Isprime. → Range → b/w 1 to 100

queries → 10,000 } →

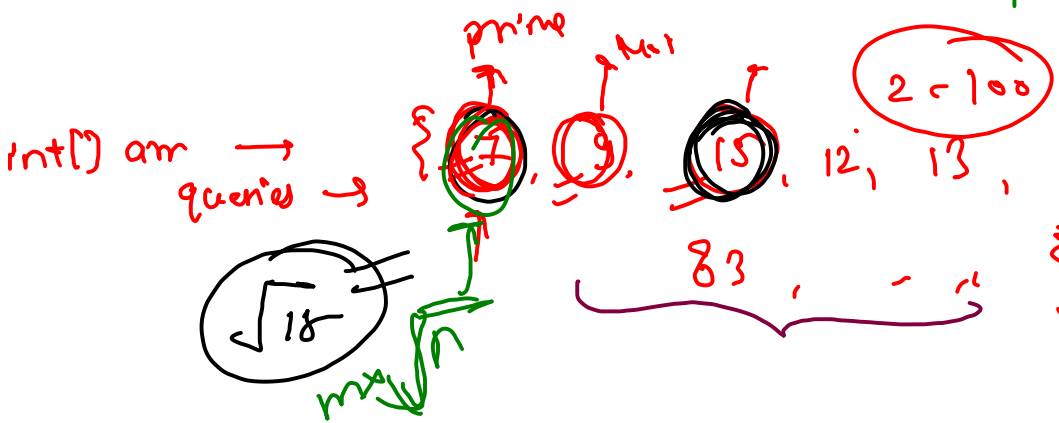
for Every queries → [prime / Not prime.

$\sqrt{n}$

$n \geq 100$

$2 \leq arr[i] \leq 100$

Integer. Range.



$\equiv$ 10^4 → m { length of queries }

→ n { Range of element

cost for Single check = $\sqrt{n}$.

Cost for m check := $m \times \sqrt{n}$.

Overall Complexity ⇒

$O(m \times \sqrt{n})$

i = 2 ; i ≤ √n

Pre Computation

queries → 10,000

Range of number → 25

boolean preCheck[ ]

