2.1.23 * Euclid's algorithm
* Consecutive integer checking algorithm
* Middle school algorithm

Euclid's algorithm:

Example: Find GCD among two numbers.

GCD(m,n) = GCD(n, m mod n) untill n=0.

iteration algorithm:

```
while (n != 0)
  {
    r ← m mod n   or   m - n
    m ← n
    n ← r
  }
```

m = 13, n = 12

step 1:

n != 0 ⇒ yes

r ← 13 mod 12 = 1

m ← 12

n ← 1

step 2:

n != 0 ⇒ yes

r ← 12 % 1 = 0

m ← 1

n ← 0

recursive algorithm:

```
int GCD(m,n) {
  if (n==0) {
    return m;
  }
  else {
    return GCD(n, m mod n)
  }
```

3.2.2023

GCD (a,b) 10>5
if a==b 5 5
return a return 5
if a>b
return GCD(a-b, b)
else
return GCD (a, b-a)

① 46==12 false ③ 22 > 12 (10, 12)
46>12 (34, 12) ④ 10<12 (10, 2)
② 34>12 (22, 12) ⑤ 10>2 (8, 2)
⑥ 8>2 (6, 2)
⑦ 6>2 (4, 2)
⑧ 4>2 (2, 2) a==b return 2.

Consecutive Integer checking algorithm:

Step 1: Consecutive integer checking algorithm for GCD (m, n)
: assign the value of min {m, n}
Step 2: Divide m by t.
if the remainder of 0 go to step 3.
otherwise go to step 4.
Step 3: Divide n by t. If remainder is 0 return the value of t as the answer and stop otherwise proceed to Step 4.
Step 4: Decrease the value of t by 1
Go to Step 2.

t = min (m, n) return t; }
if (m%t==0){ }
if (n%t==0){ else t=t-1;

m = 10, n = 8
① t = min (10, 8)
= 8
m/t = 10%8 ≠0
t = t-1 = 7

② m/t = 10%7 ≠0
t = 6

③ m/t = 10%6 ≠0
t = 5

④ m/t = 10%5 == 0
n/t = 8%5 ≠0
t = 4

⑤ m/t = 10%4 ≠0
t = 3

⑥ m/t = 10%3 ≠0
t = 2

⑦ m/t = 10%2 == 0
n/t = 8%2 == 0
return 2

if n=0, return m

Middle School procedure:
Step 1: Find the prime factors of m.
Step 2: find the prime factors of n.
Step 3: Identify all the common factors in the 2 prime expansions found in step 1 and step 2. (If p is a common factor occuring in pm and pn times in m and n respectively it should be repeated min {pm, pn} times).
Step 4: Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

m = 48, n = 12
factors of 48 = 1, ②, ③, 4, 6, 8, 12, 16, 24, 48
factors of 12 = 1, ②, ③, 4, 6, 12
48 = ②×②×2×2×③
12 = ②×②×③

GCD (48, 12) = 2×2×3 = 12

4.2.2023 Sieve

for P ← 2 to n do A[P] ← P
for P ← 2 to ⌊√n⌋ do
  if A[P] ≠ 0
    j ← P * P
    while j ≤ n do
      A[j] ← 0
      j ← j + P
i ← 0
for P ← 2 to n do
  if A[P] ≠ 0
    L[i] ← A[P]
    i ← i + 1
return L

An algorithm is a sequence of unambiguous instructions for solving a problem that is for obtaining a required output for any legitimate input in a finite amount of time

1: The non ambiguity requirement for each step of algorithm can't be compramised.
2: The range of inputs for which algorithm has to be specified carefully.
3 The same algorithm can be represented in several different ways
4: There may exist several algorithms for solving the same problem

5: Algorithm for the same problem can be based on very different ideas and can be solved the problem dramitically different speeds.

