Prime number testing:

integer 1 is not prime.

integer 2 is the least prime number.

Def: Prime: If natural number n has only two divisor viz. 1 & n then it is known as a prime number.

Putting it in other words

If n has a divisor (= which divides n completely with remainder = 0) between 2 to n-1 then it cannot be a prime number.

```
int is_prime(int n);
/* @ input : a natural number n.
   @ output : 1 if n is prime
              0 if n is not prime.
*/
#define TRUE 1
#define FALSE 0.
```

Approach: Let k be any arbitrary number between 2 to n-1.

if n is completely divisible by k then n is not prime and we should return false.

```
if (n % k == 0)
    return (FALSE)
```

to carry out this logic for all k between 2 to n-1 we can use a loop

```
for (k=2; k<n; ++k)
    if (n % k == 0)
        return (FALSE).
```

If control flow comes out of for loop then value of k must be n and (n % k == 0) must be FALSE for all k from 2 to n-1.

Because if it were true for any number between 2 to n-1 we would return from procedure.

Control flow comes out of for loop → n has no complete divisor between 2 to n-1.

→ n is prime!

```
int is_prime(int n)
{ int k;
  for (k=2; k<n; ++k)
```

$n \geq 3$.

```
        if (n % k == 0)
            return (FALSE)
    return (TRUE);
}
```

n = 2

```
int is_prime(int n)
{
    int k;
    if (n <= 1)
        return FALSE;
    if (n == 2)
        return (TRUE)
    for (k = 2; k < n; ++k)

        if (n % k == 0)
            return (FALSE)
    return (TRUE);
}

void prime_numbers(int * p_arr, int N)
{
    let p_arr[i] be any valid integer
    in array (0 <= i < N).
    How to check whether p_arr[i] is
    prime or not?
    is_prime(p_arr[i]) == TRUE
```

```
if (is_prime(p_arr[i]) == TRUE)
    printf("p_arr[%d]: %d\n", i,
           p_arr[i]);
```

to generalize this for all elements in p_arr, use loop

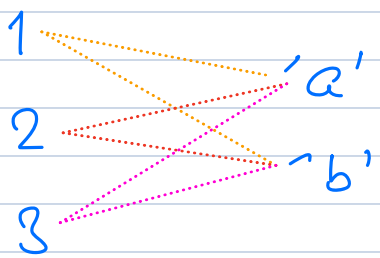
```
for (i=0; i<N; ++i)
    if (is_prime(p_arr[i]) == TRUE)
        printf("p_arr[%d]: %d\n", i,
               p_arr[i]);
}

void prime_numbers(int *p_arr, int N)
{ /* with is_prime() */
```

$A1 = \{1, 2, 3\}$

$A2 = \{'a', 'b'\}$.

$A_1 \times A_2$ You must pair every element of A_1 with A_2



(x, y) = order pair.

Pair = जोड़ी

Ordered = Sequence matters

$x \neq y$.

(x, y) (y, x) are two different ordered pairs

tuple = ordered pair

$A_1 \times A_2 = \{(x, y) : x \in A_1, y \in A_2\}$.

$(1, 'a')$
$(1, 'b')$
$(2, 'a')$
$(2, 'b')$
$(3, 'a')$
$(3, 'b')$

$A_1 = \{1, 2, 3\}$.

$A_2 = \{'a', 'b'\}$.

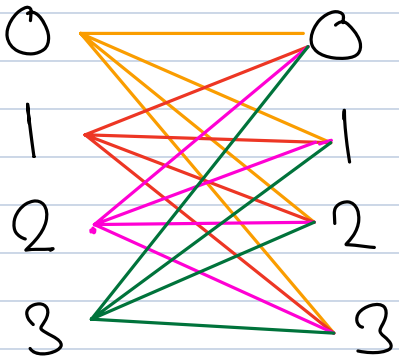
$A_1 \times A_2 = \{(x, y) : x \in A \text{ and } y \in B\}$.

L CARTESIAN PRODUCT.

We can take a cartesian product of set with itself.

$A = \{0, 1, 2, 3\}$.

$A \times A$.



$A \times A = \{ (0,0), (0,1), (0,2), (0,3)$
$(1,0), (1,1), (1,2), (1,3),$
$(2,0), (2,1), (2,2), (2,3),$
$(3,0), (3,1), (3,2), (3,3) \}$.

Let A be the array of N elements and S is some number.

