



Recursion

Examples

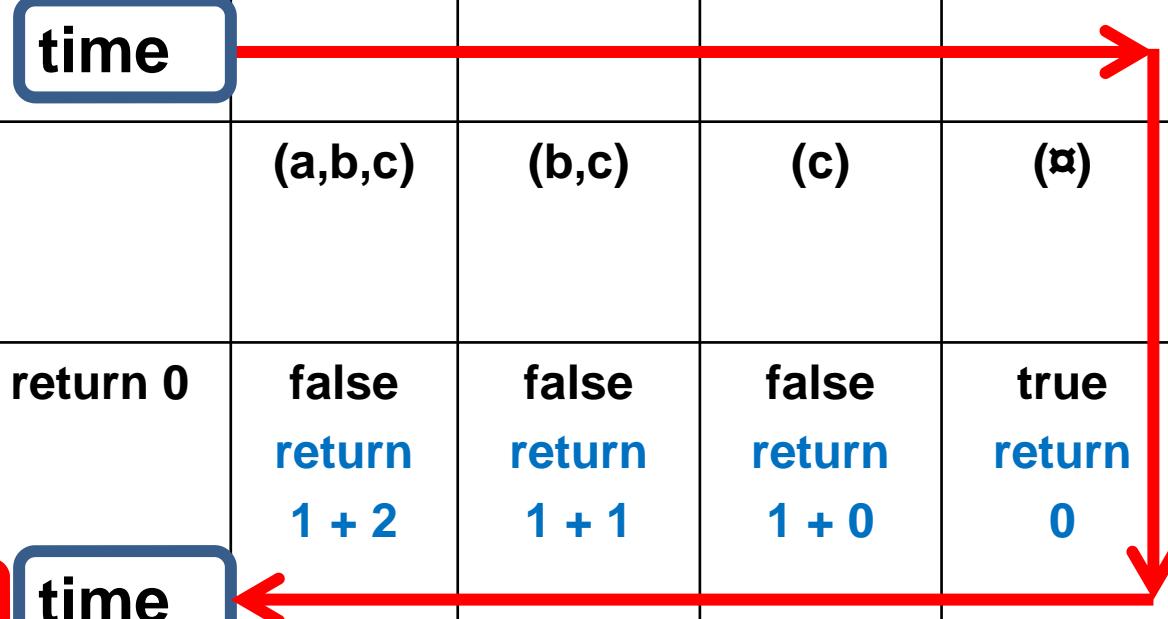
Length (size) of a sequence

code	call 1	call 2	call 3	call 4
<code>int size(list) { if is_empty(list) return 0 else return 1 + size(tail(list)); }</code>	(a,b,c)	(b,c)	(c)	(\emptyset)
	false <code>return 1 + 2</code>	false <code>return 1 + 1</code>	false <code>return 1 + 0</code>	true <code>return 0</code>

time

Output 3

time



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		
}				

The diagram illustrates the execution flow of a recursive function. It shows five columns representing different calls to the function, labeled 'call 1' through 'call 4'. The first column contains the initial code and the first recursive call. The second column contains the second recursive call. The third column contains the third recursive call. The fourth column contains the fourth recursive call. The fifth column contains the final result. Red arrows indicate the flow from one call to the next. A red box highlights the output 'a b c'.

Display list (backwards)

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

Insert - beginning (sorted sequence)

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

Insert - middle (sorted sequence)

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

The diagram illustrates the execution flow between three call frames (Call 1, Call 2, Call 3) for the `insert` function. Red arrows show the flow of control:

- A horizontal red arrow points from Call 1 to Call 2.
- A horizontal red arrow points from Call 2 to Call 3.
- A diagonal red arrow points from Call 3 back up to Call 1.

Insert - end (sorted sequence)

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

The diagram illustrates the execution flow between three function calls. Red arrows show the flow of data and control between the calls:

- A horizontal red arrow points from Call 1 to Call 2.
- A horizontal red arrow points from Call 2 to Call 3.
- A vertical red arrow points from Call 3 down to Call 1.

[Delete - beginning (sorted sequence)]

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

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
<code>delete(el, list) {</code>	(e, (a,d,e))	(e, (d,e))	(e, (e))
<code> if is_empty(list) return list;</code>	false	false	false
<code> if getval(el)==getval(head(list)) return tail(list);</code>	false	false	true return (ξ)
<code> return(cons(head(list), delete(el, tail(list))));</code>	head=a return (a,d)	head=d return (d)	
<code>}</code>			