Mathematical analysis of non recursive algorithm:

Worst case Time complexity → max times execution
Bestcase Time complexity → minimum times execution
Average case Time complexity → Considers all possible of occurence of max element & depends upon the i/p

```
#include <stdio.h>
int main()
{
 int a[10], i, max, n
 Scanf ("%d", &n);
 for(i=0; i<n; i++)
   scanf ("%d", &a[i]);
 max = a[0];
 for (i=1; i<n', i++)
 {
   if (max < a[i])
     max = a[i];
 }
 return 0;
}
```

	Execute/ not execute	Frequency	Total
	0		0
	0		0
	1		1
	1	n+1	
	1	n	
	1	1	
	1	n	
	0		
	1	n-1	
	1	n-1	
	0		
	1	1	
	0		

f(n) = 5n+1 = O(n)

3.2.23

a[0]	a[1]	a[2]	a[3]	a[4]	
10	7	6	5	4	→ 0 time
10	11	12	13	14	→ n-1 times
10	11	9	8	6	→ 1 time
10	11	12	8	6	→ 2 time
10	11	12	13	6	→ 3 time
10	11	12	13	14	→ 4 times

Avg case

(large program take only basic operation performed)

efficient

Basic operation : comparison

$C(n) = n-1 = O(n)$

∴ Mathematical proof : 5 elements ⇒ 4 comparison

n elements → n-1 comparison

n log n + 1 A(n-1)

Algorithm - Sequential search : Linear search unordered array.

```
Sequential search (A[0 ... n-1], K)
i ← 0
while i<n and A[i] ≠ K do
    i ← i+1
if i<n return i
else return -1
```

Example: n = 5 10 12 18 7 6 K=13 , i = 0

worst case

0 < 5 && A[0] = 10 ≠ 13 i = i+1 = 0+1 = 1

1 < 5 && 12 ≠ 13

2 < 5 && 18 ≠ 13

3 < 5 && 7 ≠ 13

4 < 5 && 6 ≠ 13

$C_{best}(n)$ Best case : O(1) (1st iteration element found)

Worst case : O(n) n (last element), n+1 (not in array)

Average case : Probability of success + Pr. of failure

(Time taken for success) (Time taken for failure)

$$\frac{1}{n} P[1+2+3+\dots+n] + (1-P)(n+1)$$

$$= \frac{P}{n}\left(\frac{n(n+1)}{2}\right) + (1-P)(n+1)$$

$$= \frac{P(n+1)}{2} + (1-P)(n+1)$$

If element present in array probability 1

$$= P \times \frac{(n+1)}{2} + (1-P)(n)$$

if P = 1

$$C_{Avg}(n) = \frac{n+1}{2} = O(n)$$

if P = 0

$$C_{Avg}(n) = (1-P) \times n$$

$$= 1 \times n = O(n)$$

Mathematical Analysis of nonrecursive algorithm:

```
MaxElement (A[0 ... n-1])
    maxval ← A[0]
    for i ← 1 to n-1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

⇒ Basic operation : comparison (A[i] > maxval)

$$C(n) = \sum_{i=1}^{n-1} 1 \qquad \sum_{l}^{u} 1 = u - l + 1$$

$$= n-1-1+1$$

$$= n-1$$

$$\in O(n)$$

```
UniqueElements (A[0 ... n-1])
    for i ← 0 to n-2 do
        for j ← i+1 to n-1 do
            if A[i] == A[j] return false
    return true
```

⇒ Basic operation : comparison

Time complexity : $C_w(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$

$$C_w(n) = \sum_{i=0}^{n-2} n-1-(i+1)+1$$

$= \sum_{i=0}^{n-2} n-1-i$

$= (n-1)+(n-2)+(n-3)+\ldots+(n-1-(n-2))$

$= (n-1)+(n-2)+(n-3)+\ldots+1$

$= \frac{(n-1)(n)}{2}$

$= \frac{n^2-n^2}{2}$

$\in O(n^2)$

Matrix multiplication:

```
MatrixMultiplication(A[0..n-1, 0..n-1],
                     B[0..n-1, 0..n-1])
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        c[i,j] ← 0.0
        for k ← 0 to n-1 do
            c[i,j] ← c[i,j] + A[i,k] * B[k,j]
return C
```

⇒ Basic operation = multiplication

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(n-1+1)$$

$$= \sum_{i=0}^{n-1} n\sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n(n-1+1) = n^2\sum_{j=0}^{n-1} 1$$

$= n^2[n-1+1] = n^3 \in O(n^3)$

```
Binary(n)        (no of binary digit in decimal no)
count ← 1
while n > 1 do
    count ← count + 1
    n ← ⌊n/2⌋
return count
```

⇒ Basic operation = comparison

$C(n) = \frac{n}{2^k} > 1$

$= n < 2^k$

Taking log on both sides

$\log_2 n < \log_2 2^k$

$k = \log_2 n$

$\in O(\log_2 n)$