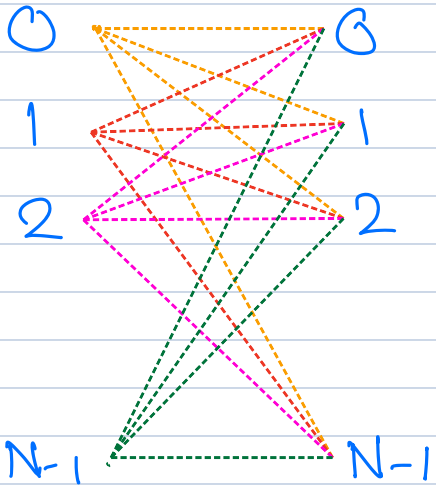
We must find a pair of two distinct elements in array whose sum is $< S$. (we must find all such pairs)

Let A[i] and A[j] be any two elements in array.

```
if ((A[i] + A[j]) <= S)
    printf (A[i], A[j]);
```

i : $0 \leq i \leq N-1$

j : $0 \leq j \leq N-1$



---

let j be any arbitrary element from 0 to N-1

let i be any arbitrary element from 0 to N-1

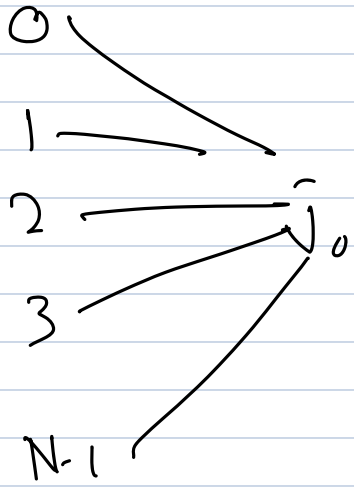
print (i, j)

keeping j same we can alter i from 0 to N-1

```
for (i=0; i<N; ++i)
    printf (i, j)
```

```
for (i=0; i<N; ++i)
    for(j=0; j<N; ++j)
        printf (j)
```

j_0 $0 \le j_0 < N-1$




```
for (i=0; i<N; ++i)
    print (i, j0)
```

```
for (i=0; i<N; ++i)
    for (j=0; j<N; ++j)
        print (i, j)
```

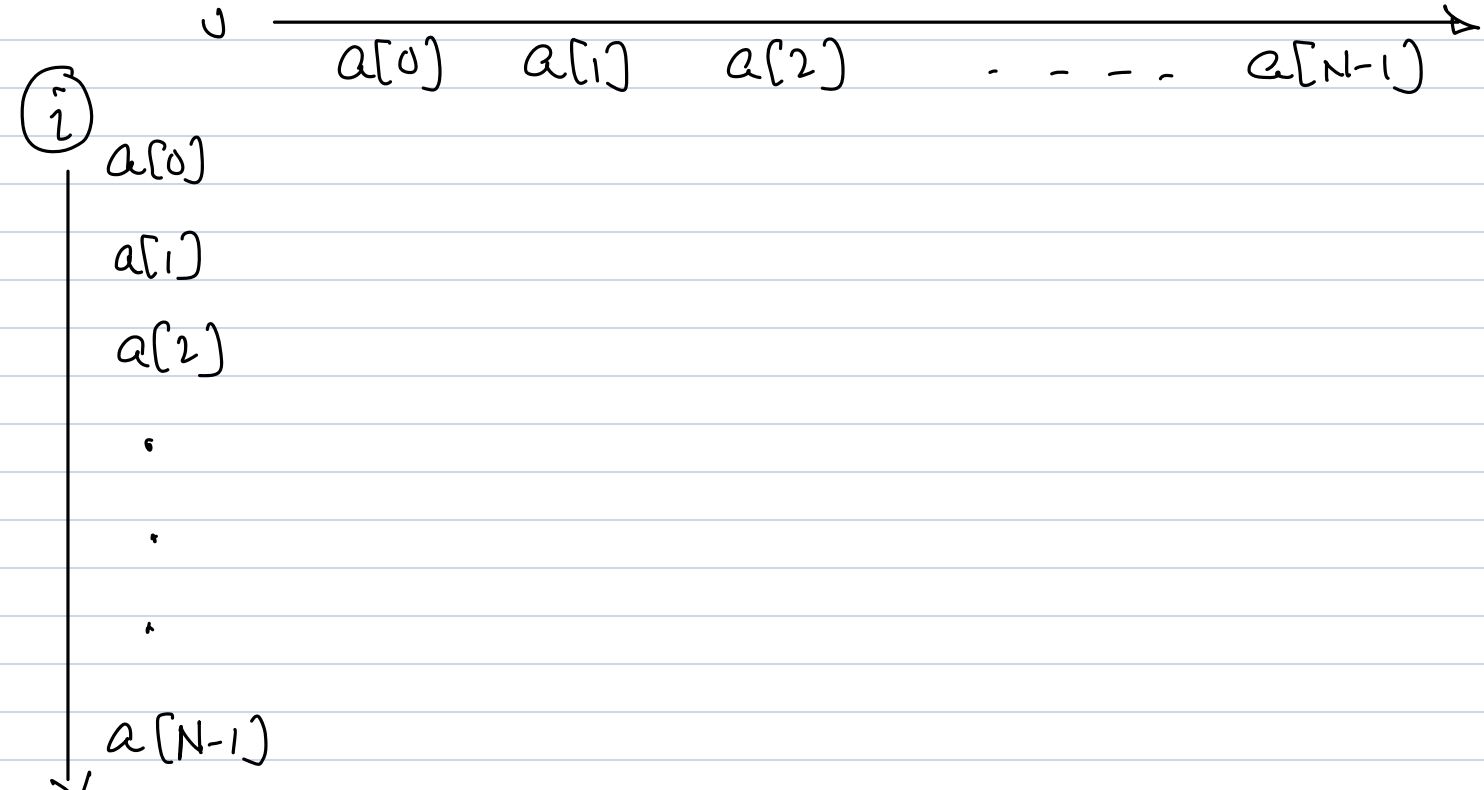
$R_i = \{0, \text{—} N-1\}$.

