WIA2005 Algorithm Design &Analysis
Semester 2
Tutorial 1

---

1. The following is an insertion sort algorithm.

```
def InsertionSort(A):
  for j in range(1,len(A),1):
    key = A[j]
    # insert A[j] into the sorted sequence A[1..j-1]
    i=j-1
    while (i>=0 and A[i]> key):
      A[i+1]=A[i]
      i=i-1
    A[i+1]= key
```

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

41	51	69	36	51	68
41	51	69	36	51	68
36	41	51	69	51	68
36	41	51	51	69	68
36	41	51	51	68	69

j	i	key	A[i]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
1	0	51	41	41	51	69	36	51	68
2	1	69	51	41	51	69	36	51	68
3	2	36	69	41	51	69	69	51	68
3	1	36	51	41	51	51	69	51	68
3	0	36	41	41	41	51	69	51	68
3	-1	36	-	36	41	51	69	51	68
4	3	51	69	36	41	51	69	69	68
4	2	51	51	36	41	51	51	69	68
5	4	68	69	36	41	51	51	69	69
5	3	68	51	36	41	51	51	68	69

2. 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
```

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

    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.

```
for i in 0 -> last index of A:
    if a[i] equal to v:
        return i
return NIL

for i in range(0,len(A),1):
    if A[i]==v :
        print(i)
        break
    elif A[i]!=v and i==len(A)-1:
        print("NIL")
```

Prompt input array A

Prompt input value v

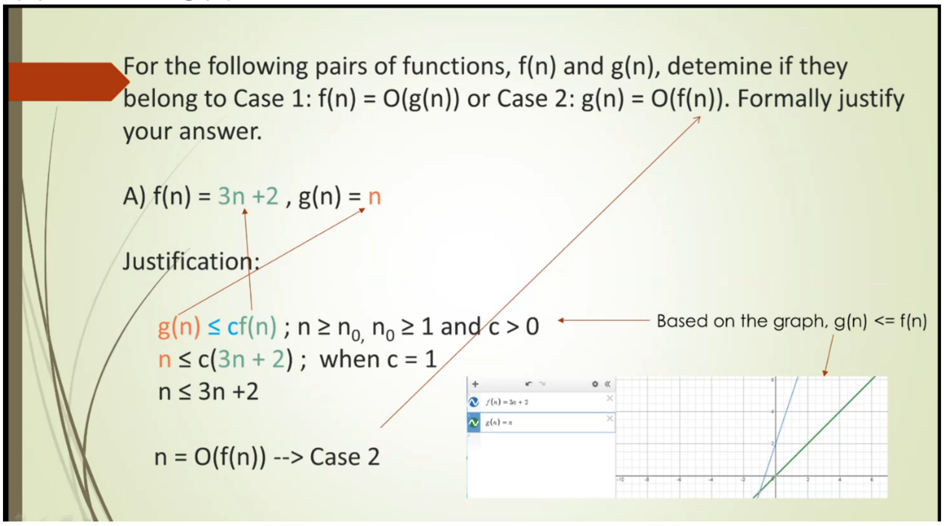
```
for i=0 to length of A
        if A[i] equals v
                return i
return NIL
```

4. Express the function $n^3 / 1000 - 100n^2 - 100n + 3$ in terms of Θ-notation.

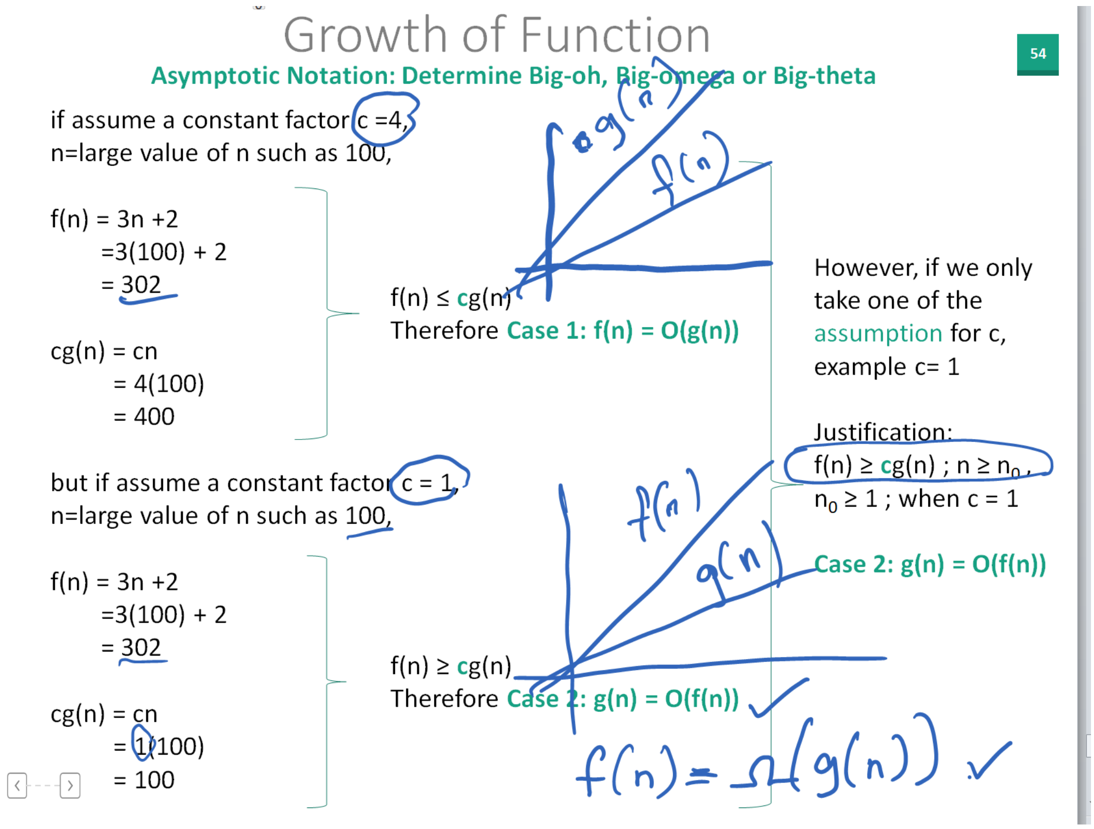
   $\Theta(n^3)$
   Asymptotic complexity only cares about the fastest growing term. This is because it analyses what the function approaches as it tends towards infinity. In this case, n^3

5. 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.