```
Sec ( arr[0..n-1], len )
    max ← min
    Sec max ← min
    for(i=0; i<len; i++)
        if (arr[i] > max)
            Sec max = max
            max = arr[i]
        if ( max > arr[i] && arr[i] > Sec max )
            Sec max = arr[i]
    return Sec max
```

Consider the following algorithm:

```
* Algorithm Foo(n)
    sum ← 0
    for i ← 1 to n do
        sum ← sum + i/i!
    return sum
```

a) what does this algorithm compute?
b) what is the basic operation?
c) How many times basic operation execute?
d) what is the efficient class of the algorithm?

a) it compute the sum of series $\frac{1}{1!} \frac{2}{2!} \frac{3}{3!} \ldots$

b) addition

c) $A(n) = \sum_{i=1}^{n} 1$

$= n+1-1 = n$

d) Linear time efficiency

* Algorithm Foo(A[0...n-1])
  val ← 100; sumgreater ← 0; sumless ← 0
  for i ← 0 to n-1 do
    if A[i] > val
      Sumgreater ← A[i]
    if A[i] < val
      Sumless ← A[i]
  return sumgreater - sumless

a) difference between maximum element and minimum element in the array

b) Comparison

c) $C(n) = \sum_{i=0}^{n-1} 1 = n-1-0+1 = n$

d) Linear time efficiency

* Algorithm Enigma (A[0...n-1, 0...n-1])
  for i ← 0 to i ← n-2 do
    for j ← i+1 to n-1 do
      if A[i,j] ≠ A[j,i]
        return false
  return true

a) Check all the elements are equal other than diagnal elements

b) Comparison

c) $C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$

$= \sum_{i=0}^{n-2} n-1-(i+1)+1$

$= \sum_{i=0}^{n-2} n-1-i-1+1 \quad (n-1)+(n-2)+\dots+1$

$= \frac{(n-1)(n)}{2} = \frac{n^2-n}{2} \in O(n^2)$

d)

17.2.23 * Algorithm GE(A[0...n-1, 0...n])
  for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
      for k ← i to n do
        A[j,k] ← A[j,k] - A[i,k] * A[j,i] / A[i,i]

a)

b)

c) $= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=i}^{n} 1$

$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (n-i+1)$

$= \sum_{i=0}^{n-2} (n-i+1) \sum_{j=i+1}^{n-1} 1$

$= \sum_{i=0}^{n-2} (n-i+1)(n-1-i-1+1)$

$= \sum_{i=0}^{n-2} (n-i+1)(n-1-i)$

$= \sum_{i=0}^{n-2} (n-i)^2 - (1)^2$

$= \sum_{i=0}^{n-2} (n-i)^2 - \sum_{i=0}^{n-2} 1$

$= \sum_{i=0}^{n-2} (n^2+i^2-2ni) - \sum_{i=0}^{n-2} 1$

$= n^2 \sum_{i=0}^{n-2} 1 + \sum_{i=0}^{n-2} i^2 - 2n \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1$

$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \dots + n^2 \approx \frac{1}{3} n^3 \Big/ \sum_{i=0}^{n-2} i^2 \approx \left(\frac{1}{3} n^3 - (n-1)\right.$

$= (n^2 [n-2-0+1]) + \left(\frac{1}{3} n^3 - 2n + 1\right) - 2n [0+1+2+\dots+n$
$- [n-2-0+1]$

