Data Structures

Insertion

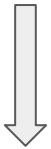
Mostafa S. Ibrahim
Teaching, Training and Coaching since for than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Insertion operation

- Another critical feature is insertion
- Let’s say we have the array on the right
  - {10, 8, 7, 5, 3}
- Now we want to insert value 17 in index 2
  - New array: {10, 8, 17, 7, 5, 3}
- How to do it? Think for 5 minutes

Index	0	1	2	3	4
values	**10**	**8**	**7**	**5**	**3**

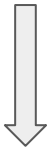


Index	0	1	2	3	4	5
values	**10**	**8**	**17**	**7**	**5**	**3**

Insertion operation

- {10, 8, 7, 5, 3}
- First, we need to shift all values from index 2 to the right side
- {10, 8, **EMPTY**, 7, 5, 3}
- Then add the value in the requested position
- {10, 8, 17, 7, 5, 3}
- Take 10 minutes to implement
  - Assume **idx is between** [0 and size-1]

Index	0	1	2	3	4
values	**10**	**8**	**7**	**5**	**3**



Index	0	1	2	3	4	5
values	**10**	**8**	**17**	**7**	**5**	**3**

Insertion operation

```
def insert(self, idx, item):
    assert 0 <= idx < self.size

    if self.size == self._capacity:
        # we can't add any more
        self.expand_capacity()

    # Shift all the data to right first CORRECTLY
    # Shift range (idx, size-1) from the back
    for p in range(self.size - 1, idx - 1, - 1):
        self.memory[p + 1] = self.memory[p]
    self.memory[idx] = item
    self.size += 1

    # Common mistake to iterate from begin to end
    # the whole array right array will be arr[idx]
```

```
array = Array(0)
array.append(56)
array.append('hello')
print(array)
#56, hello,

array.insert(0, 'A0')
print(array)
# A0, 56, hello,

array.insert(2, 'A2')
print(array)
# A0, 56, A2, hello,

array.insert(1, -9)
print(array)
# A0, -9, 56, A2, hello,
```

Efficiency

- How fast is our insertion?
- If we inserted at position idx, we need first to shift the elements to the right
- Let **number of shiftings** k = size - idx ⇒ we need 3k+1 steps
- Then we need to put the element and increase the size : 2 steps
- Total ~**3k + 2**. Let's drop constants ⇒ K
  - In complexity section, you will know why to always drop constants
- When idx = 0 (shift whole array) ⇒ size steps: the **worst** case
- Overall: we need **linear number of** steps in the **worst case** (size)
- This means we need to be careful and not call this function a lot!

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."