Regular **asymptotic analysis** looks at the performance of an individual operation.

Amortized analysis deals with the total cost over a number of runs of the routine

- is a worst-case analysis
- gives the average performance of an operation
  - a sequence of invocations of the operation

<u>Example:</u>

- A dynamic array that doubles in size when needed
- Subalg. addLast(v, el)

Regular **asymptotic analysis**
Subalg. addLast(v, el) costs O(n)
*because it **might** need to grow and copy all elements to the new array.*

Amortized analysis
adding an item really costs O(1) on average
takes into account that in order to have to grow, n/2 items must have been added without causing a grow since the previous grow

Amortized analysis on the next code:

Convention:

```
    v.els : 0-based array

Subalg. createEmpty()
    v.n=0;
    v.cap=0;
    v.els=NIL
end_createEmpty

Subalg. addLastWithRealloc1(v,el)          // double capacity
    If v.cap = 0 then
        v.cap=1;
        v.els = new TElement[1]
    Else
        If v.n = v.cap then
            newEls = new TElement[2*v.cap]
            for i=0, v.n-1 do              // copy els
                newEls [i]=v.els[i]
            endfor
            delete [] v.els
            v.els= newEls
            v.cap = 2 * v.cap
        endif
    endif
    v.els[v.n] = el
    v.n=v.n+1
end_addLastWithRealloc1


Subalg. nxaddLast(v)
    createEmpty(v)
    for i:=1, n do
        @read el
        addLastWithRealloc1(v,el)
    endfor
End_nxaddLast
```

```
Subalg. addLastWithRealloc2(v,el)              // cap. increment = 4
        If v.cap = 0 then
                v.cap=1;
                v.els = new TElement[4]
        Else
                If v.n = v.cap then
                        newEls = new TElement[v.cap + 4]
                        for i=0, v.n-1 do                    // copy els
                                newEls [i]=v.els[i]
                        endfor
                        delete [] v.els
                        v.els= newEls
                        v.cap = v.cap + 4
                endif
        endif
        v.els[v.n] = el
        v.n=v.n+1
end_addLastWithRealloc2
```

```
Subalg. removeLastWithShrink1 (v)            // half capacity
    v.n=v.n-1
    If v.n*2 = v.cap then
        newEls = new TElement [ v.cap div 2 ]
        for i=0, v.n-1 do                    // copy els
            newEls [i]=v.els[i]
        endfor
        delete [] v.els
        v.els= newEls
        v.cap = v.cap div 2
    endif
end_removeLastWithShrink1
```