   O(f(n)) represents the set of all functions upper bounded by asymptotic f(n). e.g. O(n) contains all functions upper bounded by asymptotic n. $O(e^n)$ contains all functions upper bounded by asymptotic e^n. It is more formally defined as:
   O(g(n)) = {f(n) | 0 < f(n) <= c*g(n), and c is a constant real number}
   f(n) = O(g(n)) in asymptotic notation represents f(n) ∈ O(g(n))
   https://www.youtube.com/watch?v=whjt_N9uYFI

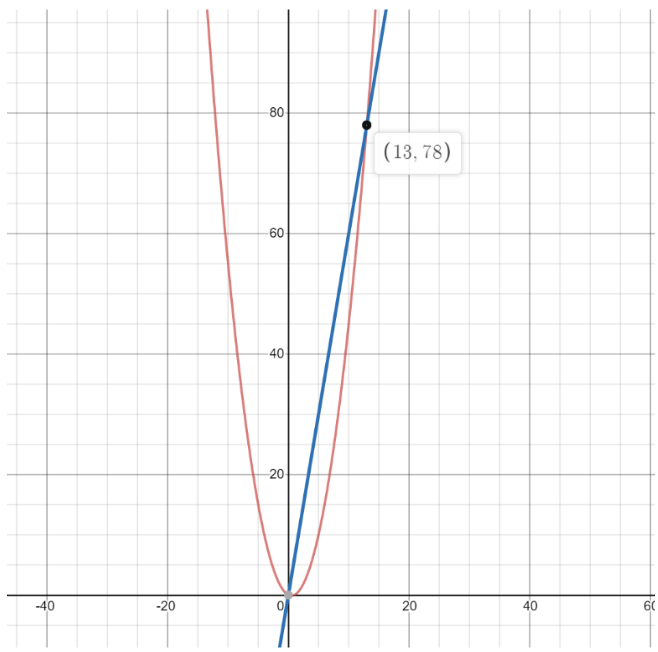
   a. f(n) = 3n +2 , g(n) = n



   Case 2, both functions have same order term but coefficient of f(n) is higher than g(n)



b. $f(n) = (n^2 - n)/2$, $g(n) = 6n$



The graph shows c = 1
(since f(n) intersect with g(n) at 2 points (0,0) and (13,78), for n_o we will take the largest intersection which is (13,78). So given , $n \geq n_o$, $n \geq 1$ and c>0, which in this case n_o =13 and c = 1, we have

proven that g(n) will always be upper bounded by f(n). Hence, it is case 2, g(n) = O(f(n)).)

Proof:
Case 2: g(n) = O(f(n))
$f(n) \geq cg(n)$, $n \geq no$, $n \geq 1$ when c = 1

Case 1: False, Case 2: True
f is in $O(n^2)$ while g is in O(n).
Functions that grow quadratically are not in the set O(n).
Case 1 is thus false
$O(n^2)$ is a superset of O(n). g(n) is in the set O(n).
It must also be in $O(n^2)$
Case 2 is true

6n <= $c(n^2 – n)/2$
12n <= $c(n^2 – n)$
12 <= c(n-1)
12 <= cn-c
c >= 12/(n-1)
As n approaches infinity. right term approaches 0
c will always win at some point
n != 1
n > 1

6cn <= $(n^2 – n)/2$
12cn <= n^2 - n
12c <= n -1
c <= (n-1)/12
As n approaches infinity. right term approaches infinity
c can never win. sad nia
n >= 1

f(n) = $(n^2 – n)/2$, g(n) = 6n
Assume n = 500, c = 3
f(n) = (500^2 - 500) / 2 > 3 * 6 (500)
Case 2

f(n) = 124750 , g(n) = 3000
