Analysis of Algorithms [Cormen 2.1]

What? Measure efficiency of an algorithm on its use of resources
e.g., time, space, ...

Why? - Design Better Algorithms
- Compare existing ones

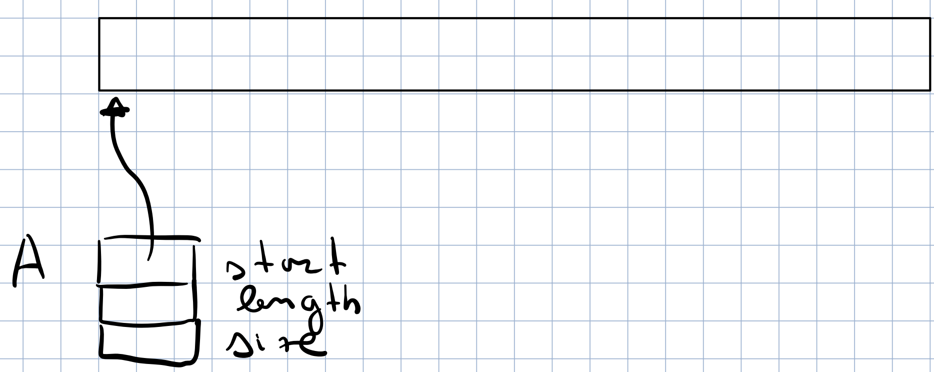
A first example

Sorting problem

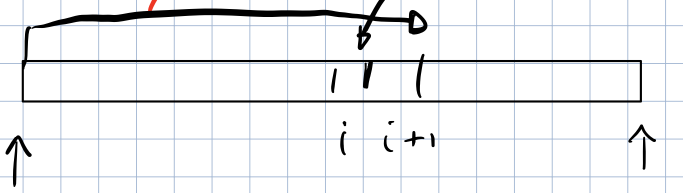
A sequence $A[1, n]$ of n elements

Goal: Find a permutation $A'[1, n]$ of A
such that $A'[i] \leq A[i+1]$, $\forall i \in [1, n-1]$

