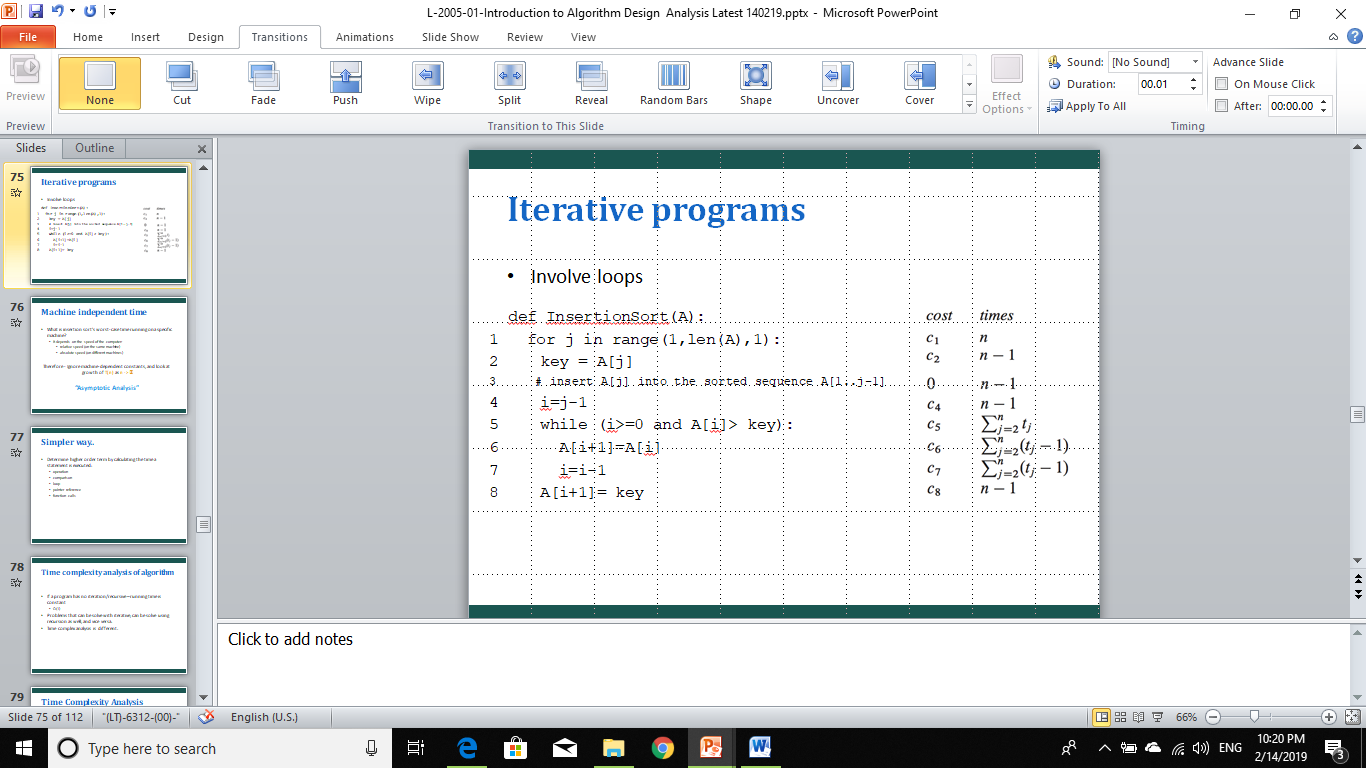
**WIA2005 Algorithm Design &Analysis**

**Semester 2**

**Tutorial 1**

1. The following is an insertion sort algorithm.



Illustrate the insertion sort operation on array A = 41, 51, 69, 36, 51, 68.

41,(51),69,36,51,68

j=1 key=51 i=0 tak lalu while (!41>51) A[i+1]=51

41,51,(69),36,51,68

j=2 key=69 i=1 tak lalu while (!51>69) A[i+1]=69

41,51,69,(36),51,68

j=3 key=36 i=2 lalu while (69>36)

A[i+1]=36

A[i]=69

letak no. besar ke tmpt no. kecik

new A[i+1]=69

i=1

41,51,69,69,51,68 key=36

A[i+1]=69

A[i]=51

letak no. besar ke tempat no. kecik

new A[i+1]=51

i=0

41,51,51,69,51,68 key=36

A[i+1]=51

A[i]=41

letak no. besar ke tempat no. kecik

new A[i+1]=41

i=-1

keluar loop

A[i+1]=36

36,41,51,69,(51),68

j=4 key=51 i=3 lalu while (69>51)