$R_j = \{0, \text{—} N-1\}$

$R_i \times R_j$

arr : | 0 | 1 | 2 | | N-1 |

arr × arr.



let a[i] & a[j] be any two elements of array

arr[i] + arr[j] <= S

$R_i \times R_j$

```
for(i=0; i<N; ++i)
    for(j=0; j<N; ++j)
        if(arr[i]+arr[j]<=S)
            print(i, j, arr[i], arr[j])
```

```
for(i=0; i<N; ++i)
    for(j=0; j<N; ++j)
        if(i != j && arr[i]+arr[j]<=S)
            printf(arr[i], arr[j]);
```

Cartesian Product of index range →
Nested loop!

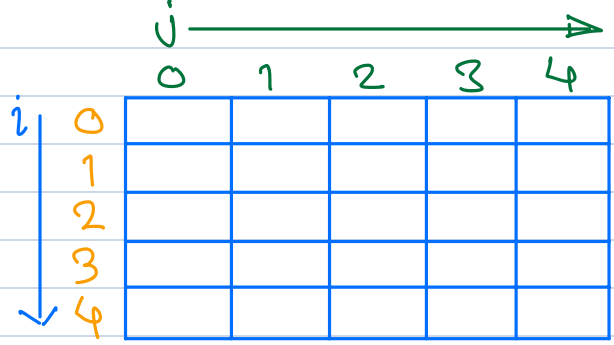
---

let a be an array 5 of integers.

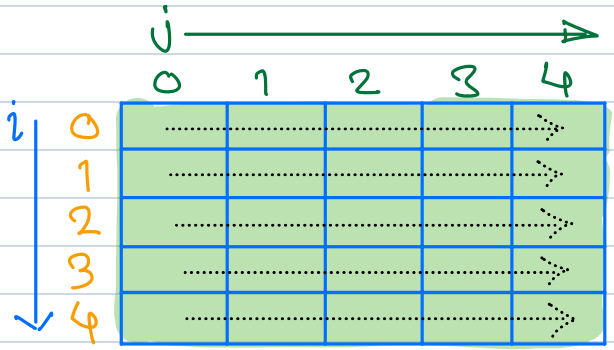
int a[5] = {-, -, -, -, -};

Range(a) = Range(0,5) = {0, 1, 2, 3, 4}.

Ra x Ra.



Pattern - 1: All pairs.



Code: for version

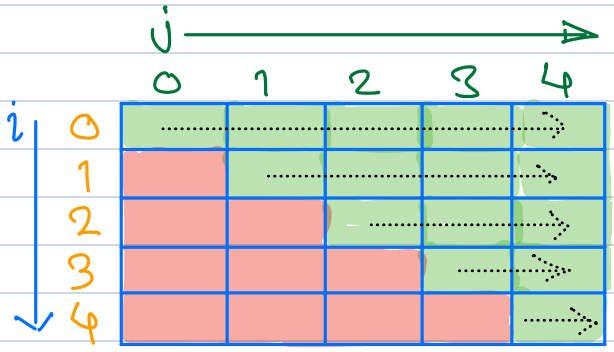
```
int i, j;
for (i=0; i<5; ++i)
{
    for (j=0; j<5; ++j)
    {
        a[i] a[j] logic
    }
}
```

