Introduction to Arrays

Mar 28, 2022

AGENDA:

- Little more on Space Complexity
- Arrays / List Data structure
  - Access / Retrieve
  - Update
  - Iteration over an array
- 3-4 interesting problems

Space Complexity

Def:

How does space (memory) taken up by a program vary as N → ∞ ? (N = input)

V.I:

Space taken up by program
= Space taken up by Input + Auxiliary / Extra space.

Input: if the input is array → O(n)

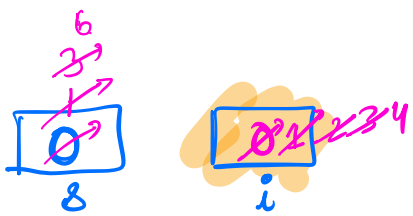
Auxiliary / Extra space: O(1)

Only this is considered while calculating space complexity.

1. Space complexity by below code:

```
s = 0
for i in range(0,10):
    s = s+i
```

O(1)

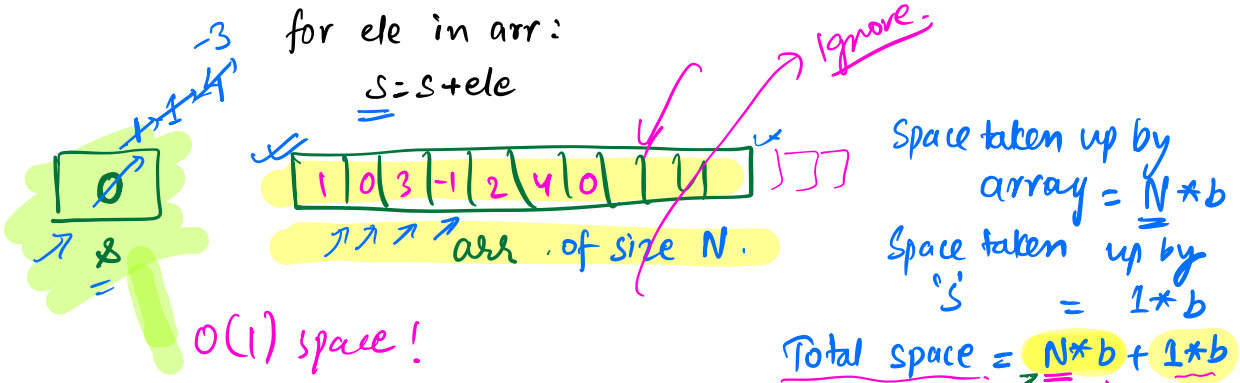


S(N) = 2 * b

O(S(N)) = O(1)

2. Space complexity by below code:

```
// arr of size N is passed as input.
s = 0
```



Space taken up by array = $N * b$

Space taken up by 's' = $1 * b$

Total space = $N*b + 1*b$

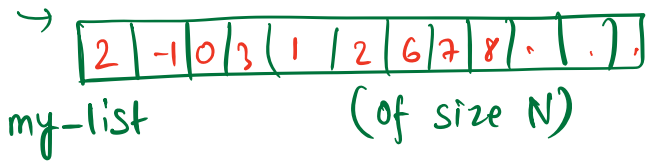
✗✗ $O(S(N)) = O(N)$

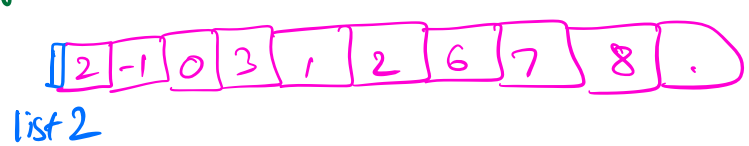
✓✓ $O(S(N)) = O(1)$

3. Space complexity by below code:

```
def copy_list(my_list):
    list2 = []
    for x in my_list:
        list2.append(x)
    return list2
```

O(N) space





Total space = Input space + (list_2)

Input space: $N*b$ (my_list)

(list_2) → $N*b$ bytes.