$= [n^3 - n^2] + \left(\frac{1}{3} n^3 - 2n + 1\right) - 2n \frac{[(n-2)(n-1)]}{2}$
$- (n-1)$

$\in O(n^3)$

d) Cubic time

Mathematical analysis of recursive algorithm

Algorithm F(n)
if n=0
when 1
else return n * F(n-1)

Basic operation = Multiplication

Initial condition $M(0) = 0$

Recurrence equation $M(n) = M(n-1) + 1$

(Backward substitution) $= M(n-2) + 1 + 1$
$= M(n-2) + 2$
$= M(n-3) + 1 + 2$
$= M(n-3) + 3$

$= M(n-i) + i$

Substitute $i = n$,
$= M(n-n) + n$
$= M(0) + n \quad = 0 + n$
$\in O(n)$

1.2.2023

Tower of Hanoi:

Basic operation: Move

Recurrence relation

$M(n) = M(n-1) + 1 + M(n-1)$
$= 2M(n-1) + 1$ $\quad M(n-1) = 2M(n-2)+1$
$= 2[2M(n-2)+1] + 1$ $\quad M(n-2) = 2M(n-3)+1$
$= 2^2 M(n-2) + 2 + 1$
$= 2^2 [2M(n-3)+1] + 2 + 1$
$= 2^3 M(n-3) + 2^2 + 2 + 1$
$= 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \dots + 2 + 1$
$= 2^i M(n-i) + 2^i - 1$
$= 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1$

Initial condition $M(1) = 1$

$= 2^{n-1} M(1) + 2^{n-1} - 1$
$= 2^{n-1} \times 1 + 2^{n-1} - 1$
$= 2^{n-1} + 2^{n-1} - 1$
$= 2^1 \times 2^{n-1} - 1$
$= 2^n - 1$
$\in O(2^n)$

$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

$= 1 + 2^1 + 2^2 + \dots + 2^n$
$= 2^{n-1+1} - 1$
$= 2^n - 1$

Sums:

1. $x(n) = x(n-1) + 5$ for $n > 1$ $\quad x(1) = 0$

Sub $n = n-1$ in ① $\quad x(n-1) = x(n-2) + 5$
$n = n-2$ in ② $\quad x(n-2) = x(n-3) + 5$

$x(n) = x(n-1) + 5$
$= x(n-2) + 5 + 5$
$= x(n-3) + 5 + 5 + 5$
$\vdots$
$= x(n-i) + i \times 5$

Sub $i = n-1$ $\quad = x(n-(n-1)) + (n-1) \times 5$
$= x(1) + 5n - 5$
$= 0 + 5n - 5$
$\in O(n)$

2. $x(n) = 3x(n-1)$ for $n>1$ $x(1)=4$

Sub $n=n-1$ $x(n-1) = 3x(n-2)$

$n=n-2$ $x(n-2) = 3x(n-3)$

$x(n) = 3x(n-1)$

$= 3(3x(n-2))$

$= 3^2(x(n-2))$

$= 3^2(3x(n-3)) = 3^3 x(n-3)$

$= 3^i x(n-i)$

$i=n-1$ $= 3^{n-1} x(n-(n-1)) = 3^4 x(1)$

$= 3^{n-1}(4)$

$\in O(3^n)$

3. $x(n) = x(n-1)+n$ for $n>0$ $x(0)=0$

Sub $n=n-1$ $x(n-1) = x(n-2)+n-1$

$n=n-2$ $x(n-2) = (x(n-3)+n-2)+(n-1)$

$x(n) = x(n-1)+n$

$= (x(n-2)+n-1)+n$

$= (x(n-3)+n-2)+n-1+n$

$= x(n-3)+3n-3$

$i=n$ $= x(n-i)+in-i$

$= x(n-n)+n^2-n = 0+n^2-n$

$\in O(n^2)$

7.2.23

Algorithm BinRec(n)

if $n=1$ return 1

else return BinRec($\lfloor n/2 \rfloor$)+1

$n=32$ BinRec($32/2$)+1 ⑥

BinRec($16/2$)+1 ⑤

BinRec($8/2$)+1 ④

BinRec($4/2$)+1 ③

BinRec($2/2$)+1 ②

1+1 ①

O/P = 6 (addition operation)

Initial condition: $A(1)=0$

Basic operation: Addition

Recurrence relation:

$A(n) = A(n/2)+1$ (here didn't use backward substitution because $A(n) \to A(n/2)$ half reduce)

