Seminar : Complexity (Algorithm Analysis)

1. TRUE or FALSE?

a)	$n^2 \in O(n^3)$	h)	$O(n) + \Theta(n^2) = \Theta(n^2)$
b)	$n^3 \in O(n^2)$	i)	$\Theta(n) + O(n^2) = O(n^2)$
c)	$2^{n+1} \in \Theta(2^n)$	j)	$O(n) + O(n^2) = O(n^2)$
d)	$2^{2n} \in \Theta(2^n)$	k)	$O(f) + O(g) = O(\max\{f,g\})$
e)	$n^2 \in \Theta(n^3)$	l)	$O(n) + \Theta(n) = O(n)$
f)	$2^n \in O(n!)$	m)	$(n+m)^2 \in O(n^2 + m^2)$
g)	$\log_{10} n \in \Theta(\log_2 n)$	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.	W C.	Avg C.	Total	
Linear Search	Θ(1)	Θ(n)	Θ(n)	O(n)	Θ(1)
Binary Search					
Selection Sort					Θ(1) – in place
Insertion Sort					
Bubble Sort					
Quick Sort					
Merge Sort					Θ(n)- out of place

3. Analyze the time complexity of the following 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

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
    i ← 1
    while found = false and i <= n execute
            if x_i = a then
                        found ← true
            end-if
            i ← i + 1
    end-while
end-subalgoritm
```

```
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

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
            k_1 ← true
            while j <= n and k_1 execute
                    @ elementary operation
                            (k_1 can be modified)
                    j ← j + 1
            end-while
            i ← i + 1
            @elementary operation
                    (k can be modified)
        end-while
end-subalgorithm

    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)

    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

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
```

Seminar : Complexity (Algorithm Analysis)

1. TRUE or FALSE?

p) $n^2 \in O(n^3)$	w) $O(n) + \Theta(n^2) = \Theta(n^2)$
q) $n^3 \in O(n^2)$	x) $\Theta(n) + O(n^2) = O(n^2)$
r) $2^{n+1} \in \Theta(2^n)$	y) $O(n) + O(n^2) = O(n^2)$
s) $2^{2n} \in \Theta(2^n)$	z) $O(f) + O(g) = O(\max\{f,g\})$
t) $n^2 \in \Theta(n^3)$	aa) $O(n) + \Theta(n) = O(n)$
u) $2^n \in O(n!)$	bb) $(n + m)^2 \in O(n^2 + m^2)$
v) $\log_{10} n \in \Theta(\log_2 n)$	cc) $3^n \in O(2^n)$
	dd) $\log_2 3^n \in O(\log_2 2^n)$

2. Complexity of search and sorting algorithms

Algorithm	Time Complexity				Extra Space Complexity
	Best C.	W C.	Avg C.	Total	
Linear Search	Θ(1)	Θ(n)	Θ(n)	O(n)	Θ(1)
Binary Search					
Selection Sort					Θ(1) – in place
Insertion Sort					
Bubble Sort					
Quick Sort					
Merge Sort					Θ(n)- out of place

3. Analyze the time complexity of the following 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

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

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

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

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

    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)


    Subalgorithm s7(n) is:
            s ← 0
            for i ← 1, n² execute
                j ← i
                while j ≠ 0 execute
                        s ← s + j
                        j ← j - 1
                end-while
            end-for
    end-subalgorithm


Subalgorithm s8(n) is:
        s ← 0
        for i ← 1, n² execute
                j ← i
                while j ≠ 0 execute
                        s ← s + j - 10 * [j/10]
                        j ← [j/10]
                end-while
        end-for
end-subalgorithm
```

4. 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 Θ (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 Θ (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.