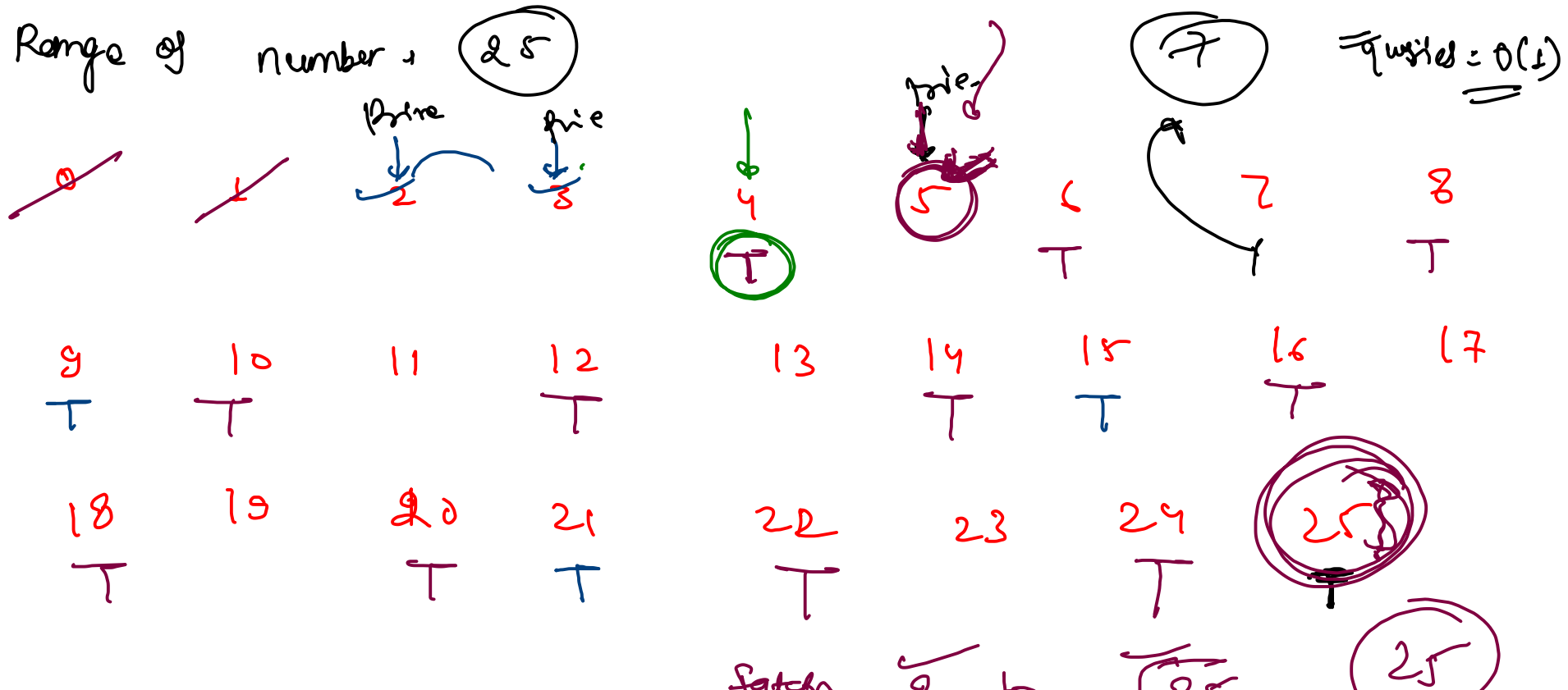
range of i

preCheck[i] == True

↓

i is Not prime.

queries = O(1)



Sieve of eratosthenes - (Sieve theorem)

pre calculation

$n + O(m) \equiv O(m)$

$m >>> n$ $\quad m^2 >> n$

n Range → 25

m queries → 10,000

→ O(m) → prime

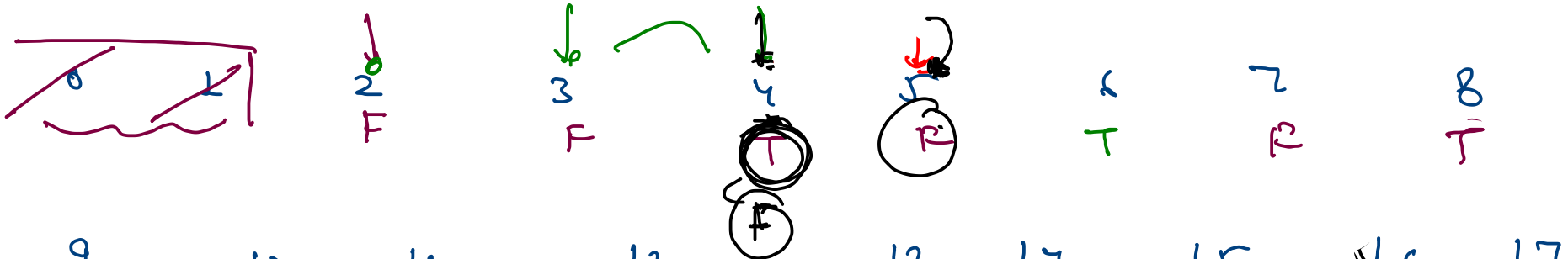
Precalculation → Answer

if (primes [i] == true) i is Not prime.

$O(n+m) \equiv O(m)$

boolean [] primes = new boolean [range + 1];

$5 <= \sqrt{25}$



2, 3, 5

$$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \cdots + \frac{n}{\sqrt{n}}$$

denominator is prime.

$$\frac{25}{\sqrt{25}} = 5$$

$$n\left[\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \cdots + \frac{1}{\sqrt{n}}\right]$$

$\log \log n \longrightarrow$ {covergence, – divergence}

$$n \log[\log(n)]$$

$$\log \log(n) \equiv 1$$

$\Rightarrow$ (n) $\longrightarrow$ is Resp.

$m_1 \rightarrow 22$.

$m_2 \rightarrow 51$

51
22
(29)

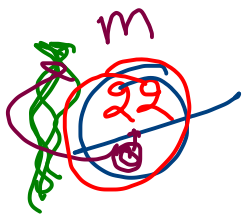
(2) → (f2)

(3) →

(4)

(5)

