**LAB 01: Asymptotic notation of complexity**

1. **Identify the efficiency class (big-Theta) of the following algorithms [in the worst cases]**

1. def function(A)

# input: A – an array of n numbers

1. for i←0 to n-2 do

2. for j←i+1 to n-1 do

3. if Ai==A[j] return false

4. return true

Basic operation: Comparison on line 3

The worst cases:

1. All elements are distinct
2. 2 equal elements are located at the end of the array

i = 0, inner loop runs n-1 times

i = 1, inner loop runs n-2 times

…

i = n-2, inner loop runs 1 time

---------------------------------------

C(n) = 1 + …+ n -2 + n - 1

Applying Summation formulas (slide 25, ch02n), we have:

C(n) = (n-1)(n-1+1)/2 ≃n^2/2=(½) \* (n^2)

C(n) ∈Θn2

### def Find (A[1 : n])

#input: A is an array of n numbers

1. If n = 1, then return A[1].

2. Else,

3. let m = Find(A[1 : n − 1]).

4. If m > A[n], then return m

5. Else return A[n].

Basic operation is comparison on line 4

The Worst case is average cases

C(n) = C(n-1) + 1

C(0) = 0

------

C(n) = C(n-1) + 1 = C(n-2 ) + 1 + 1 = … = C(0) + 1 + 1 + …+ 1

= 0 + 1 \* n = n

C(n) in big-Theta(n)

### def f(A):

1. # input A is an array of n number
2. n = len(A)
3. for i in range(0, n-1):
4. max = A[i]
5. imax = i
6. for j in range(i+1, n):
7. if A[j] > max:
8. max = A[j]
9. imax = j
10. A[i], A[imax] = A[imax], A[i]

### Function F(n):

#Input: k integer, positive

#Output: count

1. count ← 0; i = n
2. while (i ≥ 1)
   1. for j ← 1 to n do
      1. count ← count + 1
      2. print(j)
   2. end for
   3. i ← i/3

end while

### ALGORITHMS F()

1. for to of the

2. for to of the

3. if return false

4. return true

### Function f(k):

1. int i, even;
2. i := 1;
3. even := 0;
4. while( i < k ) {
5. even := even + 2;
6. i := i + 1;
7. }
8. return even .

### Algorithm: 1Even(positive integer k)

2Input: k , a positive integer

3Output: k-th even natural number (the first even being 0)

4if k = 1, then

5 return 0;

6else

7 return Even(k-1) + 2 .

Basic op: addition on line 7

Worst case: other cases

T(k) = T(k-1) + 2 if k > 1

= 0 if k = 1

T(k) = T(k-1) + 2

T(k-1) = T(k-2) + 2

…

T(2) = T(1) + 2 = 2

T(1) = 0

T(3) = T(2) + 2 = 4

T(4) = T(3) + 2 = 6

T(k) = 2 + 2 + …. + 0 = 2(k-1) => T(k) in Theta(k)

### Algorithm: Power\_of\_2(natural number k)

2Input: k , a natural number

3Output: k-th power of 2

4if k = 0, then

5return 1;

6else

7return 2\*Power\_of\_2(k – 1).

Basic OP: multiplication on line 7

Worst case: Other cases

T(k) = 2\*T(k-1) if k > 0

= 1 if k = 0

T(k) = 2T(k-1)

T(k-1) = 2T(k-2)

…

T(1) = 2T(0) = 2

T(0) = 1

T(k) = 2\*2...\*1

T(k) = 2^k => T(k) in Theta(2^k)

* + 1. Algorithm: f(natural number k)

1. if k = 0, then
2. return 1;
3. else
4. return k\*f(k - 1)

Basic OP: multiplication on line 4

Worst case: Other cases

T(k) = k\*T(k-1) if k > 0

= 1 if k = 0

T(k) = kT(k-1)

T(k-1) = kT(k-2)

…

T(1) = 1T(0) = 1

T(0) = 1

T(k) = 1\*2...\*k

T(k) = k! => T(k) in Theta(k!)

### Algorithm: f(natural number k)

1. if k = 0, then
2. return 1;
3. else
4. return k\*f(k - 1)

### Algorithm: f(natural number k)

1. int i, power;
2. i := 0;
3. power := 1;
4. while( i < k ) {
5. power := power \* 2;
6. i := i + 1;
7. }
8. return power .

Basic OP: Multiplication on line 5

Worst case: other cases

T(k) = T(k-1)\*2

T(k-1) = T(k-2)\*2

….

T(1) = 2T(0)

T(0) = 1

T(k) = 2\*2\*...\*1

T(k) = 2^k

=> T(k) in Theta(2^k)

### Algorithm: f(A[0..n-1], K)

1. i ← 0
2. while i < n and A[i] ≠ K do
3. i ← i + 1
4. if(i < n)
5. return i
6. else
7. return -1

Basic OP: comparison on line 2

Worst case: K is the last element in the array Or K is not in the array.

Count:

T(n) = n

T(n) in Theta(n)

### Algorithm: 1f(n)

2if (n == 1)

3 return 1;

4return n \* f(n-1);

Basic OP: multi on line 4

Worst case: other cases

Count:

T(n) = T(n-1)\*n

T(n-1) = T(n-2)\*(n-1)

…

T(2) = 2T(1) = 2

T(1) = 1

T(n) = n! => T(n) in Theta(n!)

### Algorithm bubbleSort(A : list of sortable items)

1 . n := length(A)

2 . repeat

3 . swapped := false

4 . for i := 1 to n-1 inclusive do

5 . /\* if this pair is out of order \*/

6 . if A[i-1] > A[i] then

7 . swap(A[i-1], A[i])

8 . swapped := true

9. end if

10. end for

11. until not swapped

12.end procedure

### Basic OP: Comparision on line 6

Worst case: Array in descending order

Count:

T(n) = (n-1)!

=> T(n) in Oh(n-1)!

### ALGORITHMS func()

1. for to do

2. for to do

3. if return false

4. return true

Basic OP: comparison on line 3

Worst case: Last two elements in the array are equal

Count:

I = 0 : inner loop runs n - 1 times

I = 1: inner loops runs n – 1 times

….

T(n) = n\*(n-1)/2 => T(n) in BigOh(n^2)

### function insertionSort(array A)

01. i ← 1

02. while i < length(A)

03. x ← A[i]

04. j ← i - 1

05. while j >= 0 and A[j] > x

06. A[j+1] ← A[j]

07. j ← j - 1

08. end while

09. A[j+1] ← x

10. i ← i + 1

11.end while

Basic OP: Comparison on line 5

Worst case: Array in descending order

Count:

T(n) ~= n!

=> T(n) in Oh(n)!

## Let's tell if the following statements are true or false. If a statement is true, prove it using definitions of asymptotic notations

* 1. ?
  2. ?
  3. If and then

(Hint: the statement is false; you may prove it with a counterexample)

3. Given two functions , let's tell if the following statements are true or false. If a statement is true, prove it using definitions of asymptotic notations

4. Prove the following conclusions:

B. ,

C.

D.

5. Arrange the following functions in ascending order of growth:

6. Arranges the following functions in ascending order of growth: