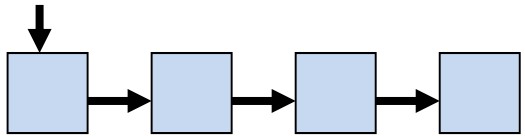
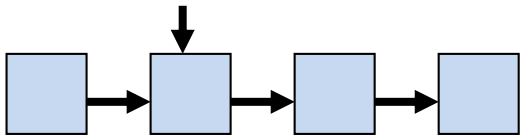
Sequence: recursive view

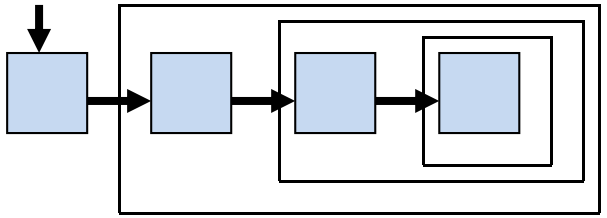
- **Iterative view**



- **pos = 1, 2, 3, 4, ...**
- **first, next(first), next(next(first)), ...**



- **Recursive view**



- **Seq ::= Head Tail | ¤**
  Head ::= element
  Tail ::= Seq
- **the “view” is reflected in the program !!!**

Sequence: recursion

```
int size(listref L) {
return is_empty(L) ? 0 : 1 + size(tail(L));
}
```

```
int size(listref L) {
    if is_empty(L) return 0;
    else return 1 + size(tail(L));
    }
```

Sequence: cons

- **Cons**truct a list (add at the **HEAD**)

```
listref cons(listref e, listref L) {
    return set_tail(e, L);
    }
```

- Set the tail of an element (**e**) to a reference to a list (empty/non-empty)

Sequence: add

```
listref b_add(int v, listref L)
{
    if (is_empty(L))                    return create_e(v);                          1
    else if (v < get_value(head(L)))    return cons(create_e(v), L);                2
    else                                return cons(head(L), b_add(v, tail(L)));    3
    }
```

add at (1) end (tail); (2) beginning (head); (3) middle

cons always adds at the head of the list

Sequence: add

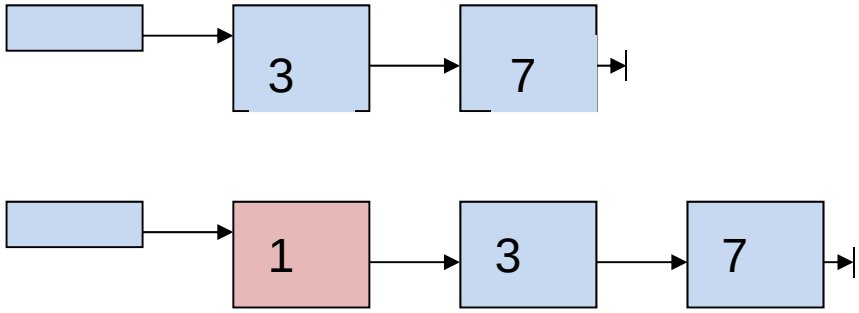
```
listref b_add(int v, listref L)
{
  return is_empty(L)              ? create_e(v)                          1
       : v < get_value(head(L))   ? cons(create_e(v), L)                 2
       :                            cons(head(L), b_add(v, tail(L)));    3
  }
```

add at (1) end (tail); (2) beginning (head); (3) middle

cons always adds at the head of the list

Sequence – add

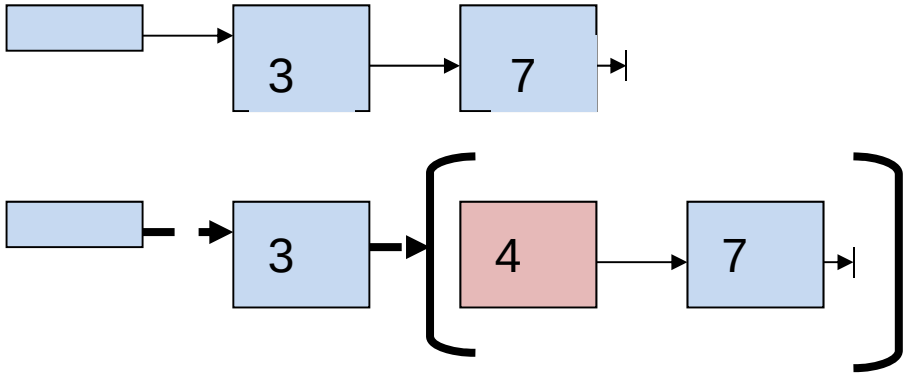
- **add at beginning**



else if (**v** < **get_value**(**head**(**L**))) return **cons**(**create_e**(**v**), **L**); //2

Sequence – add

- **add in middle**



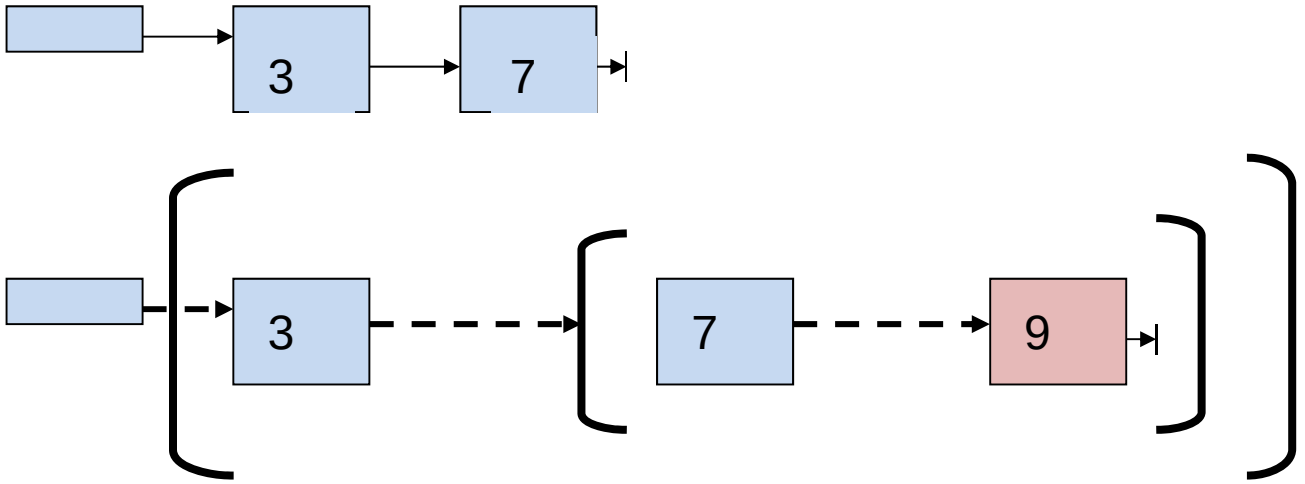
else return **cons**(**head**(**L**), **b_add**(**v**, **tail**(**L**))); //3

…

else if (**v** < **get_value**(**head**(**L**))) return **cons**(**create_e**(**v**), **L**); //2

Sequence – add

- **add at end**




```
else                 return cons(head(L), b_add(v, tail(L)));      //3 …
else                 return cons(head(L), b_add(v, tail(L)));      //3 …
if (is_empty(L))     return create_e(v);                           //1
```