Expected Time vs Amortized Time

$1 + \frac{1}{6}$

```
int roll5()
{
   int roll;
   do
   {
      roll = roll6();
   } while (roll == 5);
   return roll;
}
```

X = number of calls to roll6 in a call to roll5

$$E[X] = \frac{5}{6}\cdot 1 + \frac{1}{6}\cdot\frac{5}{6}\cdot 2 + \frac{1}{6}\cdot\frac{1}{6}\cdot\frac{5}{6}\cdot 3 + \left(\frac{1}{6}\right)^3\cdot\frac{5}{6}\cdot 4 + \cdots$$

$$= \frac{6}{5}$$ $O(1)$ expected

$P(n$ calls to roll 5 take $> 2n$ calls to roll 6$) > 0$

$P(n$ calls to roll 5 take $> 10n$ calls to roll 6$) > 0$

$P(n$ calls to roll 5 take $> n^2$ calls to roll 6$) > 0$

amortized $O(1)$

$P(n$ consecutive adds require $> 4n$ copies$) = 0$

$\frac{4n}{n} = 4$

add worst case $O(n)$

amortized $O(1)$

average

List Implementations

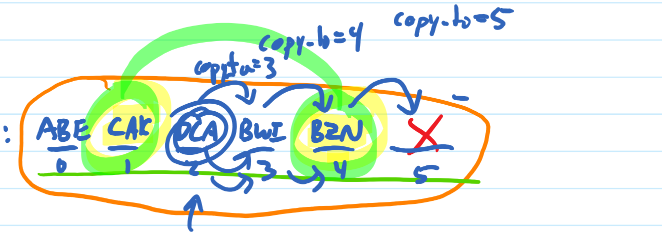
		standard array
size		O(1)
get		O(1)
add	beginning	O(n)
	arbitrary	O(n)
	end	O(1) amortized
remove	beginning	O(n)
	arbitrary	O(n)
	end	O(1) amortized
contains / find		O(n)

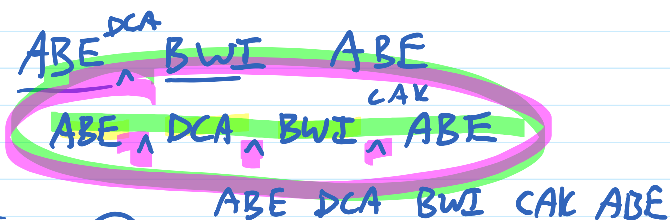
contains → does list contain element?

find → gives index of 1st occurrence

add (l, LHR, 2)

remove (l, 3)





(*l).elements[copy_to] = l->elements[copy_to - 1]

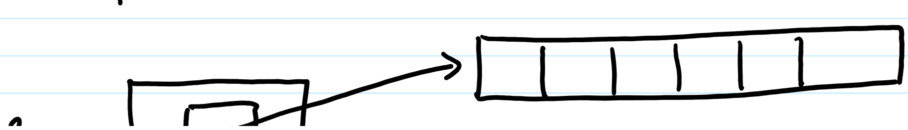
8 7 6 1

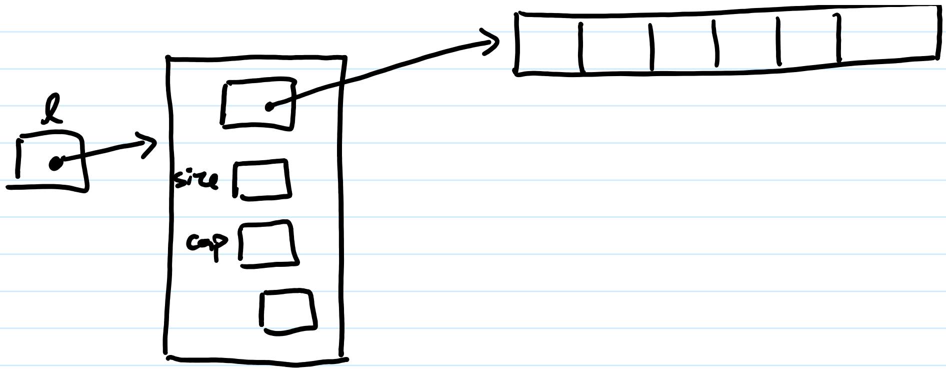
i [2]

to_add [99]

resize after 5th add
1 remove ... resize
1 add ... resize
1 remove ... resize
⋮

Wraparound array





l
size
cap