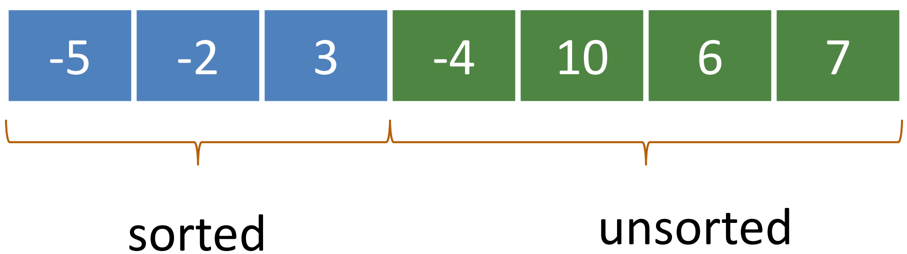
Insertion Sort

Speaker: David Walker
COS 326
Princeton University

Recall Insertion Sort

At any point during the insertion sort:

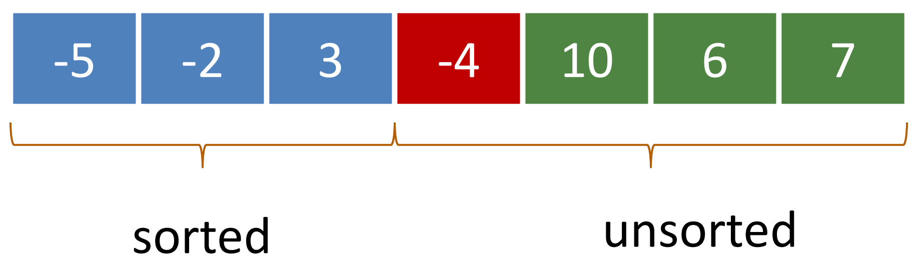
- some initial segment of the array will be sorted
- the rest of the array will be in the same (unsorted) order as it was originally



Recall Insertion Sort

At any point during the insertion sort:

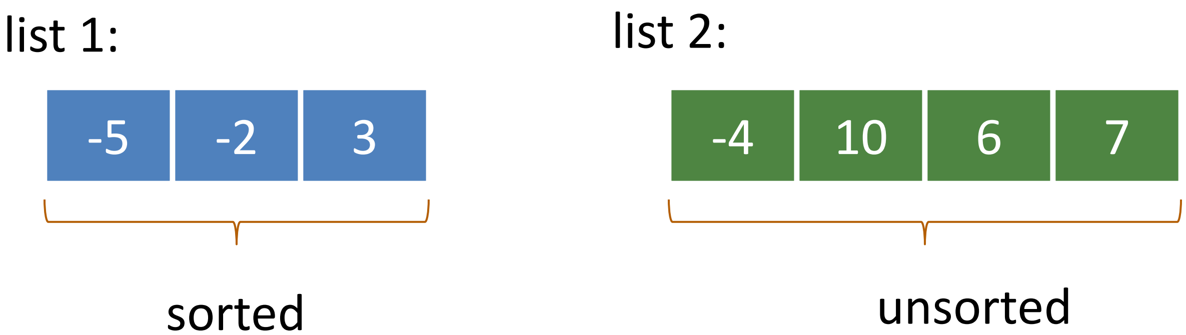
- some initial segment of the array will be sorted
- the rest of the array will be in the same (unsorted) order as it was originally



At each step, take the next item in the array and insert it in order into the sorted portion of the list

Insertion Sort With Lists

The algorithm is similar, except instead of *one array*, we will maintain *two lists*, a sorted list and an unsorted list



We'll factor the algorithm:

- a function to insert into a sorted list
- a sorting function that repeatedly inserts

Insert

```
(* insert x in to sorted list xs *)

let rec insert (x : int) (xs : int list) : int list =
```

Insert

```
(* insert x in to sorted list xs *)

let rec insert (x : int) (xs : int list) : int list =
  match xs with
  | [] ->
  | hd :: tl ->
```

a familiar pattern:
analyze the list by cases

Insert

```
(* insert x in to sorted list xs *)

let rec insert (x : int) (xs : int list) : int list =
  match xs with
  | [] -> [x]
  | hd :: tl ->
```

insert x into the empty list

Insert

```
(* insert x in to sorted list xs *)

let rec insert (x : int) (xs : int list) : int list =
  match xs with
  | [] -> [x]
  | hd :: tl ->
      if hd < x then
        hd :: insert x tl
```

build a new list with:

- hd at the beginning
- the result of inserting x in to the tail of the list afterwards

Insert

```
(* insert x in to sorted list xs *)

let rec insert (x : int) (xs : int list) : int list =
  match xs with
  | [] -> [x]
  | hd :: tl ->
      if hd < x then
        hd :: insert x tl
      else
        x :: xs
```

put x on the front of the list,
the rest of the list follows

A Common Paradigm

Some functions over inductive data do their work like this:

- step 1: set up initial conditions
- step 2: iterate/recurse over the data

A Common Paradigm

Some functions over inductive data do their work like this:

- step 1: set up initial conditions
- step 2: iterate/recurse over the data

How that looks:

```
let f x y =
  let rec loop z =
     … loop z …
  in
  let z = setup x y in
  loop z
```

recursive loop

set up

Insertion Sort

```
type il = int list

insert : int -> il -> il


(* insertion sort *)

let rec insert_sort(xs : il) : il =
```

Insertion Sort

```
type il = int list

insert : int -> il -> il


(* insertion sort *)

let rec insert_sort(xs : il) : il =

  let rec loop (sorted : il) (unsorted : il) : il =



  in
```

Insertion Sort

```
type il = int list

insert : int -> il -> il


(* insertion sort *)

let rec insert_sort(xs : il) : il =

  let rec loop (sorted : il) (unsorted : il) : il =



  in
  loop [] xs
```

Insertion Sort

```
type il = int list

insert : int -> il -> il


(* insertion sort *)

let rec insert_sort(xs : il) : il =

  let rec loop (sorted : il) (unsorted : il) : il =
    match unsorted with
    | [] ->
    | hd :: tl ->
  in
  loop [] xs
```

Insertion Sort

```
type il = int list

insert : int -> il -> il


(* insertion sort *)

let rec insert_sort(xs : il) : il =

  let rec loop (sorted : il) (unsorted : il) : il =
    match unsorted with
    | [] -> sorted
    | hd :: tl ->
  in
  loop [] xs
```

Insertion Sort

```
type il = int list

insert : int -> il -> il



(* insertion sort *)

let rec insert_sort(xs : il) : il =

  let rec loop (sorted : il) (unsorted : il) : il =
    match unsorted with
    | [] -> sorted
    | hd :: tl -> loop (insert hd sorted) tl
  in
  loop [] xs
```

Does Insertion Sort Terminate?

Recall that we said: inductive functions should call themselves recursively on *smaller data items*.

What about that loop in insertion sort?

```
let rec loop (sorted : il) (unsorted : il) : il =
    match unsorted with
    | [] -> sorted
    | hd :: tl -> loop (insert hd sorted) tl
```

Does Insertion Sort Terminate?

Recall that we said: inductive functions should call themselves recursively on *smaller data items*.

What about that loop in insertion sort?

```
let rec loop (sorted : il) (unsorted : il) : il =
    match unsorted with
    | [] -> sorted
    | hd :: tl -> loop (insert hd sorted) tl
```

growing!

shrinking!

Does Insertion Sort Terminate?

Recall that we said: inductive functions should call themselves recursively on *smaller data items*.

What about that loop in insertion sort?

```
let rec loop (sorted : il) (unsorted : il) : il =
    match unsorted with
    | [] -> sorted
    | hd :: tl -> loop (insert hd sorted) tl
```

growing!

shrinking!

Refined idea: Pick an argument up front. That argument must contain smaller data *on every recursive call*.

Exercises

- Write a function to sum the elements of a list
  - sum [1; 2; 3] ==> 6
- Write a function to append two lists
  - append [1;2;3] [4;5;6] ==> [1;2;3;4;5;6]
- Write a function to reverse a list
  - rev [1;2;3] ==> [3;2;1]
- Write a function to turn a list of pairs into a pair of lists
  - split [(1,2); (3,4); (5,6)] ==> ([1;3;5], [2;4;6])
- Write a function that returns all prefixes of a list
  - prefixes [1;2;3] ==> [[]; [1]; [1;2]; [1;2;3]]
- suffixes…