Substitute $n=2^k$ ⇒ Smoothness rule

$A(2^k) = A(2^k/2)+1$

$A(2^k) = A(2^{k-1})+1$ —①

Sub $k=k-1$ in ① $A(2^{k-1}) = A(2^{k-2})+1$

Sub $k=k-2$ in ① $A(2^{k-2}) = A(2^{k-3})+1$

$A(2^k) = (A(2^{k-3})+1)+1$

$= (A(2^{k-3})+1)+1+1$

$= (A(2^{k-3}))+3$

$= A(2^{k-i})+i$ —②

Sub $i=k$ in ②

$A(2^k) = A(2^{k-k})+k$

$= A(2^0)+k$

$= A(1)+k$

$A(2^k) = 0+k = k$

Find k: $n = 2^k$

Take log on both side

$\log n = \log_2 2^k$

$\log n = k \log_2 2$

$\log n = k$

$A(2^k) = k$

$= \log n$

$\in O(\log n)$

Fibonacci Series :

0 1 1 2 3 5 ....

$F(n) = F(n-1) + F(n-2)$ Recurrence relation

$F(0) = 0 \quad F(1) = 1$ Two initial condition

homogeneous second order linear recurrence with constant coefficients.

$ax(n) + bx(n-1) + cx(n-2) = 0$

$F(n) - F(n-1) - F(n-2) = 0$

$a=1, \ b=-1, \ c=-1$

characteristic equation

$r^2 - r - 1 = 0$

$r = \frac{-b \pm \sqrt{b^2-4ac}}{2a} = \frac{1 \pm \sqrt{(-1)^2 - 4(-1)}}{2}$

$r = \frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2}$

$r_1 = \frac{1+\sqrt{5}}{2}, \ r_2 = \frac{1-\sqrt{5}}{2}$

This characteristic eqn has two distinct real roots.

The recurrence relation is,

$F(n) = \alpha r_1^n + \beta r_2^n$

$F(n) = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^n + \beta\left(\frac{1-\sqrt{5}}{2}\right)^n$ — ①

Substitude $n=0$ in ①

$F(0) = \alpha(1) + \beta(1)$

$\alpha + \beta = 0$ — ②

Sub $n=1$ in ①

$F(1) = \alpha\left(\frac{1+\sqrt{5}}{2}\right) + \beta\left(\frac{1-\sqrt{5}}{2}\right)$

$\alpha\left(\frac{1+\sqrt{5}}{2}\right) + \beta\left(\frac{1-\sqrt{5}}{2}\right) = 1$ — ③

$\alpha + \beta = 0$

$\alpha = -\beta$

$-\beta\left(\frac{1+\sqrt{5}}{2}\right) + \beta\left(\frac{1-\sqrt{5}}{2}\right) = 1$

$\beta\left(\frac{-1-\sqrt{5}+1-\sqrt{5}}{2}\right) = 1$

$\beta\left(\frac{-2\sqrt{5}}{2}\right) = 1$

$\beta(-\sqrt{5}) = 1$

$\beta = \frac{1}{-\sqrt{5}}$

$\alpha = -\beta = -\left(\frac{-1}{\sqrt{5}}\right) = \frac{1}{\sqrt{5}}$

① ⇒ $F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$

$= \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$ $\quad \frac{1+\sqrt{5}-1+\sqrt{5}}{2}$

where ⇒ $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803 \quad \hat{\phi} = -\frac{1}{\phi} = \frac{1-\sqrt{5}}{2} \approx -0.61803$

ϕ → golden ratio

explicit formula for n^{th} fibonacci number

$F(n) = \frac{1}{\sqrt{5}}(\phi^n)$ → n^{th} fibonacci number.

Example $F(1) = \frac{1}{\sqrt{5}}(1.61803)^1 = 0.7123$

$F(2) = \frac{1}{\sqrt{5}}(\phi^2) = 1.170 \approx 1$

Recursive algorithm :