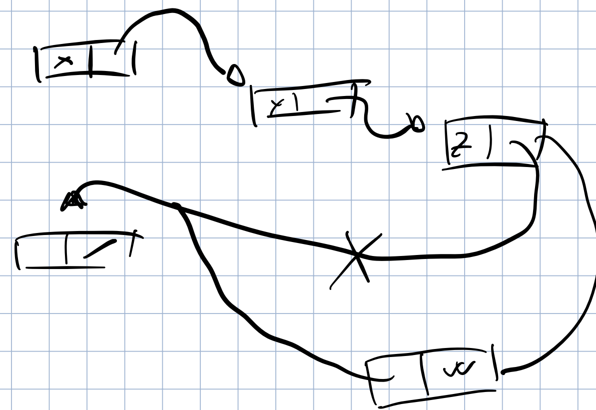
→ index == offset



Array



List

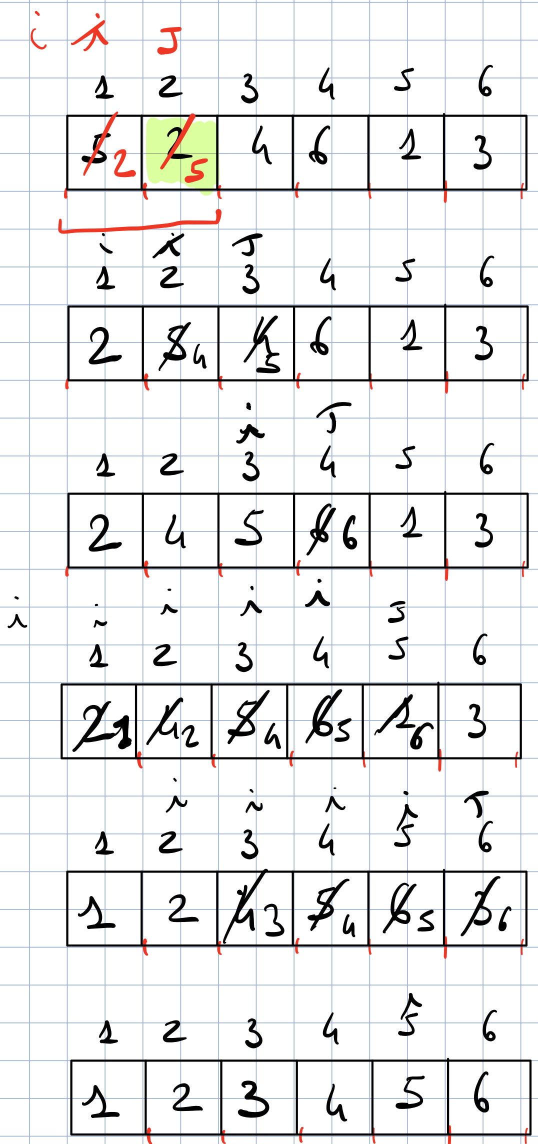


Insertion Sort

```
Insertion Sort (A)

for j = 2 to A.length
        key = A[j]
        // insert key in already sorted A[1, j-1]
        i = j - 1
        while i > 0 and A[i] > key
                A[i+1] = A[i]
                i = i - 1
        A[i+1] = key
```

Running Example



key = 2

key = 4

key = 6

key = 1

key = 3

```
Insertion Sort (A)
for j = 2 to A.length
        key = A[j]
        // insert key in already sorted A[1, j-1]
        i = j - 1
        while i > 0 and A[i] > key
                A[i+1] = A[i]
                i = i-1
        A[i+1] = key
```

Note :

at iteration j

$A[1, j-1]$

is sorted

VisuAlgo

Inplace : Does not use extra space apart from few variables and original array

Stable A ..... 8 .... 8 .... 8 .... 8 ... 8

A_{sorted} ..... 8 8 8 8 8 .....

	Inplace	Stable
Insertion Sort	✓	✓
Selection Sort	✓	✗
Bubble Sort	✓	✓

8 8 1 $\rightarrow$ 8 1 8

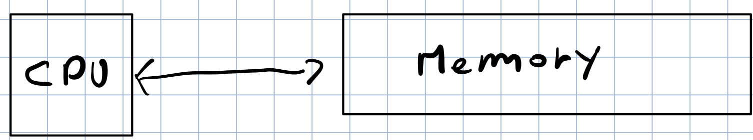
1 8 8

Analysis of Algorithms

we need Model a real computer

- Simple enough to allow analysis
- Accurate enough to give meaningfull predictions

RAM model



CPU performs operations between ≤ 2 values from memory

Set operations : $+, -, *, \backslash, \sqrt{\ }, \log$

$\lceil\ \rceil, \lfloor\ \rfloor, \ldots$

AND, OR, XOR, ...

i.e. basic operations only

i.e. no sort() op

Cost of any operation : 1

- unrealistic : $\sqrt{\ }$ costs much more AND but Accurate enough

Given the input size n, count the number of basic operations $T(n)$

Insertion Sort (A)	cost	times
for j = 2 to A.length	c_1	n
key = A[j]	c_2	$n-1$
// insert key in already sorted A[1, j-1]		
i = j - 1	c_3	$n-1$
while i > 0 and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
i = i - 1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
A[i+1] = key	c_7	$n-1$

$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) + c_7 (n-1)$$

too complicated !!!