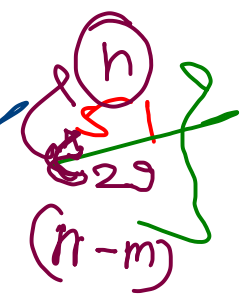
Segmented Sieve



23 1 | 24 2 | 25 3 | 26 4 | 27 5 | 28 6 | 29 7 | 30 8 | 31 9

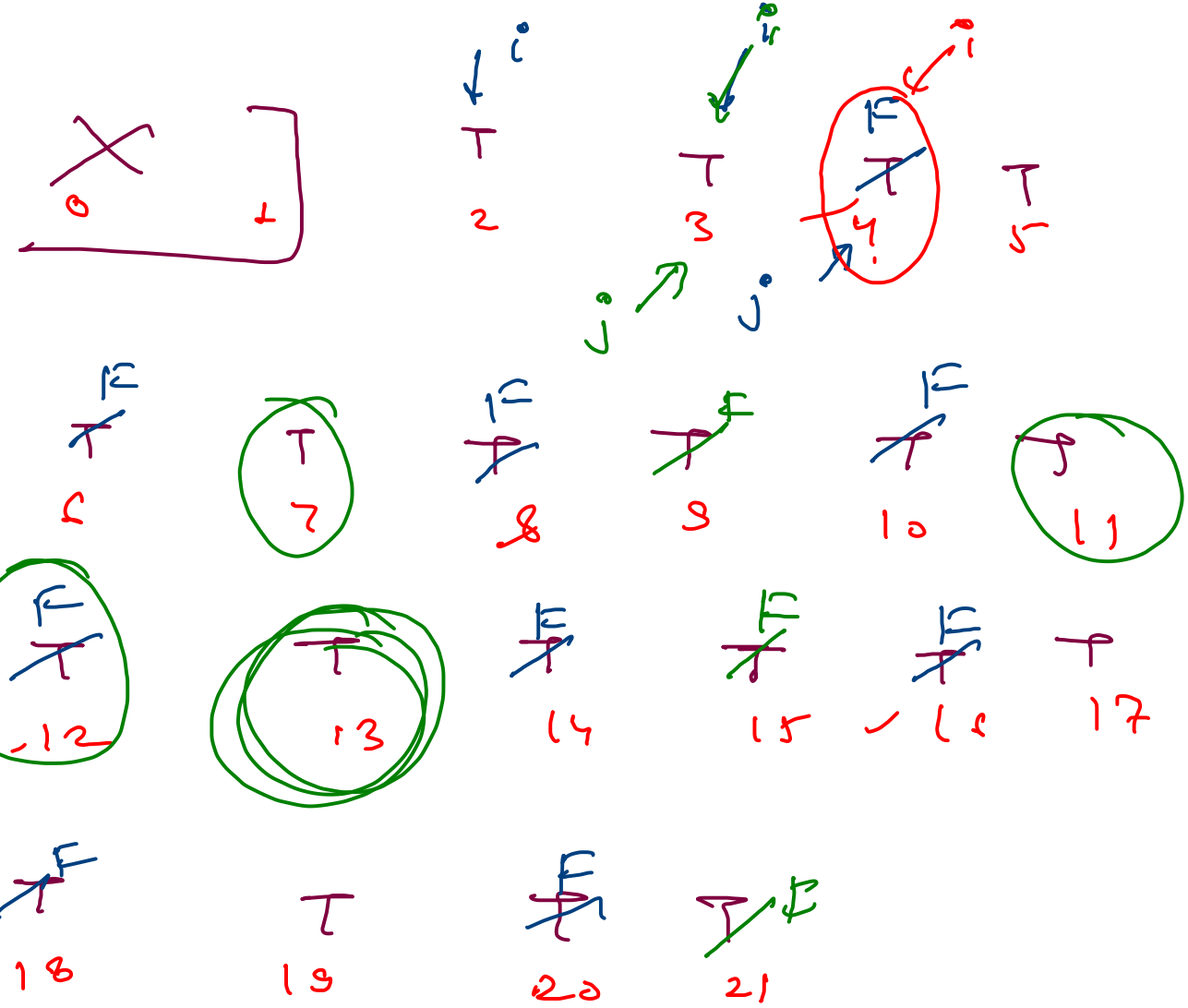
32 10 | 33 11 | 34 12 | 35 13 | 36 14 | 37 15 | 38 16 | 39 17 | 40 18 | 41 19 | 42 20

