Problem: Input $\longrightarrow$ Output

Algorithm:
- (I) correctness
- (II) efficient
  - time efficient
  - space efficient

Sorting Problem: Input: $a_1, a_2, \ldots, a_n$ (array of size n)

Output: reordering (permutation)

$a_{i_1}, a_{i_2}, \ldots, a_{i_n}$

$a_{i_1} \le a_{i_2} \le \cdots \le a_{i_n}$

Example

Input: 7, 10, 1, -2, 4

Output: -2, 1, 4, 7, 10

Insertion sort

Merge sort

Quick sort

linear sorting (radix sort, bucket sort, counting sort)

Insertion sort

Pseudo-code

```
Insertion_Sort ( A[1 ... n] )
    for i=1 to n
(1)         aux = A[i]
(2)         j = i-1
(3)         while ( j>0 && A[j] > aux )
                A[j+1] = A[j]
                j = j-1
(4)         A[j+1] = aux
```

constant time (c')

time to run: t_i

Example

Input: 7 9 6 1 8

i=1 : $\boxed{7}$ 9 6 1 8 → A[1] sorted

i=2 : $\boxed{7 \quad 9}$ 6 1 8 → A[1 2] sorted

i=3 : 7 6 9 1 8

6 7 9 1 8 → A[1 2 3] sorted

i=4: 1 6 7 9 8 → A[1 ...4] sorted

$i=5$ 1 6 7 8 9 $\rightarrow$ $A[1 \dots 5]$ sorted

	(1)	(2)	(3)	(4)
SC : constant in-place	constant	constant	X	X
TC	constant c	constant c	$t_1+t_2+\dots+t_n$	constant c

$t_i :=$ time needed to run line (3)

for-loop running time:

$i=1$: $\underbrace{(1)+(2)+(4)}_{3c} + \underbrace{(3)}_{t_1} = 3c+t_1$

$+$

$i=2$: $3c+t_2$

$+$

$\vdots$

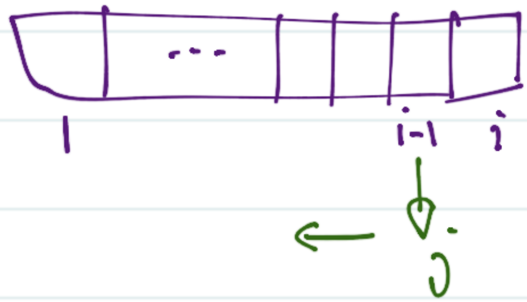
$+$

$i=n$: $3c+t_n$

$=$

$$\text{total TC} = \underbrace{3c+\dots+3c}_{n \text{ times}} + (t_1+\dots+t_n) = (3c)n + \sum_{i=1}^{n} t_i$$

What is t_i?



- Each iteration for j takes constant time.
- How many iterations do we need for j until while-loop terminates?

best case: 1 iteration

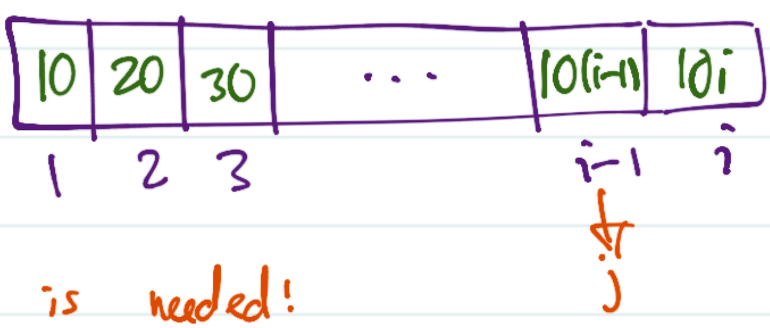
worst case: i iterations

Conclusion

$$\boxed{c' \le t_i \le c'i}$$

sorted — reversely sorted

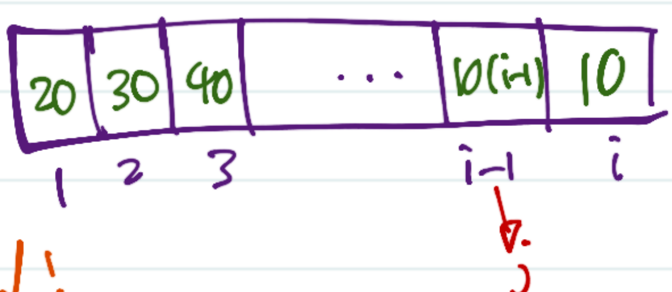
best case:



only 1 iteration is needed!

$$t_i = c'$$

worst case:



i iterations are needed!

$$t_i = \underbrace{c' + c' + \cdots + c'}_{i \text{ times}} = c'i$$

• best case (the array is already sorted)

$$\text{total TC} = (3c)n + \sum_{i=1}^{n} t_i \overset{(*)}{=} (3c)n + \underbrace{c' + \cdots + c'}_{n \text{ times}}$$

(*) $t_i = c'$ for all i

$$\Rightarrow \text{total TC} = (3c)n + c'n = \underbrace{(3c + c')}_{\text{constant}} n$$

TC of Insertion sort is linear in the input size.

• worst case (the array is reversely sorted)

$$\text{total TC} = (3c)n + \sum_{i=1}^{n} t_i \overset{(*)}{=} (3c)n + \sum_{i=1}^{n} c'i$$

(*) $t_i = c'i$ for all i

$$\Rightarrow \text{total TC} = (3c)n + c' \sum_{i=1}^{n} i$$

$$\boxed{1 + 2 + \cdots + n = \frac{n(n+1)}{2}}$$

$$= (3c)n + c' \frac{n(n+1)}{2}$$

$$= (3c)n + \frac{c'}{2}n^2 + \frac{c'}{2}n$$

$$= \underbrace{\frac{c'}{2}}_{\text{constant}} n^2 + \underbrace{\left(3c + \frac{c'}{2}\right)}_{\text{constant}} n$$

TC of insertion sort is quadratic in the input size.

TC of Insertion Sort:

Input array:	sorted / almost sorted	average case.	almost reversely sorted / reversely sorted
TC:	linear n	quadratic n^2	quadratic n^2

Question: What is the TC of Insertion sort for a given arbitrary (random) array?

On average, we need $i/2$ iterations for the while-loop.

Replace $t_i = c' \cdot i/_2$ and deduce that TC is quadratic.

Proof of Correctness: Insertion Sort

Loop invariant: At the end of the for-loop iteration for index i, the first i elements of the array become sorted with respect to each other.

How to prove that loop invariant is satisfied for all input arrays?

$$i=1 \xrightarrow{(\mathrm{II})} i=2 \xrightarrow{(\mathrm{II})} i=3 \longrightarrow \cdots \longrightarrow i=n-1 \xrightarrow{(\mathrm{II})} i=n \quad (\mathrm{III})$$

$i=1$ ↓ initialization

$i=n$ ↓ the algo works!

(I) **Initialization:** Check that the loop invariant holds for $i=1$:

the first element is always sorted wrt itself!

(II) **Maintenance:** **Assumption:** assume that the loop invariant holds for $i=k$

Conclusion: prove that the loop invariant holds for $i=k+1$

If the first k elements are sorted, in the next iteration of the for-loop, we find the correct position of the $k+1^{th}$

element among the first k elements. So all the first k+1 elements become sorted wrt each other.

(III) Termination: Once the algo terminates, by (II), we know that the loop invariant holds for $i=n$. It means that the whole array becomes sorted. So the algo works perfectly!