Code while version

```
i=0;
while(i<5)
{
    j=0
    while(j<5)
    {
        a[i], a[j] logic

        j=j+1;
    }
    i=i+1;
}
```

Pattern 2: Upper triangular matrix



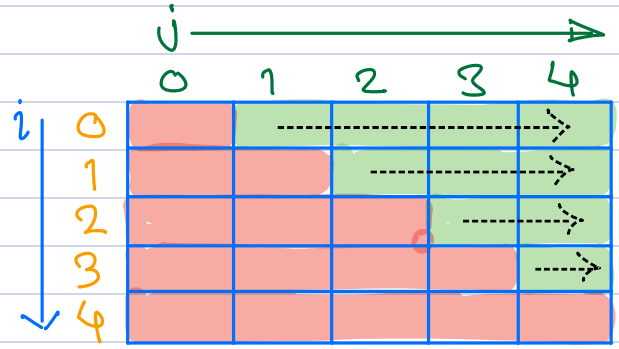
Code: for version

```
for (i=0, i<5; ++i)
    for (j=i; j<5; ++j)
        Common logic
        on a(i) & a(j)
```

while version

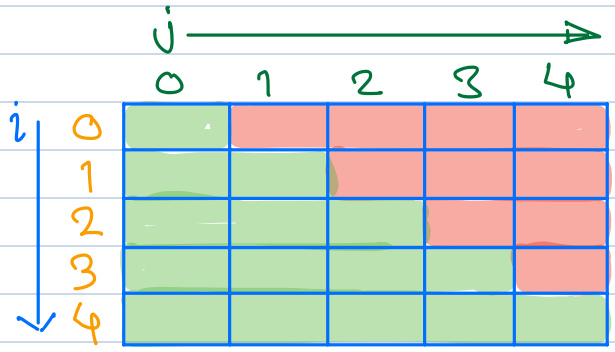
```
i=0;
while (i<5)
{   j=i;
    while (j<5)
    {   Comman logic
        for i & j.
        j=j+1;
    }
    i = i+1;
}.
```

Pattern-3: Upper triangular matrix → Diagonal



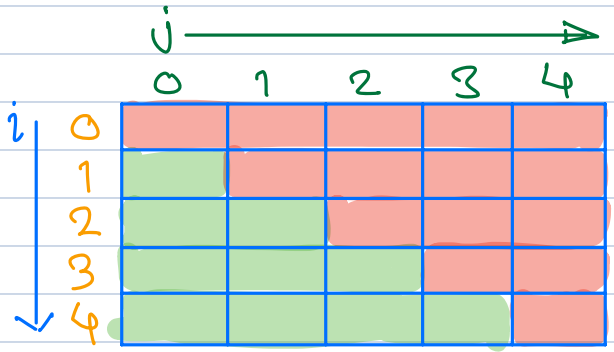

```
for (i=0; i<5; ++i)
{   for (j=i+1; j<5; ++j)
    {   Common logic on
        a[i) & a[j]
    }
}
```

Pattern-4



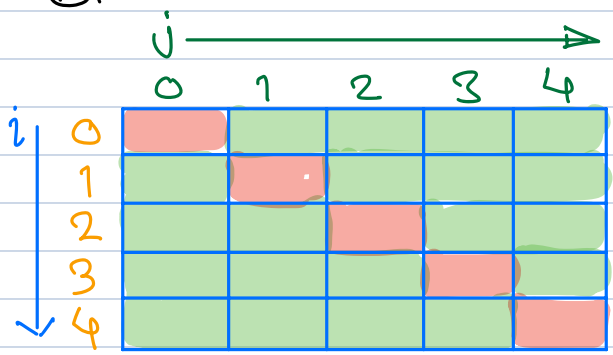

```
for(i=0; i<5; ++i)
    for(j=0; j<=i; ++j)
        Common logic.
```

Pattern-5




```
for(i=0; i<5; ++i)
    for(j=0; j<i; ++j)
        Common logic.
```

Pattern-6.




```
for(i=0; i<5; ++i)
    for(j=0; j<5; ++j)
        if(i != j)
        {
            // Common logic.
        }
```

```
for(i=0; i<N; ++i)
    for(j=0; j<N; ++j)
        if(i != j && (a[i]+a[j])<=j)
            print(a[i], a[j]
```

```
for(i=0; i < N; ++i)
    for(j=(i+1); j< N; ++j)
        if((a[i] + a[j])<=j)
            print(a[i], a[j])
```