```
Algorithm F(n)
    if n ≤ 1
        return n
    else
        return F(n-1) + F(n-2)
```

Basic operation := Addition

Initial condition ⇒ $A(0) = 0, \ A(1) = 0$

Recurrence relation $A(n) = A(n-1) + A(n-2) + 1$

$F(n) - F(n-1) - F(n-2) = 0$

$A(n) - A(n-1) - A(n-2) = 1$

$A(n) - A(n-1) - A(n-2) - 1 = 0$

$A(n) + 1 - A(n-1) - 1 - A(n-2) - 1 = 0$

$(A(n)+1) - (A(n-1)+1) - (A(n-2)+1) = 0$

$B_n = A(n) + 1$ — ①

$B(n) - B(n-1) - B(n-2) = 0$

Sub $n=0$ in ① $B(0) = A(0)+1 = 0+1 = 1$

Sub $n=1$ in ① $B(1) = A(1)+1 = 0+1 = 1$

$B(0) = 1$ $F(0) = 0$

$B(1) = 1$ $F(1) = 1$

① $\Rightarrow$ $B(n) = F(n+1)$

$F(n+1) = A(n) + 1$

$A(n) = F(n+1) - 1$

$= \frac{1}{\sqrt{5}} \phi^{n+1} - 1$

$\in O(\phi^n)$

Non-recursive algorithm:

Algorithm Fib(n)

F[0] ← 0; F[1] ← 1

for i ← 2 to n do

F[i] ← F[i-1] + F[i-2]

return F[n]

$F[i] \leftarrow F[i-1] + F[i-2]$

$i=2$ $F[2] \leftarrow F[1] + F[0] = 1+0 = 1$

$F[3] \leftarrow F[2] + F[1] =$

Basic operation : Addition

Time complexity : $\sum_{i=2}^{n} 1$

$= n-2+1$

$= n-1$

$\in O(n)$

2022 Unit - II

Brute Force Approach:

It's a straight forward approach to solve solving a problem, usually directly based on the problem statement and definitions of the concepts involved.

Algorithm selectionsort (A[0.... n-1])
```
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min]  min ← j
    swap A[i] and A[min]
```

Ex: n=6: 18 20 15 7 21 10 (indices 0 1 2 3 4 5)

Swap	i=0 min=0 ⇌ 3	7	20	15	18	21	10
	i=1 min=1 ⇌ 5	7	10	15	18	21	20
	i=2 min=2	7	10	15	18	21	20
	i=3 min=3	7	10	15	18	21	20
	i=4 min=4 ⇌ 5	7	10	15	18	20	21

Swap operation $S(n) = n-1$ times $\in O(n)$

Basic operation : comparison

Time complexity : $$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} n-1-i-1+1 = \sum_{i=0}^{n-2} n-1-i$$

$$= (n-1) + (n-2) + (n-3) + \dots + (n-1-n+2)$$

$$= (n-1) + (n-2) + (n-3) + \dots + 1$$

$$= \frac{(n-1)(n)}{2} = \frac{n^2-n}{2}$$

$\in O(n^2)$

Algorithm Bubblesort (A[0... n-1])
```
for i ← 0 to n-2 do
    for j ← 0 to n-2-i do
        if A[j+1] < A[j] swap A[j] and
```

Ex : n = 4 17 10 2 5 — pass 1 — 10 17 2 5 — 10 2 17 5 — 10 2 5 17 — pass 2 — 10 2 5 17 — 2 10 5 17 — 2 5 10 17

Basic operation : comparison

Time complexity: $$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1$$

$$= \sum_{i=0}^{n-2} n-2-i-0+1 = \sum_{i=0}^{n-2} n-1-i$$

$$= (n-1) + (n-2) + (n-3) + \dots + 1$$

$$= \frac{(n-1)(n)}{2} = \frac{n^2-n}{2}$$

$\in O(n^2)$

Algorithm BruteForceStringMatch(T[0... n-1], P[0... m-1])
```
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j] do
        j ← j+1
    if j = m return i
return -1
```

Example: Text T - a a a a a a B n = 7 = 2

Pattern P - a a a a B 5

i=0 a a a a a a B — j=0 ✓ ✓ ✓ ✓ ✗ a a a a B — 0 < 5

i=1 a a a a a a B — j=0 ✗ a a a a B

i=2 a a a a a a B — j=0 ✓ ✓ ✓ ✓ a a a a B

Basic operation : comparison.

Time complexity:

Best case : If the pattern is present in the starting index of Text the time complexity $C(n) \in O(m)$

worst case :

$$C(n) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1 = \sum_{i=0}^{n-m} m-1+1$$

$= m \sum_{i=0}^{n-m} 1 = m[n-m+1]$

$= mn - m^2 + m$ if $m^2 = 5^2 = 25$

$\in O(mn)$ $mn = 7 \times 5 = 35$

Algorithm fun(n)
for i ← 1 to n do
for j ← 1 to i do
for k ← 1 to j do
x ← x+1

$\sum_{i=1}^{n} i^k = 1^k + 2^k + \dots \simeq \frac{1}{k+1} n^{k+1}$

Time complexity : $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} 1 = \sum_{i=1}^{n} \sum_{j=1}^{i} j - 1 + 1$

$= \sum_{i=1}^{n} \sum_{j=1}^{i} j = \sum_{i=1}^{n} 1+2+3+\dots+i$

$= \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{1}{2} \sum_{i=1}^{n} i^2 + i = \frac{1}{2} \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i$

$= \frac{1}{2} (1 + 2^2 + 3^2 + \dots + n^2) + (1+2+3+\dots+n)$

$= \frac{1}{2} \left(\frac{1}{2+1} n^{2+1} \right) + \frac{n(n+1)}{2}$

$= \frac{1}{2} \left(\frac{1}{3} n^3 \right) + \frac{n(n+1)}{2}$

$\in O(n^3)$

