# Assignment 1

## Exercise 1:

(b): Let the available length of the array be $n$

In the normal case where 'deallocate' is not needed:

The time complexity is on $\Theta(1)$ since the only excution needed is length - k.

In the exceptional case, 'deallocate' is called:

complexity for 'deallocate' is $c \cdot \frac{n}{4} \in \Theta(n)$

By amortised complexity, it is on $\Theta(1)$.

## Exercise 2:

(b):

Let's first consider a list $l$ which is empty:

Obviously, src [e, f, g] for $l$ is well defined as long as $e$ is defined.

Then if $l$ contains a single element:

Obviously, src [e, f, g] for $l$ is also well-defined as long as $f(x)$ is defined.

For $l$ with several elements like $[x_1, x_2 \cdots x_n]$:

As $l$ can be written as two arbitrary lists: $l = concat(l_1, l_2)$, we need to consider whether all these lists can lead to same result when src [e, f, g] is applied.

Firstly, we can derive $src\,[e,f,g](l) = g(src\,[e,f,g](l_1), src\,[e,f,g](l_2))$

① In the case both $l_1$, $l_2$ are not empty:

Clealy, it is a recursion and we will have:

$g(src\,[e,f,g](l_n), src\,[e,f,g](l_{n+1}))$ at final recursion

with $l_n$, $l_{n+1}$ contain only one element and

$src\,[e,f,g](l_n) = f(x_n)$, $src\,[e,f,g](l_{n+1}) = f(x_{n+1})$

with $x_n \in l$, $x_{n+1} \in l$ and by recursing back, we can obvously get same result as $f(x)$ are the same.

② In the case where $l_1$ or $l_2$ is empty:

$src\,[e,f,g](l) = g(src\,[e,f,g](l_1), src\,[e,f,g]([])) = g(src\,[e,f,g](l_1), e)$

$= src\,[e,f,g](l_1)$

For $src\,[e,f,g](l_1)$ in the $g$, we have shown that it will have same result according to previous proof.

Hence, src is well-defined.

(iii):

① For get_length(l), we can define according to src [e, f, g]:

when l is empty, src [e, f, g] (l) = e = 0

when l is singleton: src [e, f, g] (l) = f(x) = 1

when l contains many elements:

src [e, f, g] (l) = g ( src(l₁), src(l₂) ) = get_length(l₁) + get_length(l₂)

Hence, get_length(l) = $\begin{cases} 0 \\ 1 \\ \text{get\_length}(l_1) + \text{get\_length}(l_2) \end{cases}$

② For func(l):

when l is empty, func(l) = e = []

when l is single: func(l) = func(x) = f(x), $x \in l$

else: func(l) = func($l_1$) + func($l_2$), $l_1, l_2 \in l$, $l_1 + l_2 = l$

③ For sublist(l, φ):

when l is empty, sublist(l, φ) = e = []

when l is single, sublist(l, φ) = f(x) = $\begin{cases} [x], & \text{if } \varphi(x) = 1 \\ [], & \text{if } \varphi(x) = 0 \end{cases}$

else: sublist(l, φ) = sublist($l_1$, φ) + sublist($l_2$, φ), $l_1 + l_2 = l$

(iv): Let's consider a list with n elements, by observing previous operations, we can denote that every element in l will be checked which means f(x) will be applied to all elements.

Also, as it is recursion, g(x) will be excuted several times. We can derive that times of the excution of g(x) is propositional to n

Assume that the complexity of f(x) is k, the complexity of g(x) is p then we can derive $O(kn+pn)$