Assume n = 500, c = 5
12470 <= (5)(3000)
Case 1

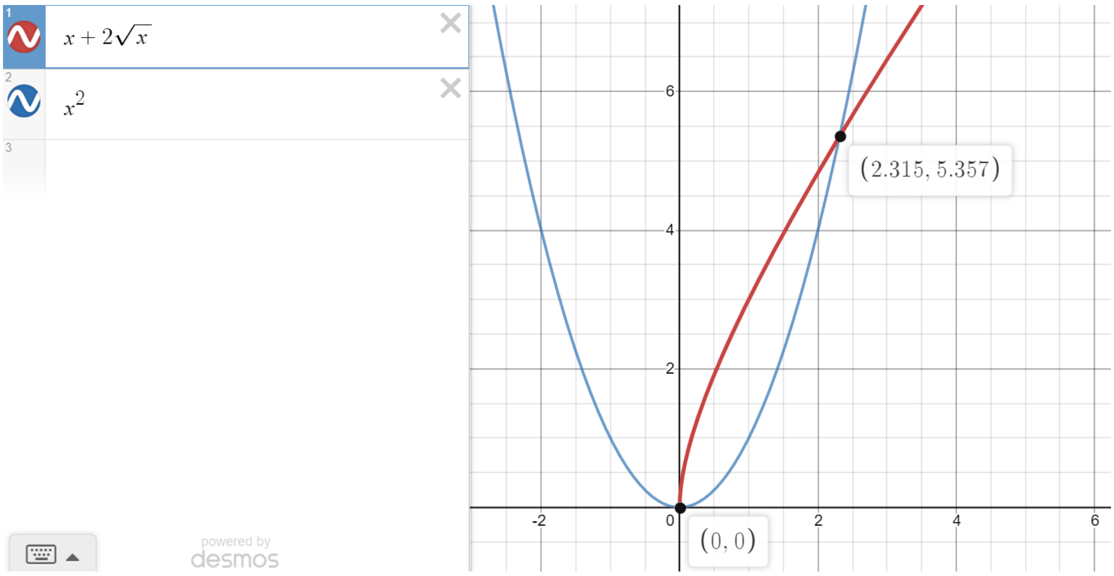
Case 2, because f(n) has higher order term compared to g(n)

c. $f(n) = n+2\sqrt{n}$, $g(n) = n^2$
Case 1: True, Case 2: False
f is in $O(n)$ while g is in $O(n^2)$
This is a reflection case of 5(b)

Case 1, because g(n) has higher order term compared to f(n)

let x be n

$$\lim_{x\to\infty}\left(\frac{x^2}{x+2\sqrt{x}}\right) = \infty$$, as x tends to infinity,



case 1: $f(n) <= cg(n)$; $n \geq n0$, and $c > 0$
$n+2\sqrt{n}<=cn^2$
let choose C=10
$n+ 2\sqrt{n} <= 10n^2$
so n^2 greater than n,
$f(n)=O(g(n))$
so case1 true

case 2: $c f(n) >= g(n)$; $n \geq n0$, and $c > 0$
$c(n+2\sqrt{n})>=n^2$
let choose C=10
$10n+ 20\sqrt{n} >= 10n^2$

so n^2 greater than n, case 2 is false

Case 1: $f(n) = O(g(n))$
$f(n) \leq cg(n)$, $n \geq no$, $n \geq 1$ when $c = 3$

d. $f(n) = n^2 + 3n + 4$, $g(n) = n^3$
Case 1: True, Case 2: False
f is in $O(n^2)$ while g is in $O(n^3)$
$O(n^3)$ is a superset of $O(n^2)$. $g(n)$ is in the set $O(n^2)$.
It must also be in $O(n^3)$
Case 1 is true
Functions that grow cubically are not in the set $O(n^2)$.
Case 2 is thus false

Case 1, because g(n) has higher order term compared to f(n)

case 1 :

$f(n) <= cg(n)$; $n \geq n0$, and $c > 0$
$n^2 +3n +4 <= c\, n^3$
let choose C=10
$n^2 +3n +4 <= 10n^3$
so n^3 greater than n^2,
$f(n)=O(g(n))$
so case 1 true

case 2:

$c\, f(n) >= g(n)$; $n \geq n0$, and $c > 0$
$c(n^2 +3n +4) >= n^3$
let choose C=10
$10n^2 +30n +40 >= 10n^3$
so n^3 greater than n^2,
so case 2 is false

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

$f(n) \leq cg(n)$, $n \geq no$, $n \geq 1$ when $c = 8$

6. Given the iterative function below (in Java), calculate their time complexity.

a. function1 (){

