Analysis of Algorithms

Once an algorithm is given for a problem and decided (somehow) to be correct, an important step is to determine **how much in the way of resources**, such as **time** or **space**, the algorithm will require.

- **Space Complexity** → memory and storage are very cheap nowadays. ×
- **Time Complexity** ✓ Different platforms → different time. Absolute time is hard to measure as it depends on many factors.

Example: moving between university buildings: it depends on who are walking, which way he/she use, etc. time is not good measurement. Number of steps is a better one.

Example:

- Consider the problem of summing $$\sum_{k=1}^{n} k = 1 + 2 + 3 + \ldots + n$$

Come up with an algorithm to solve this problem.

Algorithm A	Algorithm B	Algorithm C

`sum = 0`
`for i = 1 to n`
`    sum = sum + i`
	`sum = 0`
	`for i = 1 to n`
	`{`
	`    for j = 1 to i`
	`        sum = sum + 1`
	`}`
		`sum = n * (n + 1) / 2`

Counting Basic Operations

- A **basic operation** of an algorithm is the most significant contributor to its total time requirement.

	Algorithm A	Algorithm B	Algorithm C
Additions	n	$n(n+1)/2$	1
Multiplications			1
Divisions			1
Total basic operations	n	$(n^2+n)/2$	3

How to calculate the time complexity?

- Measure execution time. × Algorithm for small data size will take small time comparing to a large data.
- Calculate time required for an algorithm in terms of the size of input data. × Does not work as the same algorithm over the same data will not take the same time.

Run summing code 2 times and compare time

- Determine order of **growth** of an algorithm with respect to the size of input data. ✓

Order of time or **growth of time:**

Go back to summing result

Linear growth

Quadratic growth

Constant growth

n,	A,	B,	C
1)	7183	7183,	820
10)	2052	4105,	102
100) ,	7183,	155974,	1026
1000) ,	66700,	2983004,	3079
10000) ,	411484,	149256917,	2052
100000) ,	1903500,	13209223813,	1027

In term of **time complexity**, we say that algorithm **C** is better than **A** and **B**

Types of Time Complexity

- Best case analysis — ✗ too optimistic
- Average case analysis — ✗ too complex (statistical methods)
- Worst case analysis — ✓ it will not exceed this

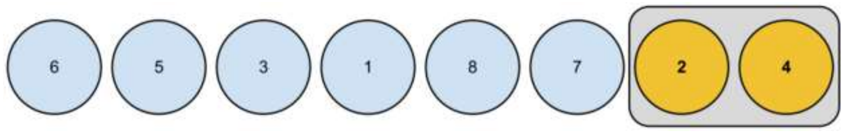
RAM model of computation

We assume that:

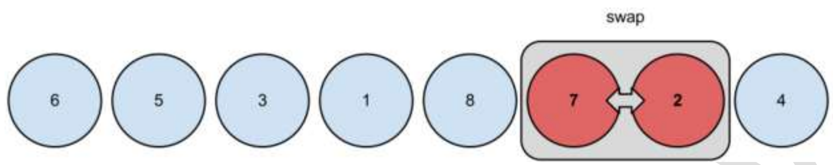
- We have infinite memory
- Each operation (+, -, *, /, =) takes 1 unit of time
- Each memory access takes 1 unit of time
- All data is in the RAM

Bubble Sort:

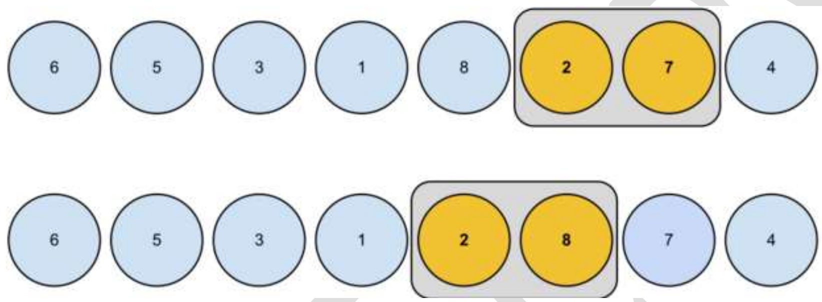
1. Each two adjacent elements are compared:



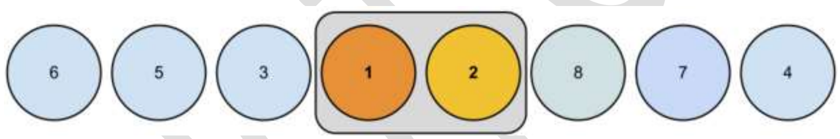
2. Swap with larger elements:



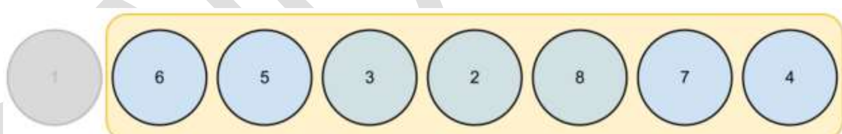
3. Move forward and swap with each larger item:



4. If there is a lighter element, then this item begins to bubble to the surface:



5. Finally the smallest element is on its place:



Make a demo using the following data set

12	8	7	5	2

Worst case analysis

After 1st round:

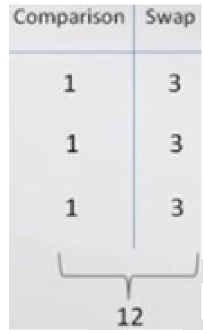
8	7	5	2	12

Comparison	Swap
1	3
1	3
1	3
1	3

16

After 2nd round:

7	5	2	8	12



For whole sorting algorithm: **16+12+8+4** for a data size of 5 elements:

= 4 (4 + 3 + 2 + 1) = 4 (n-1 + n-2 + …. + 2 + 1) = 4 (n-1*n/2) =

2 * n * (n-1) ➔ **$pn^2 + qn + r$** → **p**, **q**, and **r** are some constant.

Implement and test effectiveness of bubble sort algorithm

for (**int** i = 0; i < arr.**length**-1; i++) {
   for (**int** j = 0; j <arr.**length**-i-1 ; j++) {
      if(arr[j+1]<arr[j]){
         temp = arr[j];
         arr[j] = arr[j+1];
         arr[j+1] = temp;
      }
   }
}
	i=0
	i=1
	:
	:
	i=n-1
		j=n-1
		j=n-2
		:
		:
		j=0
			n-1
			n-2
			:
			:
			1