Ignored!

$O(S(n)) = O(N)$

ARRAYS / LIST.

What is an array?

** Collection of elements → same-type. { C++, Java etc.

→ different_types { Python.

array → [1, 2, 5, 6, -3, 0]

→ [1, 2, "scaler", true, 5.0]

why do we need array?

a = 1
b = 2
c = 5
d = 6
e = -3
f = 0

100 students in the class.
Store their marks.

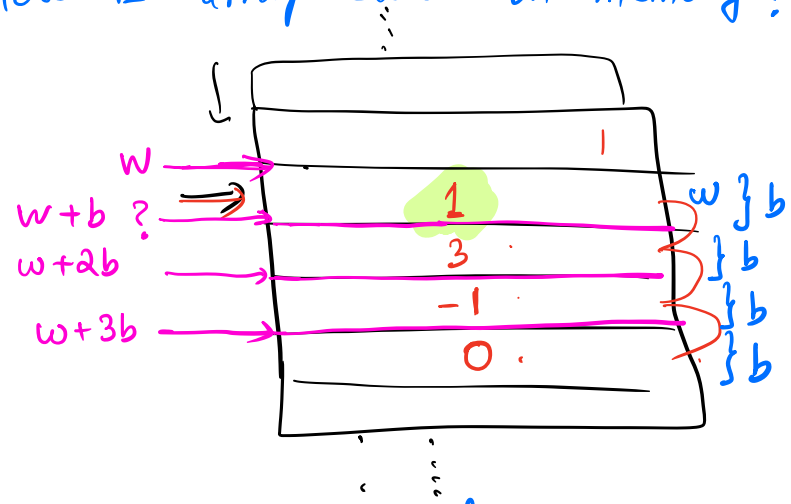
a1 = 97
a2 = 83
a3 = 76
a4 = 100
.
.
a100 = ?

* What is the max marks?

* What is the marks of 76th student?

** Array is a contigous block of memory.

How is array stored in memory?



array of size 4.

[1, 3, -1, 0]

** w → Starting address of array.

3rd element → ? (-1)

↓

w+2b

w → , nth element by {w+(n-1)b}

** what is the time complexity of accessing an array element?

RAM → Random Access Memory.

Address → RAM → Value that is stored there

input → RAM → output

O(1) time

When you declare an array, you know the starting address.

ar = (w) → [1, 3, -1, 0, 2, 6, 4]

0 1 2 3 4 5 6

ar[3] = ?

w + 3b.

AM

O

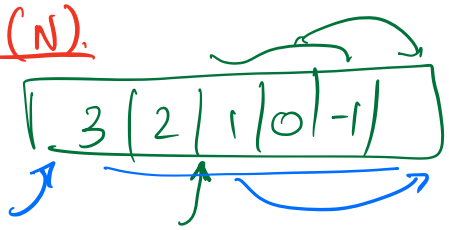
in constant time

w: starting address of the array.

** Recap.

1) Accessing the element / Retrieve → O(1)

2) Insert → O(N).



3) Iterate over an array?

↓

in constant time

Q. Given an array, find sum of array elements.

arr = [7, 17, 1, 2, 5, 4, 3]

* How do you access an array element?

Q. Print sum of 1st and 5th element.

A: [5, -4, 8, 9, 10]

1st elem. A[0]

5th element = A[4].

A[0] + A[4]

** Accessing each element of the array to get total sum:

Approach: Iterate over the array, sum each element.

```
s = 0
N = len(arr)
for i in range(0,N):
    s = s + arr[i]
return s
```

i → 0 to N-1.

Python

```
s = 0
for element in arr:
    s = s + element
return s
```

No. of iterations = N

TC: O(N)

SC: O(1)

Problem Solving

Q. Given N array elements, find no. of elements having atleast 1 element greater than itself.

e.g. [2, 5, 1, 4, 8, 0, 8, 1, 3, 8]

Ans = ? 7 ✓

How many 8's. → 3

10 − 3 = 7

e.g. [1, 2, 3, 4, 5, 5, 6, 7, 7, 7]

10 − 3 = 7 → 7 ✓

Approach :

1. Find total no. of elements = N
2. Find Max no. of list = maxim
3. Count how many max we have. = count
4. Ans = N − count.

