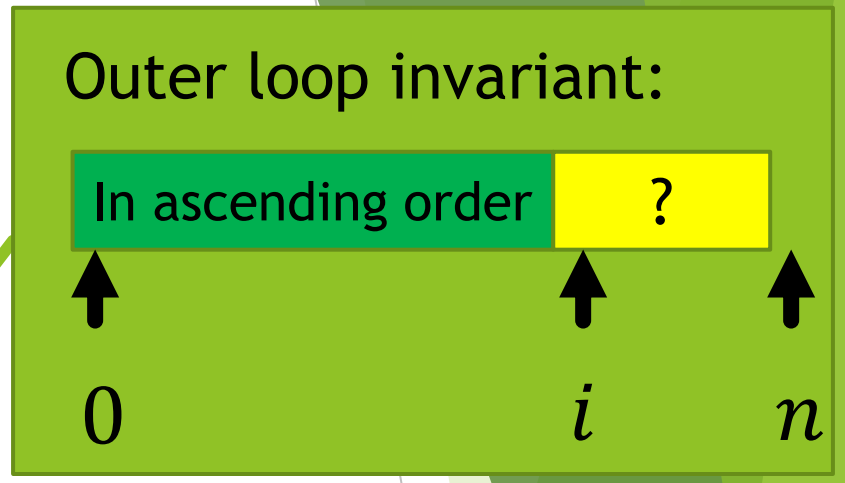
Swap values in a_j and a_{j-1}
Moves the value to the left

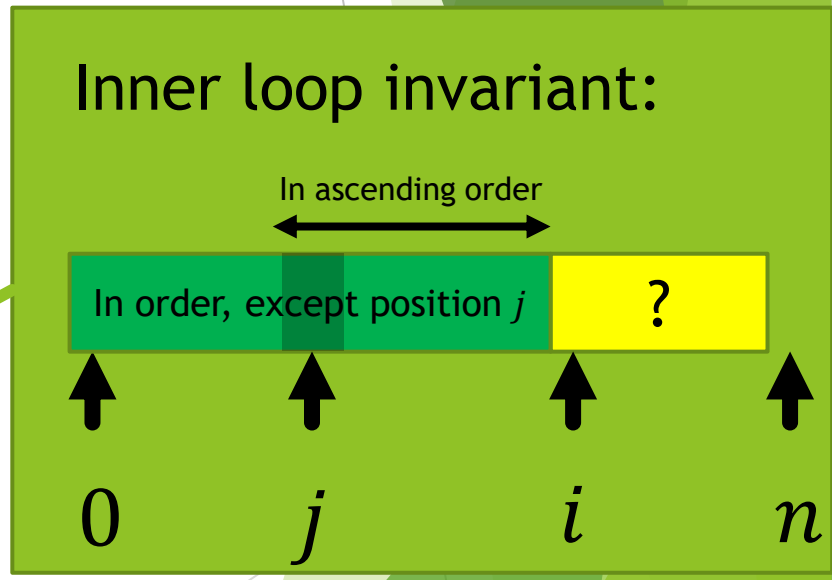
Sorting: Insertion sort

{
Usage: sort($a_0, a_1, \ldots, a_{n-1}$)
Pre: $a_0, a_1, \ldots, a_{n-1}$ is a sequence of numeric variables
Post: The values in the sequence have been permuted such that the values are in ascending order
}

function sort($a_0, a_1, \ldots, a_{n-1}$: sequence of numeric variables)

```
    i := 0
    while i ≠ n
        { a_0, a_1, ..., a_{i-1} are in ascending order, 0 ≤ i ≤ n }
        j := i; i := i + 1
        while j ≠ 0 and a_j < a_{j-1}
            { 0 ≤ j < i ≤ n, a_j, a_{j+1}, ..., a_{i-1} are ascending, }
            { a_0, a_1, ..., a_{j-1}, a_{j+1}, ..., a_i are also ascending. }
            { The value in position a_j is perhaps too far right. }
            m := a_j;   a_j := a_{j-1};   a_{j-1} := m;   j := j − 1
```





```
method InsertionSort( m: multiset<int> ) returns ( r: seq<int> )
    ensures multiset(r) == m;
    ensures forall p,q | 0 <= p < q < |r| :: r[p] <= r[q];
{
    r := [];
    var rest := m;
    while rest != multiset{}
        decreases rest;
        invariant m == multiset(r)+rest;
        invariant forall p,q | 0 <= p < q < |r| :: r[p] <= r[q];
    {
        var x :| x in rest;
        rest := rest - multiset{x};
        var p := |r|;
        while p!=0 && r[p-1] > x
            invariant 0 <= p <= |r|;
            decreases p;
            invariant forall s | p <= s < |r| :: r[s] > x;
        {
            p := p-1;
        }
        assert forall s | 0 <= s < p :: r[s] <= x;
        assert r == r[..p]+r[p..];
        r := r[..p]+[x]+r[p..];
    }
}
```

Necessary hints for Dafny