```
int a=0, b=0;
for (i=0; i<M; i++) {
    a = a+rand();
}
for (j=0; j<M; j++) {
    b = b+rand();
}
```

Time complexity : $\sum_{i=0}^{N} \sum_{j=0}^{M} 1 = \sum_{i=0}^{N} (M+1) = (M+1) \sum_{i=0}^{N} 1$

$= (M+1)(N+1) = MN + M + N + 1$

$O(M) \quad O(N) \quad \in O(M+N)$

```
function (int n)
{
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=n; j++) {
            print("*");
            break;
        }
    }
}
```

Time complexity : $\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} (n-1+1) = n \sum_{i=1}^{n} 1$

$= n - 1 + 1 = n$

$\in O(n)$

```
for (int i=1; i<=n; i*=c) {
}
for (int j=1; j>0; j/=c) {
}
```

Time complexity

$\sum_{i=0}^{n} a^i = 1 + a + a^2 + \dots + a^n = \frac{a^{n+1} - 1}{a-1}$

$1 + c + c^2 + c^3 + \dots + c^k = \frac{c^{k+1} - 1}{c-1}$

$= c^{k+1} = c^k$

$c^k \le n$

$\log c^k = \log n$

$k \log c = \log n$

$= c^k (\log n)$

$a^{\log_b n} = n^{\log_b a}$

$= n^{\log_c c}$

PD 05

3.3.2023 10m

Asymptotic notation:

Types:

upper bound **Big-oh (O)** : Substitute n=1 that will be C

A function t(n) is said to in O(g(n)) denoted t(n) ∈ O(g(n)), if t(n) is bounded above by some constant multiple of g(n) for all large n that is if there exist some positive constant c and some non negative integer n_0 such that (t(n) is) $t(n) \leq c\,g(n)$ for all $n \geq n_0$

$t(n) \leq c\,g(n)$

$100n+5 \in O(n^2)$ ⇒ $100n+5 \leq c(n^2)$

(t(n)) g(n)

assumption ① $100n+5 \leq 100n+n$ n=1 $105 \leq 101(n^2)$

$100n+5 \leq 101n$ $105 \leq 101$

it is c=101 n=1 $105 \leq 101$ n=2 $205 \leq 404$ (need better)

n=2 $205 \leq 202$ ② $100n+5 \leq 100n+5n$

n=3 $305 \leq 303$ $100n+5 \leq 105n$ (c=105)

n=4 $405 \leq 404$ n=1 $105 \leq 105$

it is $n_0=5$ Big n=5 $505 \leq 505$ ✓ $100n+5 \leq c(n^2)$

$n \geq n_0$ $5 \geq 5$ n=1 $105 \leq 105$

lower bound **Big-omega (Ω)** :

A function (t(n)) is said to in Ω(g(n)) denoted t(n) ∈ Ω(g(n)), if t(n) is bounded below by some constant multiple of g(n) for all large n that is if there exist some positive constant c and some non negative integer n_0 such that $t(n) \geq c\,g(n)$ for all

① $n^3 \in \Omega(n^2)$

$t(n) \geq c\,g(n)$

$n^3 \geq n^2$

c=1, n_0=1 n=1 $1^3 \geq 1^2$

$\frac{n(n-1)}{2} \in \Omega(n^2)$ ⇒ $\frac{n(n-1)}{2} \geq c(n^2)$

$\frac{n^2}{2} - \frac{n}{2} \geq \frac{n^2}{2} - \frac{n}{2} \times \frac{n}{2}$

$\frac{n^2}{2} - \frac{n}{2} \geq \frac{n^2}{4}$

$c = \frac{1}{4}$

$n_0 = 2$ n=2

exact **Big-Theta (Θ)**

A function t(n) is said to in Θ(g(n)) denoted t(n) ∈ Θ(g(n)) if t(n) is bounded both above and below by some constant multiple of g(n) for all large n that is if there exist some positive constant c_1 and c_2 and some non negative integer n_0 such that $c_2\,g(n) \leq t(n) \leq c_1\,g(n)$ for all $n \geq n_0$

