Chapter 2 - Getting started [cormen]

- **Question: How can we analyze algorithms?**
  - Behaviour?
  - Time?
  - Etc...

- Lets look at an example...

- **Problem: Sorting problem**

  Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$

  Output: A permutation (reordering) $\langle a_1', a_2', \dots, a_n' \rangle$ of the input sequence such that $a_1' \leq a_2' \leq \dots \leq a_n'$

- **Algorithm: Insertion sort**
  - Idea: Sequentially compare an element against all other elements

1st step:	5	2	4	6	1	3
2nd step:	2	5	4	6	1	3
3rd step:	2	4	5	6	1	3
4th step:	2	4	5	6	1	3
5th step:	1	2	4	5	6	3
6th step:	1	2	3	4	5	6

```
for j = 2 to length[A]
    do key = A[j]
       i = j - 1
       while i > 0 and A[i] > key
           A[i+1] = A[i]
           i = i - 1
       A[i+1] = key
```

Question: How much time does the algorithm require to solve a task?

- Different computers have different running times:
  - CPU
  - RAM
  - Cache
  - Bus
  - Load
  - Etc

LD

- The <u>same</u> computer has <u>different</u> running times
  - Load
  - OS
  - cache
  - Bus
  - Etc...

- We need to represent ~~this or~~ the overall "time" in a manner that is independent of these variables...

- Lets look again at insertion sort:

Cost to execute the instruction on a certain machine

Insertion Sort (A)	Cost	Times
for $j = 2$: length[A]	c_1	n
do key = A[j]	c_2	$n-1$
$i = j-1$	c_3	$n-1$
while $i>0$ and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$ *
do A[i+1] = A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
$i = i-1$	c_6	$\sum_{j=n}^{n} (t_j - 1)$
A[i+1] = key	c_7	$n-1$

* t_j = the number of times the while loop is executed for that value of <u>j</u>

- <u>Question: What is the total running time to sort n elements?</u>

$$T(n) = c_1 n + c_2(n+1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

- The running time depends on:
  - the size of the input: n
  - which input is given:
    - If the array is already sorted there will be intrisicly less operations to execute...
    - If the array is completely reversed there will be a lot of operations to execute...
- The best running time happens when all elements are already sorted:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_3 + c_5 + c_8)n + (c_2 + c_3 + c_4 + c_8)$$

- This running time can be expressed as $an + b$ and is thus a linear function of n

- If the array is in reverse sorted order the worst case result
  - We must compare each element $A[j]$ with each element in the entire subarray $A[1 \ldots j-1]$ and so $t_j = j$ for $j = 2, 3, \ldots, n$

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

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

Notes:

Arithmetic series:

$$\sum_{k=1}^{n} k = \frac{1}{2} n(n+1)$$

$$\frac{1}{2} n(a_1 + a_n)$$

see the included notes of the Arithmetic Series

Arithmetic Series

- Sequence of numbers such that the difference between the consecutive terms is constant

- $a_n = a_1 + (n-1)\,d$
- $a_n = a_m + (n-m)\,d$

a_1 = first term
a_n = last term
a_m = the m^{th} term
d = constant difference

If we use the normal sum then:

- $S_n = a_1 + (a_1 + d) + (a_1 + 2d) + \dots + [a_1 + (n-2)d] + (a_1 + (n-1)d$

If inverted we start summing from the end

- $S_n = (a_n - (n-1)d) + (a_n - (n-2)d) + \dots + (a_n - 2d) + (a_n - d)$

Adding both sides of the equation we get

$$2S_n = n(a_1 + a_n) \quad \left[\begin{array}{l}\text{All terms involving } d \\ \text{are cancelled}\end{array}\right]$$

$$\Leftrightarrow S_n = \frac{n}{2}(a_1 + a_n)$$

$$\Leftrightarrow S_n = \frac{n}{2}\left[a_1 + [a_1 + (n-1)d]\right]$$

$$\Leftrightarrow S_n = \frac{n}{2}\left[2a_1 + (n-1)d\right]$$

But this always assumes that the initial index of the sum is 1.

Question: What happens when $k > 1$?

- Lets look at an example:

$\sum_{j=1}^{n} 2j$ $\quad j = 1\ 2\ 3\ 4\ 5$: $2\ 4\ 6\ 8\ 10$:

$\sum_{j=1}^{n} 2j$ $\quad j = 5\ 4\ 3\ 2\ 1$: $10\ 8\ 6\ 4\ 2$:

$$\begin{array}{rccccc} & 2 & 4 & 6 & 8 & 10 \\ + & 10 & 8 & 6 & 4 & 2 \\ \hline & 12 & 12 & 12 & 12 & 12 \end{array}$$

Question:
Why $1/2$? Because we are using $\sum_{j=1}^{n} 2j$ and $\sum_{j=1}^{n} 2j$ this we will have double the initial value

↳ $\frac{1}{2}(a_1 + a_n) \cdot n =$

$= \frac{1}{2}(a_1 + a_n)(n - (1 + 1))$

$\Downarrow$

$$\frac{1}{2}(a_1 + a_n)(n - \text{initial Index} + 1)$$

- Lets see if it works:
  - Lets calculate manually: $\sum_{j=1}^{n=5} 2j = 2 + 4 + 6 + 8 + 10 = 30$
  - Now using the derived formula: $\sum_{j=1}^{n=5} 2j = \frac{1}{2}(2 + 2.5)(5 - 1 + 1) = \frac{1}{2} \cdot 12.5 = \frac{25}{2} = 30$
  - It works ✓

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left[\frac{n(n+1)}{2} - 1\right] + c_5\left[\frac{n(n-1)}{2}\right] + c_6\left[\frac{n(n-1)}{2}\right] + c_7(n-1) + c_8(n-1)$$

$$= \left[\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right]n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_6}{2} + c_7 + c_8\right)n - (c_2 + c_3 + c_4 + c_7)$$

- This worst-case running time can be expressed as $an^2 + bn + c$ and is thus a quadratic function of n

Question: With what metric do you think we should be dealing with?
- Best-case?
- Worst-case?

- The worst-case running time of an algorithm is an upper-bound of the running time for any input.
  - the algorithm will never take longer...
  - Allows us to make an educated guess...

2.3 Designing Algorithms

- Insertion sort used what is called an incremental technique
- There is also the divide and conquer approach
  - Recursive in nature...

- The divide-and-conquer paradigm:
  - Divide: the problem into subproblems...
  - Conquer: the subproblems by solving them recursively
  - Combine: the solutions to the subproblems into the solution for the original problem