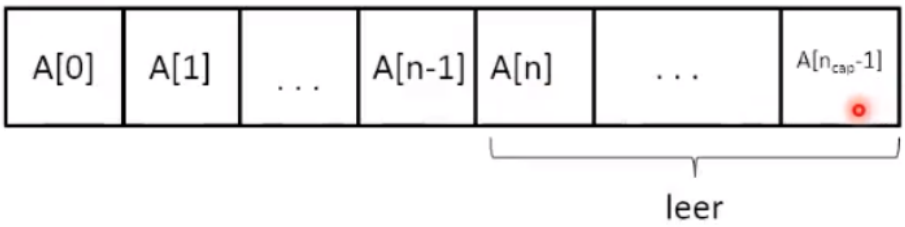
Eigenschaften

- herkömmliches [[Array]]
  Die effizienten Operationen auf einer Liste L sind:
  - L[i].....Zugriff auf Index i (Lesen oder Schreiben)
  - L.append(x)...Element x am Ende der Liste anhängen
  - L.pop().....Letztes Element zurückgeben und löschen
  - hat jedoch fixe Größe
- dynamische Größe
- gleich effiziente Operationen
  - amortisierter konstanter Zeit
- Ein dynamisches Array ist ein Tupel $DA = (A, n)$
  - A ...Array der Größe n_{cap} (d.h. $len(A) = n_{cap}$)
  - n ....Anzahl der gespeicherten Elemente

Grundprinzip



- ersten n Elemente sind befüllt
  - restlichen können befüllt werden
- Größe wird verdoppelt, wenn $n_{cap} + 1$ Elemente notwendig
  - Array mit doppelter Größe anlegen
  - Elemente hinüberkopieren
- analog beim Löschen
  Wenn man löscht und weniger als $\frac{n_{cap}}{4}$ Elemente hat, wird die Größe halbiert (um Speicher zu sparen).

Operationen

- Einfügen

Algorithmus 19 : add($\mathcal{DA}, x$)

Input : A dynamic array $\mathcal{DA}$ and the element x to be added
Output : $\mathcal{DA}$ with x appended at the end

```
[1] if n < n_cap then
[2]     A[n] ← x
[3]     return (A, n + 1)
[4] else
[5]     A_old ← A
[6]     A = new Array of size 2n_cap
[7]     A[0, ··· , n − 1] ← A_old[0, ··· , n − 1]
[8]     free(A_old)
[9]     return add((A, n), x)
```

- Löschen

Algorithmus 20 : delete($\mathcal{DA}$)

Input : A dynamic array $\mathcal{DA}$
Output : $\mathcal{DA}$ without the last element

```
[1] if n > n_cap/4 then
[2]     return (A, n − 1)
[3] else
[4]     A_old ← A
[5]     A = new Array of size n_cap/2
[6]     A[0, ··· , n − 1] ← A_old[0, ··· , n − 1]
[7]     free(A_old)
[8]     return delete((A, n))
```

Speicherverbrauch und Laufzeit

Der Speicherverbrauch ist durch n_{cap} gegeben.
Durch die `add-delete`-Konstruktion haben wir jederzeit

$$n \leq n_{cap} \leq 4n$$

Also: $S(n) \in \Theta(n)$

-
- Einfügen und Löschen haben potentiell schlechte Laufzeit $\Omega(n)$
  - wenn erweitert/geschrumpft wird

Amortisiere Laufzeit:

> Man betrachte k aufeinanderfolgende `add` oder `delete`-Operationen in beliebiger Reihenfolge auf einem anfangs leeren dynamischen Array.
> Dann ist die Laufzeit für diese k Operationen $T(k) = O(k)$.

  - [[Amortisierte Analyse]]