→ STL INBUILT ALGORITHMS

```
INT N;      // 6
CIN >> N;
VECTOR<INT> V(n);   // 2 3 1 6 7 6

FOR (INT i=0;  i<n;  i++)
{
    CIN >> V[i];
}
```

→ These all STL inbuilt Algorithms takes O(n) T.C, and they can take custom iterators like V.BEGIN( ) + 2 or V.END( ) - 3 etc.

→ MIN_ELEMENT and MAX_ELEMENT function returns pointer in case of array, and iterator in case of vector.

```
AUTO IT = MIN_ELEMENT(V.BEGIN(), V.END());
COUT << *IT << "\n";
```

→ Another way to write upper code:

```
INT MIN = *MIN_ELEMENT(V.BEGIN(), V.END());
COUT << MIN << "\n";      // 1

INT MAX = *MAX_ELEMENT(V.BEGIN(), V.END());
COUT << MAX << "\n"       // 7
```

→ ACCUMULATE() function gives the sum of an array or vector.
→ SYNTAX: ACCUMULATE(STARTING_ITERATOR, ENDING_ITERATOR, INITIAL_SUM)

```
INT SUM = ACCUMULATE(V.BEGIN(), V.END(), 0);
COUT << SUM << "\n"  // 25
```

→ COUNT() function gives the count of an element in an array or vector.
→ SYNTAX: COUNT(STARTING_ITERATOR, ENDING_ITERATOR, ELEMENT)

```
INT CT = COUNT(V.BEGIN(), V.END(), 2);
COUT << CT << "\n"   // 1
```

→ FIND() function simply checks if that element is present in that array or vector and if yes then it returns a pointer or iterator of that element only, else give next to last iterator (V.END()).

```
AUTO IT = FIND(V.BEGIN(), V.END(), 2);
IF (IT != V.END())
{
    COUT << *IT << "\n";  // 2
}
ELSE
{
    COUT << "ELEMENT NOT FOUND" << "\n";
}
```

→ REVERSE() function simply reverses the array or vector.

```
REVERSE(V.BEGIN(), V.END());
FOR (AUTO VAL: V)
{
    COUT << VAL << " ";  // 6 7 6 1 3 2
}
```