36,41,51,69,(51),68

A[i+1]=51//i+1 will always refer to the key value

A[i]=69

new A[i+1]=69

i=2

keluar dari loop since (51=51)

A[i+1]=51//overwrite 69

36,41,51,51,69,(68)

j=5 key=68 i=4 lalu while (69>68)

36,41,51,51,69,(68)

A[i+1]=68

A[i]=69

new A[i+1]=69//copy A[i] value

i=3

36,41,51,51,69,69 key=68

keluar dari loop since (51<68)//lagi kecik dari the key

A[i+1]=68

36,41,51,51,68,69

1. Modify the insertion sort algorithm to sort array into decreasing order.

def InsertionSort(A):

for j in range(1,len(A),1):

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

return A

print(InsertionSort(A))

1. Write a pseudocode for linear search for the following requirement:

**Input:** A sequence of *n* numbers A = 〈a1, a2, …, an〉 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.

for i=0 to length(A)

if A[i]==v

return i

return NIL

1. Express the function n3 / 1000 – 100n2 – 100n + 3 in terms of Θ-notation.

Θ(e) = Θ(max(n 3 /1000, −100n2 , −100n, 3)).

Θ(1)< Θ(n)< Θ(n2)< Θ(n3)

Then

Θ(e) = Θ(max(n 3 /1000, −100n2 , −100n, 3)) = Θ(n3)

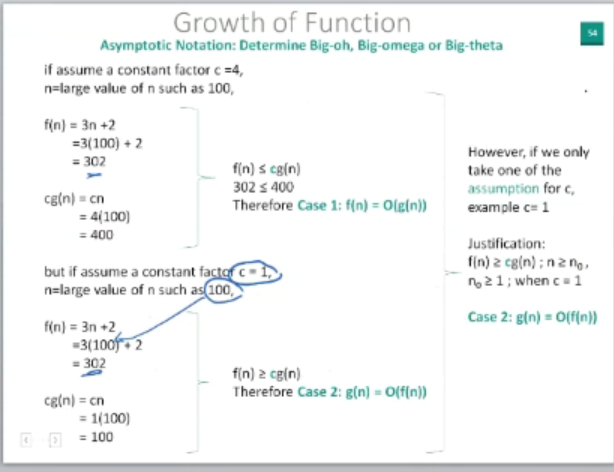
1. For the following pairs of functions, f(n) and g(n), determine if they belong to Case 1: f(n) = O(g(n)) or Case 2: g(n) = O(f(n)). Formally justify your answer.
   1. f(n) = 3n +2 , g(n) = n

Case 2: g(n) = O(f(n))

because f(n) >= cg(n) for n> n0

For example for n=1000

f(n) = 302 while g(n) =100



* 1. f(n) = (n2 – n)/2 , g(n) = 6n

always Case 2: g(n) = O(f(n))

because Of(max(n2,n)) = Of(n2)

Og(n)<Of(n2) for n>n0

* 1. f(n) = n+2√n , g(n) = n2

Case 1: f(n) = O(g(n))

because Of(max(n, √n)) = Of(n)

Og(n2)>Of(n) for n>n0

* 1. f(n) = n2 + 3n + 4 , g(n) = n3

Case 1: f(n) = O(g(n))

because Of(max(n2, n)) = Of(n2)

Og(n3)>Of(n2) for n>n0

#note kalau power both eq sama then baru check c

1. Given the iterative function below (in Java), calculate their time complexity.
   1. function1 (){

for (int i = 1; i <= n; i ++) {

printf(“Hello world”);

}

}

T(n) = O(n)

* 1. function2(){

for (int i = 1; i <=n; i ++) {

for (int j = 1; j <=n; j ++) {

printf(“Hello world”);

}

}

T(n) = O(n2)

* 1. function3 (){

for (int i = 1; i2 <= n; i ++) {

printf(“Hello world”);

}

}

T(n) = O(n0.5)

* 1. function4 (){

for (int i = 1; i <= n; i = i\*2) {

printf(“Hello world”);

}

}

T(n) = O(log2(n))

* 1. function3(){

for (int i = n/2; i <=n; i ++) {

for (int j <= 1; j <=n/2; j = 2\*j) {

for (int k = 1; k <= n; k\*2) {

printf(“Hello world”);

}

}

}

}

}

T(n) = O(n[log2(n)]2)= O(n[log22(n)])

