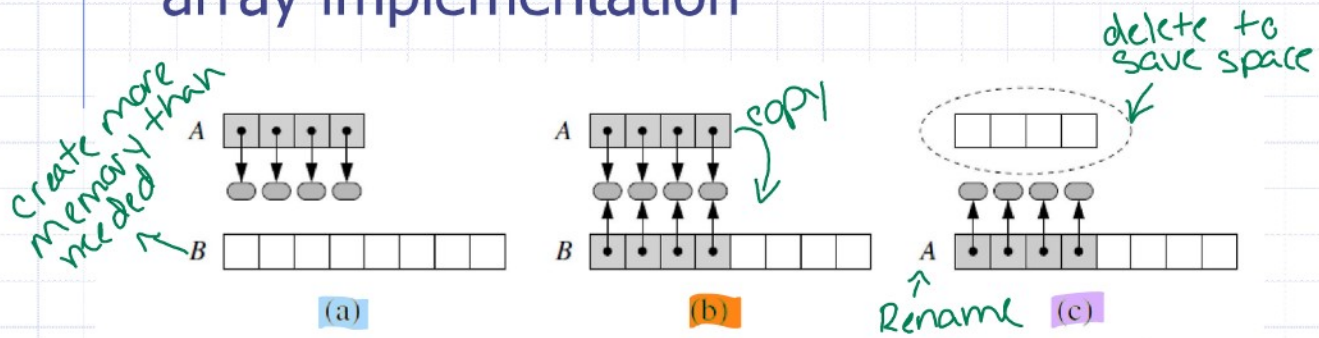
List & tuple

Dynamic Array

- A Python list can grow.
- Python list class uses an efficient dynamic array implementation



(a) create new array *B* (b) store elements of *A* in *B* (c) reassign reference *A* to the new array.

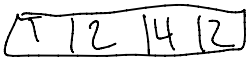

```
B = [ ] * 2n # doubling space → only need 1 extra cell, why double?
                                ↳ why not open up as needed (append)
                                  ↳ to eliminate repetitive steps/operations
for i=0 : n-1 # whole array from i=0 up to n-1
B[i] = A[i]   # Dump elements of A -> B
A = B         # Rename
```

if full -> then append

① `[ ] * n` # create empty list of size n

OR

② extend list all at once
↳ `list.extend(another_list)` # copy & extend



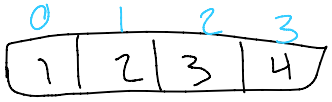
`data.index(4)`
loop
↳ start at index 0
↳ compare each value to 4

O(k+1) → # operations
↳ k = first occurance index location
complexity is at most n
↳ O(n)

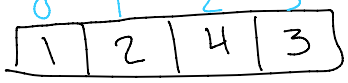
---

k = index

data 1



data 2



3 = k+1
O(k+1)

Want to see if they are equivalent
↳ compare each element

k=0 ↳ 1 == 1 ✓
k=1 ↳ 2 == 2 ✓
k=2 ↳ 3 == 4 ✗ stops
+1 = 3

Worst case:
↳ go through whole list
↳ O(n)

Slicing → exclusive?

data[j:k] → non - inclusive

s[2:4] → s

0	1	2	3	4	5
0	1	2	3	4	5

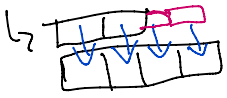
→ index

s[2] → 2 O(1)
s[3] → 3 O(1)
} 2 operations

k-j+1
↳ storing value/slice in another array
O(n)

---

data1 + data2
↳ add all indices up
↳ same size → n operations → O(n) → best case
↳ not same size → append zeros → O(2n) → worst case

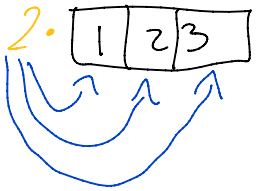


4+2

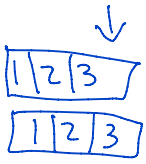
---

c * data

2.



↳ not multiplying data by constant
↳ duplicating the lists
↳ O(cn) → build list c times



↳ addition of sets
↳ merges lists together

data[i] = val
↳ setting value

[1, 2, 3, 4]

now points to 100
1 operation
↓

data[2] = 100 → [1, 2, 100, 4]

---

data.append(value)

↳ does list have enough space?
  ↳ best case O(1) → 1 operation → over time → experimentally
  ↳ worst case → O(n) → when it needs to grow
    ↳ mathematically
↳ based on dynamic array
  ↳ doesn't actually double for every growth spurt
  ↳ doesn't add every time → not n² / arithmatic

something in between
↳ dynamic array

↳ either start w/:
  ↳ size = 0 → build
  OR
  ↳ size = n / empty
    ↳ make 2n if needed

---

data.insert(K, value) O(n - K + 1) (4 - 1 + 1)

[1, 2, 100, 4] — moving 3 cells right

↳ worst case → insert @ index 0
  ↳ n shifts → O(n)

data.insert(1, 2000) → [1, 2000, 2, 100, 4]
index 1 → 4

---

data.pop() → insert @ end

↳ easiest b/c all other indices don't change → only last

data.pop(k)

↳pop data @ index k
↳method of class list

del data[k]

↳delete data @ index k
↳works not just on lists

[1, 2000, 2, 100]

↳pop.data(2)
↳[1, 2000, ~~2~~, 100]
move

↳del data [2]
↳[1, 2000, ~~2~~, 100]
move

O(n-k)
↓ ↓
4 2

2 operations
① Delete
② moving

---

data.remove (value)

↳searches for value -> found -> delete
↳based on values not indices

---

data1.extend(data2) O(n_2)
↳size of data2

adding list2 to list1

↳arithmatic version:
↳data1 += data2

---

data.reverse( )

[1, 2000, 2, 100] -> [100, 2, 2000, 1]

↳can recreate w/ recursion pattern

↳best case sorting algorithm: O(n log n)
memorize
↳Tim sort