* $\frac{n(n-1)}{2} \in \Theta(n^2)$

$c_2\,g(n) \leq t(n) \leq c_1\,g(n)$

L.H.S: $\frac{n(n-1)}{2} \geq c(n^2)$ R.H.S: $\frac{n(n-1)}{2} \leq \frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2}$

$\frac{n(n-1)}{2} \geq \frac{1}{4}n^2$

$1 \geq 1$ $\frac{n(n-1)}{2} \leq \frac{n^2}{2}$

$c_2 = \frac{1}{4}$ $c_1 = \frac{1}{2}$

$n_0 = 2$ $n_0 = 2$ $1 \leq 2$

$c_2 = \frac{1}{4}$, $n_0 = 2$, $c_1 = \frac{1}{2}$ $n_0 = 1$

$\frac{1}{4}n^2 \leq \frac{n^2-n}{2} \leq \frac{1}{2}n^2$

$\therefore \frac{n^2-n}{2} \in \Theta(n^2)$

Using limits for comparing orders of growth:

$\lim_{n\to\infty} \frac{t(n)}{g(n)} =$ { 0 implies that $t(n)$ has a smaller order of growth
c " " same
∞ " " larger

1. $100n + 5 \in O(n^2)$ (2) $\frac{n^2-n}{2} \in \Omega n^2$

$t(n) = 100n+5$, $g(n) = n^2$

$\lim_{n\to\infty} \frac{100n+5}{n^2} = \frac{100}{2n}$

$= \frac{100}{\infty}$

$= 0$

$\lim_{n\to\infty} \frac{(n^2-n)/2}{n^2} = \frac{\frac{1}{2}(2n-1)}{2n} = \frac{1}{2}$

$= \frac{1}{2}$ constant $\neq 0$

3. $t(n) = n^3$, $g(n) = n^2$

$\lim_{n\to\infty} \frac{n^3}{n^2} = \frac{3n^2}{2n} = \frac{6n}{2} = 3(\infty) = \infty$

Basic efficiency class: (increasing order)

$O(1)$ = constant time
$O(\log n)$ = logarithmic time (only one half of input is used to find solution)
$O(n)$ = linear time
$O(n \log n)$ = n logarithmic time (like divide and
$O(n^2)$ = quadratic time (nested loop)
$O(n^3)$ = cubic time (3 for loop)
$O(2^n)$ = exponential time } scientific prob
$O(n!)$ = factorial time execution } (calculate you
approximate solution

Determine whether the following assertions are t
or false.

i) $\frac{2n(n-1)}{2} \in O(n^3)$ ii) $\frac{2n(n-1)}{2} \in O(n^2)$

iii) $\frac{2n(n-1)}{2} \in \Theta(n^3)$ iv) $\frac{2n(n-1)}{2} \in \Omega(n)$

For each of the following functions, indicate the class $\Theta(g(n))$ func belongs to prove your assertions
a) $(n^3+1)^6$ b) $\sqrt{10n^4+7n^2+3n}$ c) $3^{n+1} + 3^{n-1}$

Arrange the following functions in the increasing order of growth $\log n$, 3^{3n}, 3^{2n}, $(n^2+3)!$, $\log n^3 n$
$0.05n^{10} + 3^{n^3} + 1$
Indicate each of the following func indicate how much the func value will change if the it arguements increased by threefold $n \log 2n$, n, n^3, n^2, $n!$, 2^n

$\sum_{i=2}^{n-1} i$ $\sum_{i=2}^{n-1} i^{2+1}$

4. i) $t(n) = \frac{2n(n-1)}{2}$ $g(n) = n^3$

$\lim_{n\to\infty} \frac{n(n-1)}{n^3} = \frac{1}{3n^2}$

$= \frac{1}{\infty}$

$= 0$ (True)

(ii) $t(n) = \frac{2n(n-1)}{2}$, $g(n) = n^2$

$\lim_{n\to\infty} \frac{n(n-1)}{n^2} = \frac{n^2-n}{n^2}$

$= \frac{2n-1}{2n} = \frac{2}{2} = 1 = C$ (False)

iii) $t(n) = \frac{2n(n-1)}{2}$ $g(n) = n^3$

= False

iv) $\lim_{n\to\infty} \frac{n(n-1)}{n} = \frac{2n-1}{1}$

$= 2(\infty) - 1 = \infty$ (False)