Pseudo-code :

```
N = len(arr)
# maxi = max(arr)        <- Don't use this
maxi = arr[0]
for i in range(1, N):
    maxi = max(maxi, arr[i])
```

Iterate and compare.

if arr[i] > maxi :
    maxi = arr[i]

```
# Count how many maxi are there.
count = 0
for element in arr:
    if element == maxi:
        count += 1
```

Iterating and compare.

```
return N - count
```

Time complexity -> O(N)

Space -> O(1)

Break till : 10:10 pm.

Q. Reverse the given array.

Input = [5, 2, 7, -1, 0]

Output = [0, -1, 7, 2, 5]

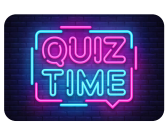
Naive approach.

```
# inp = [5, 2, 7, -1, 0]
# Create a new array.

outp = [] [0, -1, 7, 2, 5]

                                  moving back.
for i in range(n-1, -1, -1):      [n-1, 0]
    outp.append(inp[i])
```

→ range(start, end) → [start, end-1]

→ range(start, end, step) → [start, start-1, end+1, start-2 ...]

step = -1



Time complexity: O(N)

Space complexity: O(N) → You created a new array.

a[::-1]

Can we optimise ?

↳ Time
↳ Space.

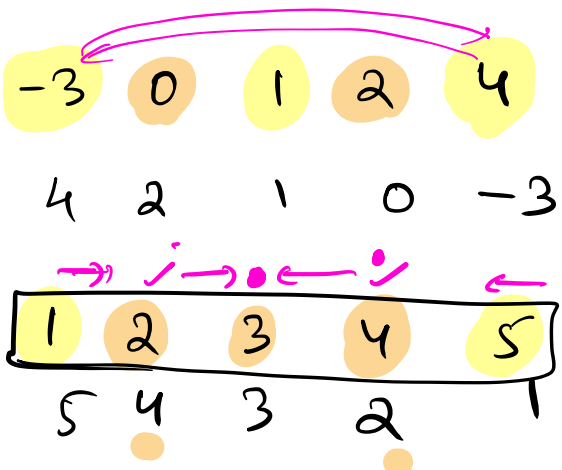
O(N) → ✓ Best!

Can we do it in O(1) ?

inp = | -3 | 0 | 1 | 2 | 4 |

↳ Modify this array itself.

Inplace algorithm



Approach.

Start → 0
end → N-1

Pseudo-code.

```
def reverse(arr):
    N = len(arr)
    start = 0
    end = N-1

    while(start < end):

        swap(arr[start], arr[end])
        start += 1
        end -= 1
```

st	st	st	end	end	end
5	3	2	-1	0	6

end st

```
fun swap(a, b):

    a, b = b, a.   // In python
```

```
a = b
b = a
```

```
temp = a
a = b
b = temp
```

Time: O(N)

No. of iterations = N/2 = O(N)

Space: O(1) ← Inplace.

Q.

Reverse part of the given array.

e.g. → Input: [4, 3, 5, 1, 0, 7, 2, 8]

start = 3 (index 3)

end = 6 (index 6)

→ 0-indexed

Output: [4 3 5 2 7 0 1 8]

```
def reverse(arr, start, end):

    N = len(arr)
    start = 0
    end = N-1
    while (start < end):
        swap(arr[start], arr[end])
        start += 1
        end -= 1
```

Time complexity →

Best case → ? O(1)

start = end

Worst case → O(N)

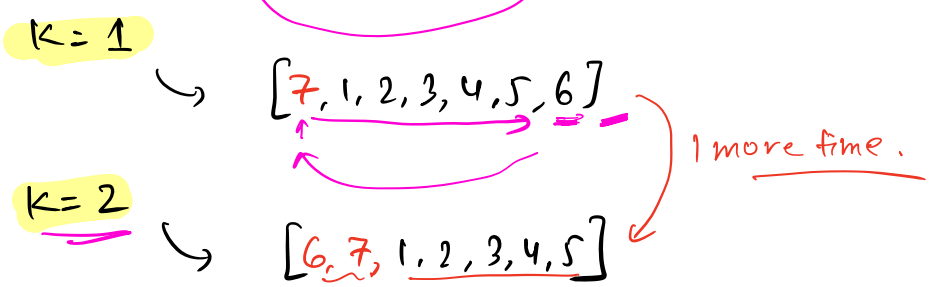
start = 0, end = N-1

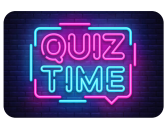
Time complexity : O(N)
Space complexity : O(1)

Q. ** Very interesting Interview Problem. "Google/Amazon"

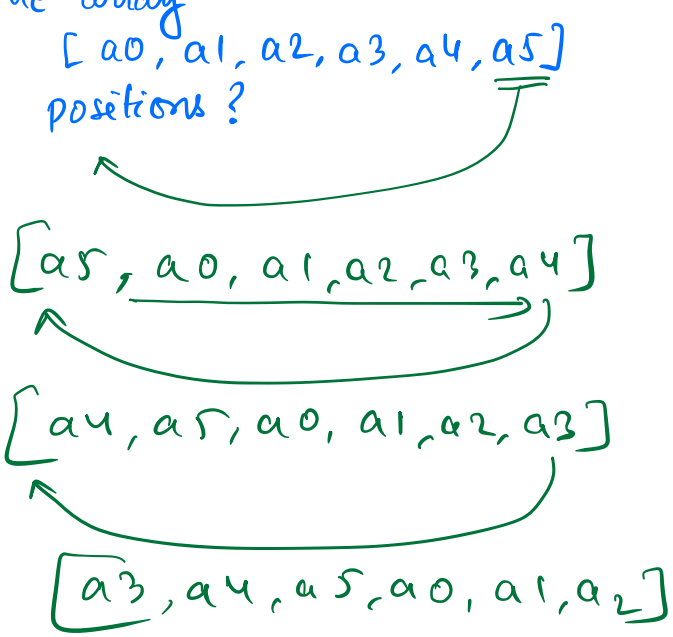
Given an input array, rotate the given array 'k' times from left to right.

e.g. Input: [1, 2, 3, 4, 5, 6, 7]





Rotate the array [a0, a1, a2, a3, a4, a5] by 3 positions?



** What happens if k > N? **

Array has 5 elements.

Rotate the array k times.

arr = [1, 2, 3, 4, 5]

k = 1 : 5, 1, 2, 3, 4

k = 2 : 4, 5, 1, 2, 3

k = 3 : 3, 4, 5, 1, 2

k = 4 : 2, 3, 4, 5, 1

k = 5 : 1, 2, 3, 4, 5 ← k = 0 5 % 5

k = 6 : 5, 1, 2, 3, 4 ← k = 1 6 % 5

k = 7 : 4, 5, 1, 2, 3 ← k = 2 7 % 5

Conclusion:

**

if k > N:
    k = k % N

Brute Force approach.

* Pick up the last element
  Insert it at the first.

→ O(N).

How many times you need to perform this? → K times.

Total. TC: O(N*K)

K is an input.

