Recursion

Examples

Length (size) of a sequence

code	call 1	call 2	call 3	call 4
int size(list) {	**(a,b,c)**	**(b,c)**	**(c)**	**(¤)**
if is_empty(list) return 0 **else return 1 +** **size(tail(list));** **}**	**false** **return** **1 + 2**	**false** **return** **1 + 1**	**false** **return** **1 + 0**	**true** **return** **0**

time

time

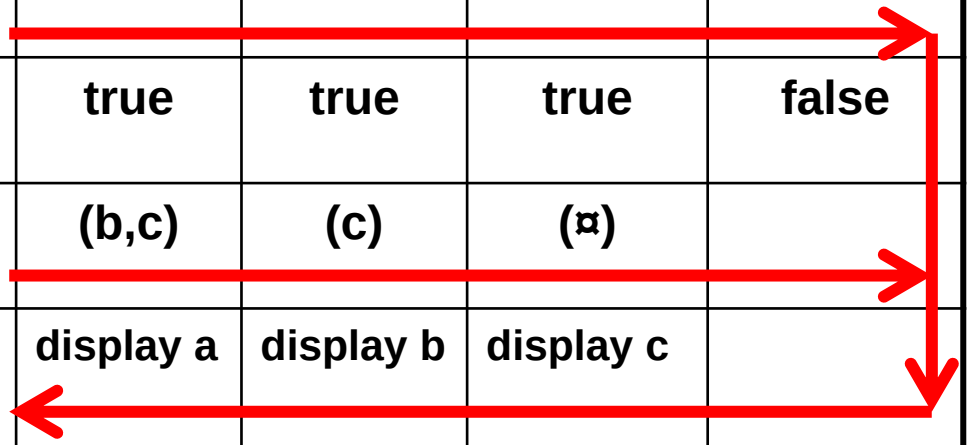
Output 3

Display list (forwards)

code	call 1	call 2	call 3	call 4
display(list) {	**(a,b,c)**	**(b,c)**	**(c)**	**(¤)**
if not is_empty(list) {	**true**	**true**	**true**	**false**
display_el(head(list));	**display a**	**display b**	**display c**	
display(tail(list));	**(b,c)**	**(c)**	**(¤)**	
}	**Output**	**a b c**		
}				

Display list (backwards)

code	call 1	call 2	call 3	call 4
display(list) {	**(a,b,c)**	**(b,c)**	**(c)**	**(¤)**
if not is_empty(list) {	**true**	**true**	**true**	**false**
display(tail(list));	**(b,c)**	**(c)**	**(¤)**	
display_el(head(list));	**display a**	**display b**	**display c**	
}	**Output**	**c b a**		
}				

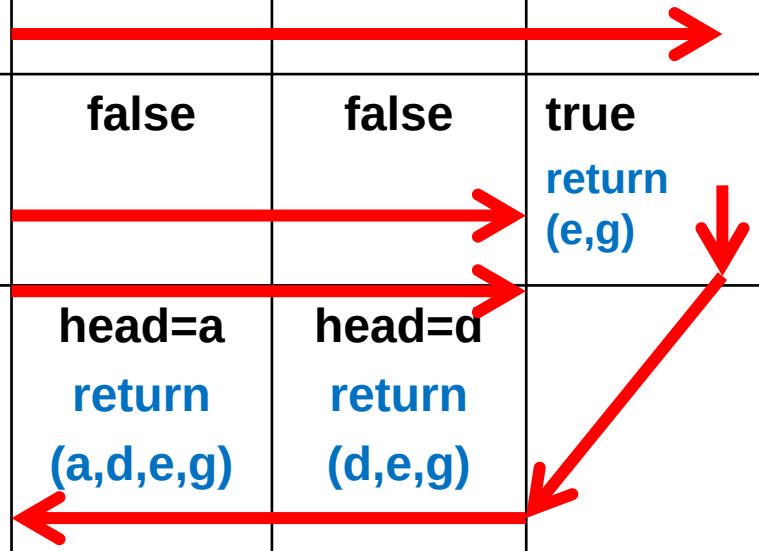


Insert - beginning (sorted sequence)

code	Call 1
insert(el, list) {	**(a, (c,d,g))**
if is_empty(list) return cons(el, list);	**false**
if getval(el)<getval(head(list)) **return cons(el, list);**	**true** **return** **(a,c,d,g)**
return(cons(head(list), **insert(el, tail(list)));**	
}	

