Tutorial 1

Analysis and Design of Algorithms

What is an Algorithm?

An algorithm is a sequence of steps to transform an **input** into an **output**.

Example: Sorting Problem :-

Input	→	Output
A sequence of n numbers <a1,a2,…,an>		A permutation <a1’,a2’,…,an’> such that a1’≤a2’ ≤… ≤an’

An Algorithm should be...

Efficient

The algorithm's resource consumption (or computational cost) is at or below an acceptable level

Correct

For every input instance, the algorithm halts with the correct output

Efficiency: Time Complexity

- Time complexity estimates the time needed to run an algorithm with respect to the input size.
- Our technique must be machine-independent
- Using the RAM model we assume the following properties:
  - Operations such as: {+,/,-,*,if} are considered simple operations and take one step
  - Loops and subroutines consist of simple operations
  - Memory is irrelevant (considered unlimited and takes one step to access whether it is cache or disk)

We measure time complexity by taking into consideration:

Best Case: minimum number of steps to run algorithm

Worst Case: maximum number of steps to run algorithm

Average Case: average number of steps to run algorithm

Exercise 1-3 From CLRS (©MIT Press 2001)

Consider the **searching problem**:
Input: A sequence of n numbers $A[a_1, a_2, ..., a_n]$ and a value v.
Output: An index i such that $v = A[i]$, or the special value NIL if v does not appear in A.

Linear Search

ii. Analyze the best and worst-time complexity for linear search.

```
1: function LinearSearch(A,v)
2:     for i ← 1, A.length do
3:         if A[i] == v then
4:             return i
5:         end if
6:     end for
7:     return NIL
8: end function
```

T(n) Best:

Line	Cost	Times
2		
3		
4		
5		
6		
7		

T(n) Worst:

Line	Cost	Times
2		
3		
4		
5		
6		
7		

Useful Rules

Quick Math Recap

- **Summation Expansion**

1. $\sum_{i=1}^{n} c = c + c + c + \ldots + c$ (n times) $= cn$, where c is a constant.
2. $\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$.

- **Summation Distributivity**

$$\sum_{i=0}^{n-1} (b + id)\, a^i = b \sum_{i=0}^{n-1} a^i + d \sum_{i=0}^{n-1} i a^i$$

Correctness: Loop Invariants

- A loop invariant is some **condition** that **holds** for **every iteration of a loop**
- Which means the condition must be true during…

1. **Initialization:** before the loop starts
2. **Maintenance:** before each iteration of the loop
3. **Termination:** after the loop terminates

Example: Finding the max :-

```
int max = 0;
for (int i=0; i<A.length; i++){
if (max < A[i]) max = A[i]
}
```

Exercise 1-3 From CLRS (©MIT Press 2001)

Consider the **searching problem**:

Input: A sequence of n numbers $A[a_1, a_2, ..., a_n]$ and a value v.

Output: An index i such that $v = A[i]$, or the special value NIL if v does not appear in A.

iii. Using a loop invariant, prove that your algorithm is correct.

```
function LinearSearch(A,v)
    for i ← 1, A.length do
        if A[i] == v then
            return i
        end if
    end for
    return NIL
end function
```

Exercise 1-6

Consider sorting n numbers in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then, finding the second smallest element of A, and exchanging it with $A[2]$. Continue in this manner for the first $n-1$ elements of A. Write pseudo code for this algorithm, which is known as **selection sort**.

i. Give the best-case and worst-case running times of selection sort.

ii. Why does the algorithm need to run for only the first $n-1$ elements, rather than for all n elements?

iii. Prove that selection sort is correct.

Selection Sort

```
1:  function SelectionSort(A)
2:      n ← A.length
3:      for i ← 1, n − 1 do
4:          minIndex ← i
5:          for j ← i + 1, n do
6:              if A[j] < A[minIndex] then
7:                  minIndex ← j
8:              end if
9:          end for
10:         Exchange A[i] ↔ A[minIndex]
11:     end for
12: end function
```

T(n) Best:

Line	Cost	Times
2		
3		
4		
5		
6		
7		
10		
11		

T(n) Worst:

Line	Cost	Times
2		
3		
4		
5		
6		
7		
10		
11		

Exercise 1-7

Consider the following pseudo code for the algorithm Gnome Sort:

```
function GnomeSort(Array A)
    i ← 1
    while i ≤ n do
        if (i == 1) or (A[i − 1] ≤ A[i]) then
            i++
        else
            Exchange A[i] ↔ A[i − 1]
            i−−
        end if
    end while
end function
```

i. Provide an example for each of best and worst-case inputs.

ii. Analyze the best and worst-time complexity of gnome sort.

iii. Prove that the algorithm is correct.

Gnome Sort

```
1: function GnomeSort(Array A)
2:     i ← 1
3:     while i ≤ n do
4:         if (i == 1) or (A[i − 1] ≤ A[i]) then
5:             i++
6:         else
7:             Exchange A[i] ↔ A[i − 1]
8:             i−−
9:         end if
10:    end while
11: end function
```

T(n) Best:

Line	Cost	Times
2		
3		
4		
5		
7		
8		

T(n) Worst:

Line	Cost	Times
2		
3		
4		
5		
7		
8		

All done!