Assumption $\forall i$: $c_1 = c_2 = \dots = c_7 = c$

$$T(n) = cn + 3c(n-1) + c \sum_{j=2}^{n} t_j + 2c \sum_{j=2}^{n} (t_j - 1)$$

Best case: Already sorted

$$\forall_j \; t_j = 1$$

$$T(n) = cn + 3c(n-1) + cn$$
$$\simeq 5cn$$ LINEAR TIME

Worst case: Sorted in reverse order

$$\forall_{j}, \; t_j = j$$

$$T(n) = cn + 3c(n-1) + c\sum_{j=2}^{n} j$$
$$+ 2c\sum_{j=2}^{n}(j-1)$$

$$\sum_{j=2}^{n} j = \frac{(n-1)n}{2} - 1$$ GAUSS SUM

$$T(n) \simeq \frac{3c(n+1)n}{2} + 4cn$$

DOMINANT TERM

QUADRATIC TIME

Order of growth

No need to be so precise

Idea: Forget about constant factors
and lower order terms

$$\cancel{3}c\,\frac{(n+\cancel{1})n}{\cancel{2}} + \cancel{4c}n \Rightarrow \Theta(n^2)$$

For large enough input size n,
a $\Theta(n)$ algorithm is better than
a $\Theta(n^2)$ algorithm

$$T'(n) = \cancel{1.000.000}\,n + 10 \Rightarrow \Theta(n)$$

$$T''(n) = \frac{n^2}{\cancel{1000}} + n \Rightarrow \Theta(n^2)$$

Important: sequential vs nested parallel loops

for i : 1 to n : // $\Theta(n)$
    s += 1

for j : 1 to n : // $\Theta(n)$
    s += 1

||

$\Theta(n)$

for i : 1 to n :
    s += 1
    for j : 1 to n :
        s += 1

||

$\Theta(n^2)$

for i : 1 to n :
    s += 1
    for j : ~~1~~ i to n :
        s += 1

$$\sum_{i=1}^{n} (n-i) \approx \sum_{i=1}^{n} i$$

$$= \Theta(n^2)$$

for i : 1 to n :
    s += 1
    for j : 1 to $\frac{n}{4}$:
        s += 1

$\Theta(n^2)$

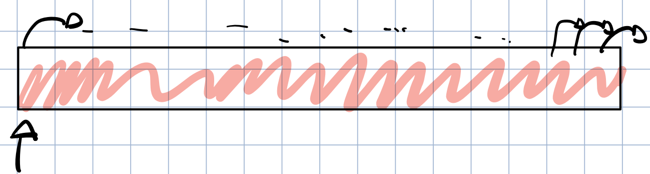
```
A = [10] * n
sum = 0
for e in A:
    sum += e
```

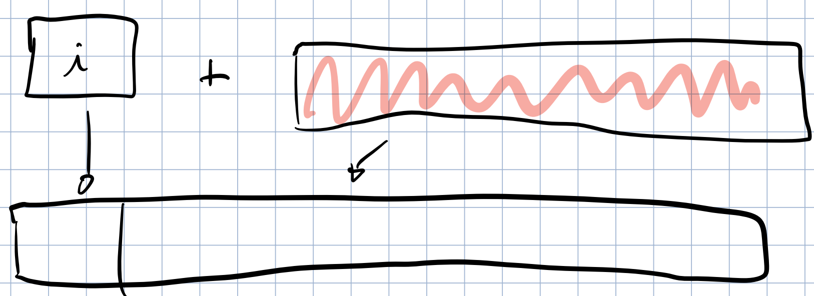
$\Theta(n)$

~~for i in range(len(A)):~~
~~sum += A[i]~~

```
A = []
for i in range(n):
    A.insert(i, 0)
```

$\cancel{\Theta(n)}$ $\Theta(n^2)$





Selection Sort

```
SelectionSort(A)
    for i: 1 to A.length
        min_pos = i
        for j : i+1 to A.length
            if A[j] < A[min_pos]:
                min_pos = j
        swap( A[i], A[min_pos])
```

$\Theta(n^2)$

no difference between Best and Worst cases

From analysis IS and SS are equivalent in the worst case

but
- IS best case is $\Theta(n)$
- SS best case is $\Theta(n^2)$