```
        for (int i = 1; i <= n; i ++) {
              printf("Hello world");
        }
   }
```

$O(n)$

b. function2(){

```
        for (int i = 1; i <=n; i ++) {
                    for (int j = 1; j <=n; j ++) {
                         printf("Hello world");
         }
     }
```

$O(n^2)$

n*n

c. function3 (){

```
        for (int i = 1; i² <= n; i ++) {
              printf("Hello world");
        }
   }
```

$O(\sqrt{n}) = O(n^{0.5})$

you can also move the square term to the other side and get $i <= \sqrt{n}$.

My another proving style

K	i	Generalise K
1	1	$\sqrt{1}$
2	4	$\sqrt{4}$
3	9	$\sqrt{9}$
4	16	$\sqrt{16}$
5	25	$\sqrt{25}$
6	36	$\sqrt{36}$
k	n	$\sqrt{n}$

Let said n = 36

When i = 6, n = 36

K = $\sqrt{n}$

Answer: $O(\sqrt{n})$

d. function4 (){

```
        for (int i = 1; i <= n; i = i*2) {
             printf("Hello world");
        }
}
```

O(logn)

the loop actually doing 2^c

where c is the number of iteration

$2^c <= n$

$c <= \log_2 n$

e. 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 =k*2) {

                       printf("Hello world");

                       }

                  }
```

```
        }
    }
}
```

$O(n\log^2 n) = O(n[\log n]^2)$