43 21 | 44 22 | 45 23 | 46 24 | 47 25 | 48 26 | 49 27 | 50 28

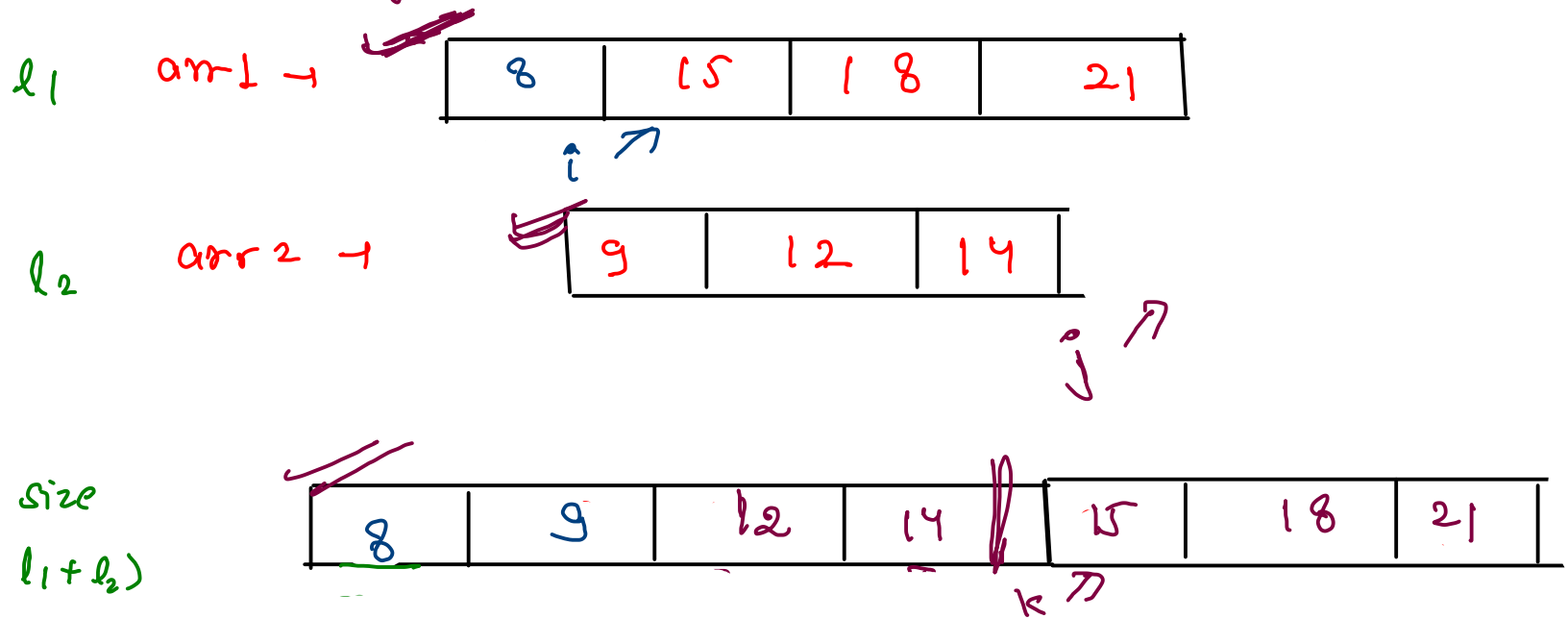


2 < 21

```java
// sieve
public static void sieve(int[] queries) {
    int range = 20;
    boolean[] primes = new boolean[range + 1];

    // make every element as true
    Arrays.fill(primes, true);

    // primes[i] == true, i is prime
    // primes[i] == false i is not prime

    for(int i = 2; i * i <= range; i++) {
        if(primes[i] == false) continue;

        for(int j = i + i; j <= range; j += i) {
            primes[j] = false;
        }
    }

    for(int q = 0; q < queries.length; q++) {
        int num = queries[q];
        if(primes[num] == true) {
            System.out.println(num + " is Prime");
        } else {
            System.out.println(num + " is Not Prime");
        }
    }
}
```



Merge Two Sorted Array-




```
if( arr1[i] < arr2[j] ){
    res[k] = arr1[i];
    i++;
} else {
    res[k] = arr2[j];
    j++;
}k++
```

MergeSort (High level Analysis)

Expectation, Sort(arr, lo, hi)

faith → First Half → sort(arr, lo, mid)

Second Half → sort(arr, mid+1, hi)

Merging of faith and Expectation → sort(arr, lo, hi) = { Recursive work

mid = (lo+hi)/2;

First Half → sort(arr, lo, mid);

Second Half → sort(arr, mid+1, hi);

Done by self

res = Merging Two Sorted arr(F.H, S.H);

