

# Foundations of Computer Science

## Take, drop & search

Dr. Robert Harle & Dr. Jeremy Yallop

2020–2021

**Question 1:** What is the type of this function?

```
In[1]: let rec flatten = function
      | [] -> []
      | l :: ls -> l @ flatten ls

Out[1]: val flatten : 'a list list -> 'a list = <fun>
```

**Question 2a:** What is the cost of evaluating `xs @ ys`?

**Question 2b:** What is the cost of evaluating `x :: xs`?

**Question 1:** What is the type of this function?

```
In[1]: let rec flatten = function
      | [] -> []
      | l :: ls -> l @ flatten ls
```

```
Out[1]: val flatten : 'a list list -> 'a list = <fun>
```

**Question 2a:** What is the cost of evaluating  $xs @ ys$ ?

**Question 2b:** What is the cost of evaluating  $x :: xs$ ?

**Question 1:** What is the type of this function?

```
In[1]: let rec flatten = function
      | [] -> []
      | l :: ls -> l @ flatten ls
```

```
Out[1]: val flatten : 'a list list -> 'a list = <fun>
```

**Question 2a:** What is the cost of evaluating `xs @ ys`?

**Question 2b:** What is the cost of evaluating `x :: xs`?

**Question 1:** What is the type of this function?

```
In[1]: let rec flatten = function
      | [] -> []
      | l :: ls -> l @ flatten ls
```

```
Out[1]: val flatten : 'a list list -> 'a list = <fun>
```

**Question 2a:** What is the cost of evaluating  $xs @ ys$ ?

$O(\text{List.length } xs)$

**Question 2b:** What is the cost of evaluating  $x :: xs$ ?

**Question 1:** What is the type of this function?

```
In[1]: let rec flatten = function
      | [] -> []
      | l :: ls -> l @ flatten ls
```

```
Out[1]: val flatten : 'a list list -> 'a list = <fun>
```

**Question 2a:** What is the cost of evaluating  $xs @ ys$ ?

$$O(\text{List.length } xs)$$

**Question 2b:** What is the cost of evaluating  $x :: xs$ ?

$$O(1)$$

$$xs = \underbrace{[x_0; x_1; \dots; x_{i-1}]}_{\text{take}(xs,i)} \underbrace{[x_i; x_{i+1}; \dots; x_{n-1}]}_{\text{drop}(xs,i)}$$

```
let rec take = function  
| ([], _) -> []  
| (x::xs, i) ->  
    if i > 0 then x :: take (xs, i - 1)  
    else []
```

```
let rec drop = function  
| ([], _) -> []  
| (x::xs, i) ->  
    if i > 0 then drop (xs, i - 1)  
    else x::xs
```

```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```



```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```

```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```

```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```

```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```

```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```

```
val take : 'a list * int -> 'a list = <fun>  
val drop : 'a list * int -> 'a list = <fun>
```

```
In[2]: let a = [1; 2; 3; 4; 5; 6; 7]
```

```
Out[2]: val a : int list = [1; 2; 3; 4; 5; 6; 7]
```

```
In[3]: take (a, 3)
```

```
Out[3]: - : int list = [1; 2; 3]
```

```
In[4]: drop (a, 3)
```

```
Out[4]: - : int list = [4; 5; 6; 7]
```

Find  $x$  in list  $[x_1; x_1; \dots; x_n]$  by **comparing with each element**

Obviously  **$O(n)$  time**

Simple & general

**Ordered searching** needs only  $O(\log n)$

**Indexed lookup** needs only  $O(1)$

More about search in later lectures . . .

```
In[5]: let rec member x = function
      | [] -> false
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

**Equality testing** is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: ~ : bool = true
```

...but **not** for functions

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```



```
In[5]: let rec member x = function  
      | [] -> false  
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

**Equality testing** is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: - : bool = true
```

...but **not** for functions

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```

```
In[5]: let rec member x = function  
      | [] -> false  
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

Equality testing is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: ~ : bool = true
```

...but not for functions

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```

```
In[5]: let rec member x = function
      | [] -> false
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

**Equality testing** is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: ~ : bool = true
```

...but not for functions

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```

```
In[5]: let rec member x = function  
      | [] -> false  
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

**Equality testing** is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: - : bool = true
```

...but not for functions

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```

```
In[5]: let rec member x = function  
      | [] -> false  
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

**Equality testing** is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: - : bool = true
```

...but **not for functions**

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```

```
In[5]: let rec member x = function  
      | [] -> false  
      | y :: l -> x = y || member x l
```

```
Out[5]: val member : 'a -> 'a list -> bool = <fun>
```

**Equality testing** is ok for integers...

```
In[6]: member 3 [2;3;4]
```

```
Out[6]: - : bool = true
```

...but **not for functions**

```
In[7]: member take [take; drop]
```

```
Out: Exception: Invalid_argument "compare: functional value".
```