Quadratic

N → 10^5
K → 10^5
10^{10}

Space complexity:
Time complexity:

** Let's try to optimise on Time.

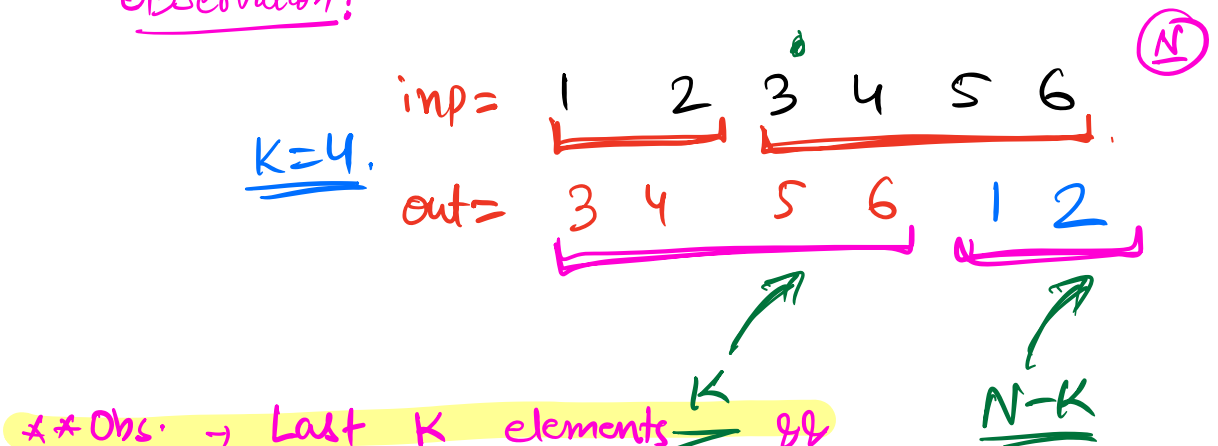
K < N K = K % N

Original array : [1, 2, 3, 4, 5]
K = 3 ; New array : [3, 4, 5, 1, 2]

** We will be using a new array.

How can we directly translate original values to new array?

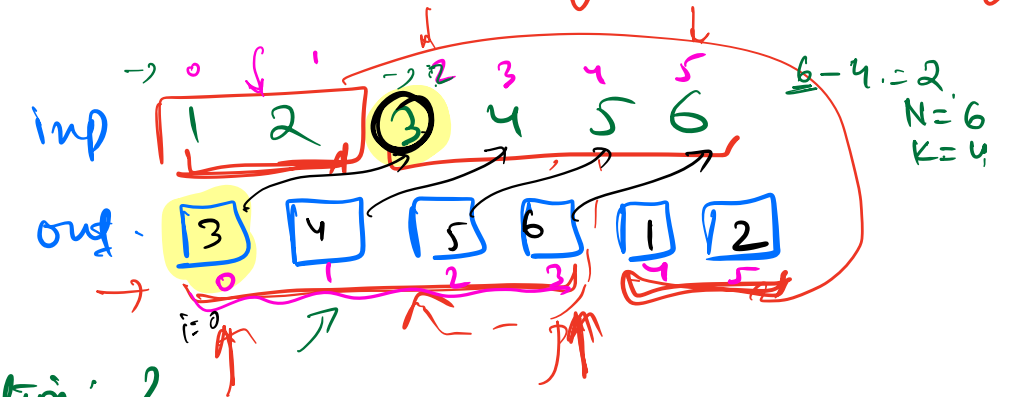
Observation!



** Obs. → Last K elements ⇒ come at start.

1st n-k elements go at end.

* Pick up values from your input array.
Start filling values in your output array.



Translation: ?

For first 4 values,

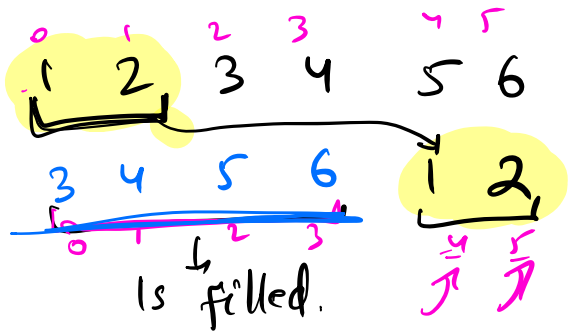
i = 0, 1, 2, 3

out[0] = inp[2]
out[1] = inp[3]
out[2] = inp[4]
out[3] = inp[5]

