Last Class

- Proof of Correctness
- Analysis of Algorithms.

Problem: Find the maximum element of an array of integers.

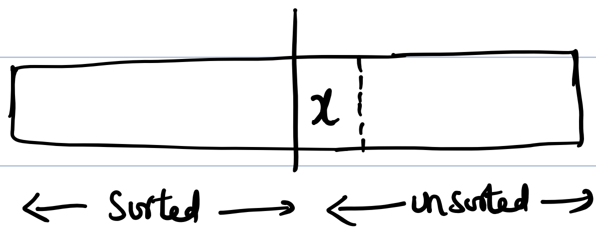
Pseudo Code $A.length = n$

```
FIND-MAX(A)
    max = A[1]
    for j = 2 to A.length
        if max < A[j]
            max = A[j]
    return max
```

Q: What is the running time?

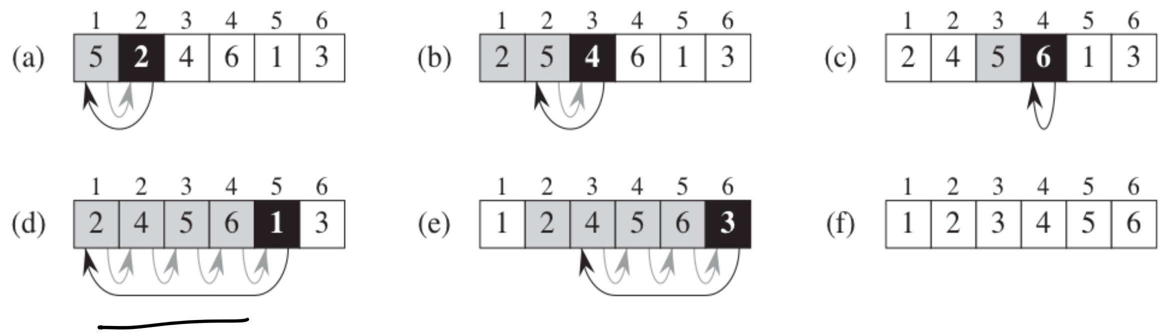
Insertion Sort [Reference: Coreman - Chapter 2]

Given
Array of n elements.



Example

Input array: 5 | 2 | 4 | 6 | 1 | 3



Pseudocode

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence 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
```

Analysis of Insertion Sort

Suppose $A.length = n$

INSERTION-SORT(A)	*cost*	times
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	—
$i = j - 1$	c_4	n $n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

Recall: $T(n)$, the running time of INSERTION-SORT on input of n values.

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

$$+ c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} t_j - 1 + c_8(n-1)$$

The running time depends on the type of input given

For example, say,

"the input is already sorted". [best case*] doesn't mean a sorted input.

then in line 5, $A[i] \leq key$.

So line 5 executes exactly once for each $j = 2, 3, \ldots n$

ie, $t_j = 1$

so $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

$$= \underbrace{(c_1 + c_2 + c_4 + c_5 + c_8)}_{a} n - \underbrace{(c_2 + c_4 + c_5 + c_8)}_{b}$$

$T(n) = an + b$

[Worst Case]

If the array is in reverse sorted order then in line 5, A[j] is compared with each element in the entire sorted subarray A[1, .. j-1].

$\therefore$ $t_j = j$ for $j = 2, .. n$.

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n} (j-1) + c_7 \sum_{j=2}^{n} j-1 + c_8(n-1)$$

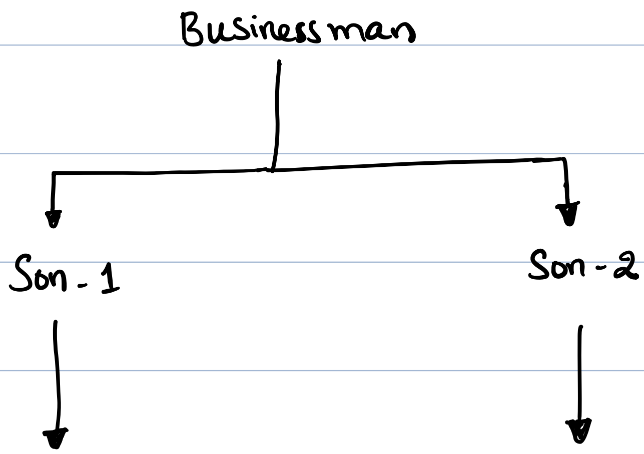
$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = an^2 + bn + c$$

Recall: We usually find worst-case running time, that is the longest running time for any input of size n.

[Don't disclose the answer]

Puzzle



$W(t) = 10^{8}\, t + 400$

$W(t)$ ↓ wealth at time t (in years)

$W(t) = 10^{-6}\, t^{2} + 400$

Q) Who is doing better in their business Son-1 or Son-2?

How Business / wealth is related to Algorithms?

Wealth (maximized)

Wealth as fun of time

Running time (minimized)

Running time as fun of input size.

Order of growth (or) rate of growth

Order of growth of an algorithm means how the Computation time increase when we increase the input size.

We consider only the leading term as lower order terms are insignificant for large size inputs.

We ignore the leading term's Constant Coefficient.

Imp: Asymptotic efficiency of algoritms:

We are only interested in how the running time of an algorithm increases with the size of the input in the limit, as the size of the input increases without bound.

" usually an algoritm that is asymptotically more efficient will be the best Choice for all but very Small i/ps "

We usually consider one algorithm to be more efficient than another if its worst case running time has a lower order of growth.

* Due to constant factors and lower order terms, an algorithm whose running time has a higher order of growth might take less time for small inputs than an algorithm whose running time has a lower order of growth.

Eg: $T(n) = n^2 + 5$, $T(n) = 10^5 n + 8$