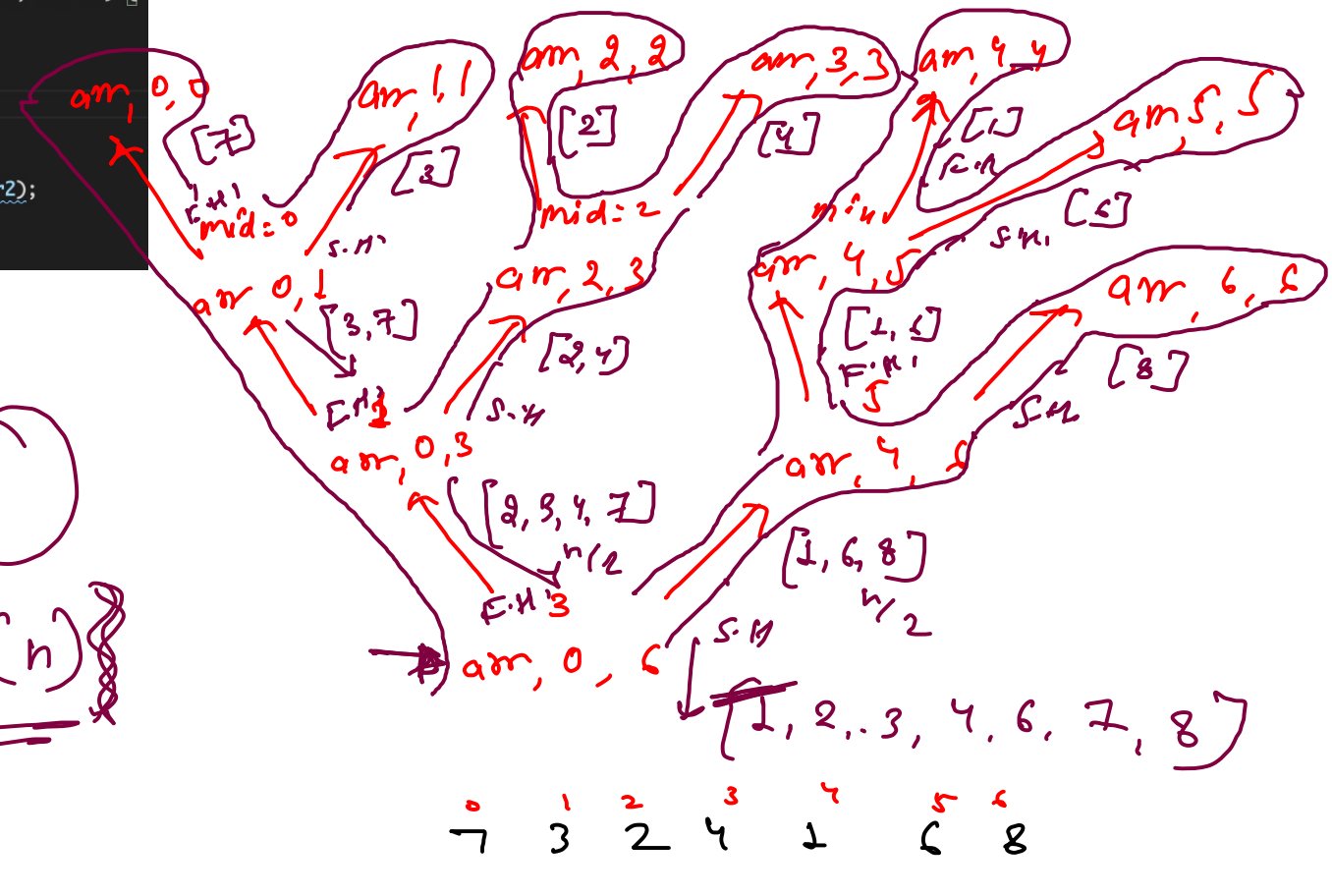
return res; }

Low level Analysis (MergeSort)

```java
public static int[] mergeSort(int[] arr, int lo, int hi) {
    int mid = lo + (hi - lo) / 2;

    // faith
    int[] arr1 = mergeSort(arr, lo, mid);
    int[] arr2 = mergeSort(arr, mid + 1, hi);

    int[] res = mergeTwoSortedArrays(arr1, arr2);
    return res;
}
```

lo == hi → Base case

int[] base = new int[1];
base[0] = arr[lo];



At max → n

Space → O(n)

Bubble Sort

↳ Adjacent Swapping

arr → 7 6 9 2 1 (0 1 2 3 4)

Sum. Sat

10 − 2
2 − 6
6 − 10

i=0

I

1.1 6 7 9 2 1
1.2 6 7 9 2 1
1.3 6 7 2 9 1
1.4 6 7 2 1 9

→ 6 7 2 1 9

n

j=0; j < size−i

i=1

II

2.1 6 2 7 1 | 9
2.2 6 2 1 7 | 9
2.3 6 2 1 | 7 9

n−1

j=2

i=2

III

3.1 2 6 1 | 7 9
3.2 2 1 6 | 7 9
3.3 2 1 | 6 7 9

n−2

i=3

IV

4.1 1 2 | 6 7 9
4.2 1 | 2 6 7 9

n−3

i=4

VI

5.1 1 | 2 6 7 9 |

no. of Itr: 1+2+...+n

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

Cost $= \frac{n^2}{2} + \frac{n}{2}$

$= O(n^2)$

Selection Sort

↳ Selection of Min Element