In terms of N, i, K
6-4=2
N-K ✓
N-K+1
N-K+2
N-K+3

$N-K+i$

1st part { for $i = 0, 1, 2, 3$ → out[i] = input[N-K+i]



→ out[4] = inp[0]
→ out[5] = inp[1]

K+1
K+2

for $i = 0, 1$:

→ output[k+i] = Input[i]

Pseudo-code.

```
K = K % N
out = [0]*N                          0,1,2,3
filled up 1st part. { for i in range(0, K):        ← K
                        out[i] = inp[N-K+i]
for i in range(0, N-K):                            N-K
    out[ k+i ] = inp[i]
```

Time complexity : O(N) // ← No

Space " : O(N) // ← Yes.

↳ You are creating a new array.

** Can we optimise on space ?

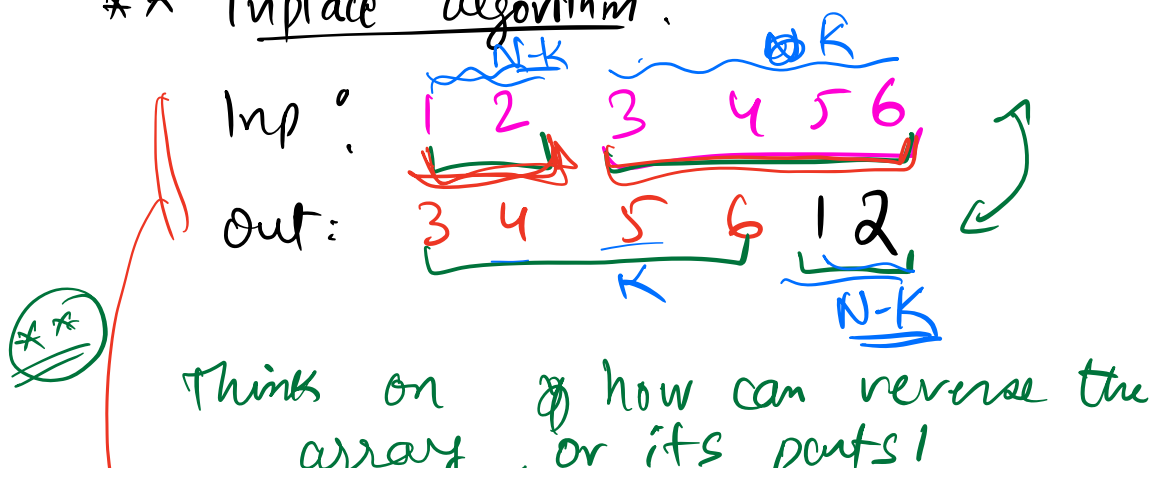
Inp: [1, 2, 3, 4, 5, 6]

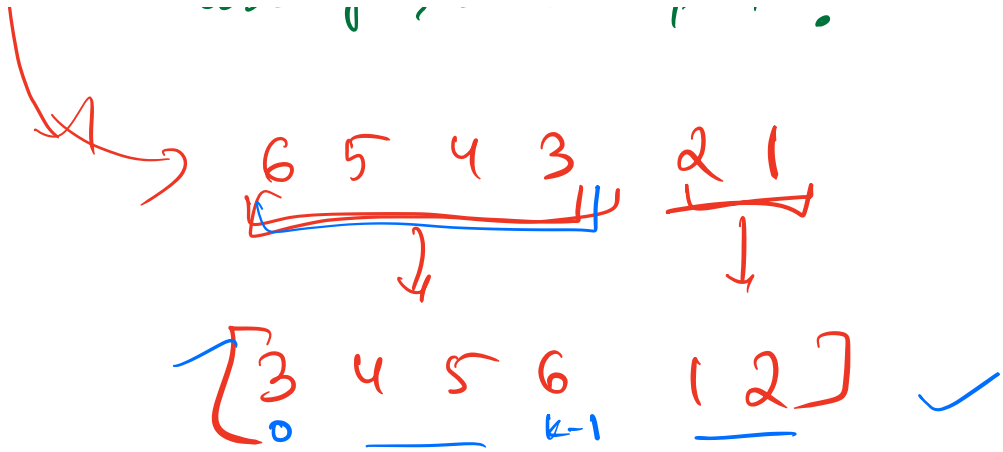
K=4; Output: [3, 4, 5, 6, 1, 2]

** Observation !

Trying to do in O(1) space complexity.

