Seminar 1 – Complexity (Algorithm Analysis)

1. TRUE or FALSE?
   a. $n^2 \in O(n^3)$
   b. $n^3 \in O(n^2)$
   c. $2^{n+1} \in \Theta(2^n)$
   d. $2^{2n} \in \Theta(2^n)$
   e. $n^2 \in \Theta(n^3)$
   f. $2^n \in O(n!)$
   g. $\log_{10} n \in \Theta(\log_2 n)$
   h. $O(n) + \Theta(n^2) = \Theta(n^2)$
   i. $\Theta(n) + O(n^2) = O(n^2)$
   j. $O(n) + O(n^2) = O(n^2)$
   k. $O(f) + O(g) = O(\max\{f,g\})$
   l. $O(n) + \Theta(n) = O(n)$
   m. $(n + m)^2 \in O(n^2 + m^2)$
   n. $3^n \in O(2^n)$
   o. $\log_2 3^n \in O(\log_2 2^n)$

2. Complexity of search and sorting algorithms

Algorithm	Time Complexity				Extra Space Complexity
	Best C.	Worst C.	Average C.	Total	
Linear Search	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$O(n)$	$\Theta(1)$
Binary Search	$\Theta(1)$	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$	$O(\log_2 n)$	$\Theta(1)$
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$ – in place
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(1)$ – in place
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(1)$ – in place
Quick Sort	$\Theta(n \log_2 n)$	$\Theta(n^2)$	$\Theta(n \log_2 n)$	$O(n^2)$	$\Theta(1)$ – in place
Merge Sort	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n)$- out of place

3. Analyze the time complexity of the following two subalgorithms:

```
subalgorithm s1(n) is:
    for i ← 1, n execute
        j ← n
        while j ≠ 0 execute
            j ← ⌊j/2⌋
        end-while
    end-for
```

```
end-subalgorithm


subalgorithm s2(n) is:
      for i ← 1, n execute
            j ← i
            while j ≠ 0 execute
                  j ← ⌈j/2⌉
            end-while
      end-for
end-subalgorithm
```

4. Analyze the time complexity of the following two subalgorithms:

```
subalgorithm s3(x, n, a) is:
      found ← false
      for i ← 1, n execute
            if x_i = a then
                  found ← true
            end-if
      end-for
end-subalgorithm


subalgorithm s4(x, n, a) is:
      found ← false
      while found = false and i < n execute
            if x_i = a then
                  found ← true
            end-if
            i ← i + 1
      end-while
end-subalgoritm
```

5. Analyze the time complexity of the following algorithm (x is an array, with elements x_i <= n):

```
Subalgorithm s5(x, n) is:
      k← 0
      for i ← 1, n execute
            for j ← 1, x_i execute
                  k ← k + x_j
            end-for
      end-for
end-subalgorithm
```

a. if every $x_i > 0$

b. if x_i can be 0

- Does the complexity change if we allow values of 0 in the array?

6. Consider the following problems and find an algorithm (having the required time complexity) to solve them :
    a. Given an arbitrary array with numbers $x_1...x_n$, determine whether there are 2 equal elements in the array. Show that this can be done with $\Theta(n \log_2 n)$ time complexity.
    b. Given an arbitrary array with numbers $x_1...x_n$, determine whether there are two numbers whose sum is *k* (for some given k). Show that this can be done with $\Theta(n \log_2 n)$ time complexity. What happens if *k* is even and *k/2* is in the array (once or multiple times)?
    c. Given an array of distinct integers $x_1...x_n$, ordered ascending, determine whether there is a position such that A[i] = i. Show that this can be done with $O(\log_2 n)$ complexity.

7. Analyze the time complexity of the following algorithm:

```
subalgorithm s6(n) is:
      for i ← 1,n execute
            @elementary operation
      end-for
      i ← 1
      k ← true
      while i <= n – 1 and k execute
            j ← i
            k1 ← true
            while j <= n and k1 execute
                  @ elementary operation (k1 can be modified)
                  j ← j + 1
            end-while
            i ← i + 1
            @elementary operation (k can be modified)
      end-while
end-subalgorithm
```

8. Analyze the time complexity of the following algorithm:

```
subalgorithm p(x,s,d) is:
   if s < d then
         m ← [(s+d)/2]
         for i ← s, d-1, execute
               @elementary operation
         end-for
         for i ← 1,2 execute
               p(x, s, m)
         end-for
   end-if
end-subalgorithm
```

Initial call for the subalgorithm: p(x, 1, n)

9. Analyze the time complexity of the following algorithm:

```
Subalgorithm s7(n) is:
      s ← 0
      for i ← 1, n^2 execute
            j ← i
            while j ≠ 0 execute
                  s ← s + j
                  j ← j – 1
            end-while
      end-for
end-subalgorithm
```

10. Analyze the time complexity of the following algorithm:

```
Subalgorithm s8(n) is:
      s ← 0
      for i ← 1, n^2 execute
            j ← i
            while j ≠ 0 execute
                  s ← s + j – 10 * [j/10]
                  j ← [j/10]
            end-while
      end-for
end-subalgorithm
```

11. Analyze the time complexity of the following algorithm:

```
subalgorithm operație(n, i) is
      if n > 1 then
            i ← 2 * i
            m ← [n/2]
            operation(m, i-2)
            operation(m, i-1)
            operation(m, i+2)
            operation(m, i+1)
      else
            write i
      end-if
end-subalgorithm
```