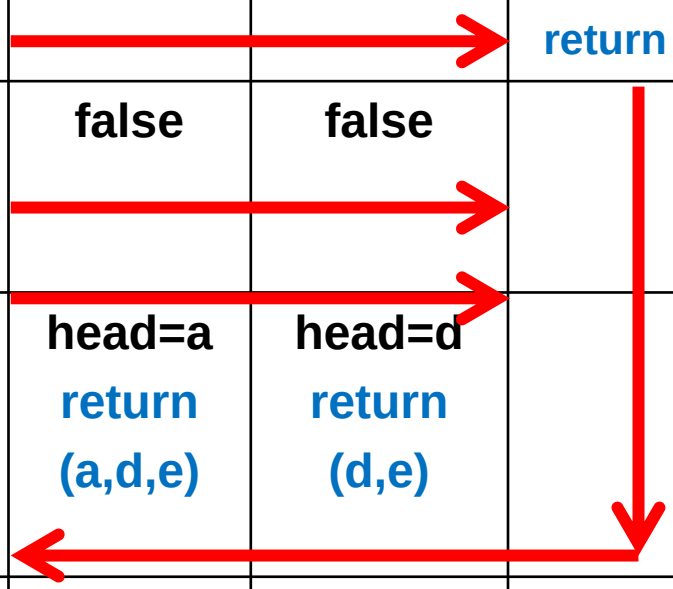
Insert - middle (sorted sequence)

code	Call 1	Call 2	Call 3
insert(el, list) {	**(e,(a,d,g))**	**(e, (d,g))**	**(e, (g))**
if is_empty(list) **return cons(el, list);**	**false**	**false**	**false**
if getval(el)<getval(head(list)) **return cons(el, list);**	**false**	**false**	**true** **return (e,g)**
return(cons(head(list), **insert(el, tail(list)));**	**head=a** **return (a,d,e,g)**	**head=d** **return (d,e,g)**	
}			



Insert - end (sorted sequence)

code	Call 1	Call 2	Call 3
insert(el, list) {	**(e, (a,d))**	**(e, (d))**	**(e, (¤))**
if is_empty(list) **return cons(el, list);**	**false**	**false**	**true** **return (e)**
if getval(el)<getval(head(list)) **return cons(el, list);**	**false**	**false**	
return(cons(head(list), **insert(el, tail(list)));**	**head=a** **return (a,d,e)**	**head=d** **return (d,e)**	
}			

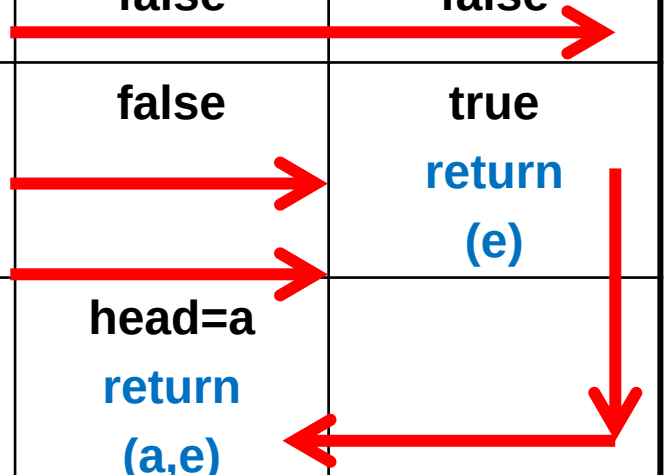


Delete - beginning (sorted sequence)

code	Call 1
delete(el, list) {	**(a, (a,d,e))**
if is_empty(list) return list;	**false**
if getval(el)==getval(head(list)) return tail(list);	**true return (d,e)**
return(cons(head(list), delete(el, tail(list)));	
}	

Delete - middle (sorted sequence)

code	Call 1	Call 2
delete(el, list) {	**(d, (a,d,e))**	**(d, (d,e))**
if is_empty(list) return list;	**false**	**false**
if getval(el)==getval(head(list)) **return tail(list);**	**false**	**true** **return** **(e)**
return(cons(head(list), **delete(el, tail(list)));**	**head=a** **return** **(a,e)**	
}		



Delete - end (sorted sequence)

code	Call 1	Call 2	Call 3
delete(el, list) {	**(e, (a,d,e))**	**(e, (d,e))**	**(e, (e))**
if is_empty(list) return list;	**false**	**false**	**false**
if getval(el)==getval(head(list)) return tail(list);	**false**	**false**	**true** **return (¤)**
return(cons(head(list), delete(el, tail(list)));	**head=a** **return (a,d)**	**head=d** **return (d)**	
}