** Inplace algorithm.





** ① Reverse the whole array. TC: O(N) — SC: O(1)

② Reverse [array, 0, K-1] TC: O(N) — SC: O(1)

③ Reverse [array, K, N-1] TC: O(N) — SC: O(1)

Total:

TC: O(N)

SC: O(1)

Doubt Session.

insert(my_list, 0, num)

my_list.insert(element, location)

[1, 2, "Hello", 3, true]

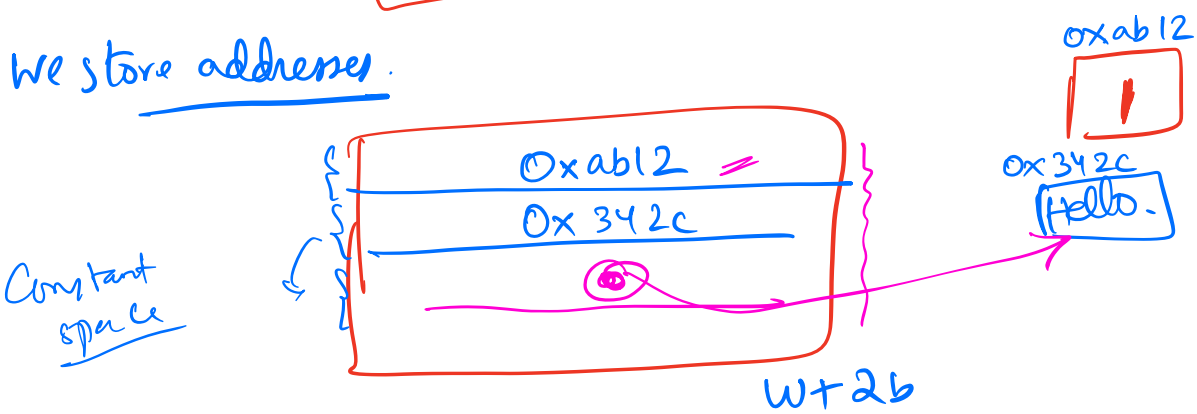
36 bytes · 2b bytes · 1 byte

W + 3b

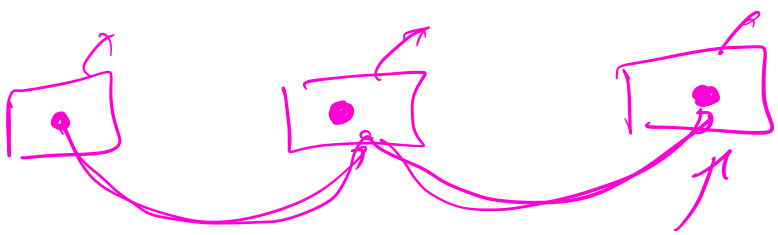
← C++, Java

X.

We store addresses.



W + (n-1)b



```
i = 1
while (i < N):
    x = i        x = 1
    while (x > 0):
```

x -= 1 }

i += 1

i = 1.	inner loop iter.	
1	1	1
2	2	2
3	3	3
4	4	
⋮	⋮	⋮
N	N	2^N
2^N	$1+2+3+4\cdots+N$	$\frac{2^N(2^N+1)}{2}$

$= \frac{N(N+1)}{2}$

$O(N^2)$

$1 << N$

2^N

$2^N * 2^N$

$O(2^{N+1})$

4^N

$(2 \times 2 \times \cdots)$ (N times) $\times$ $(2 \times 2 \times \cdots)$ (N times)

$= (4 \times 4 \times 4 \times \cdots)$ (N times) $= 4^N$

$O(4^N)$