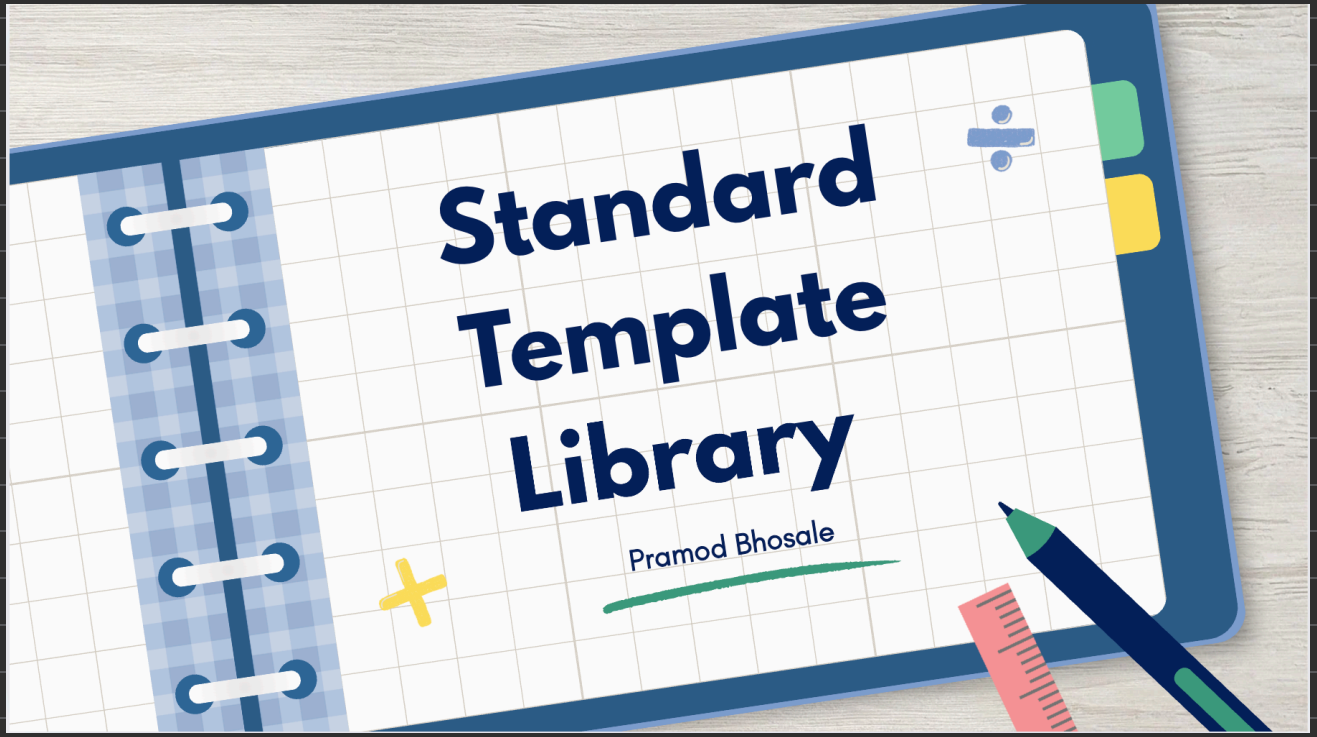
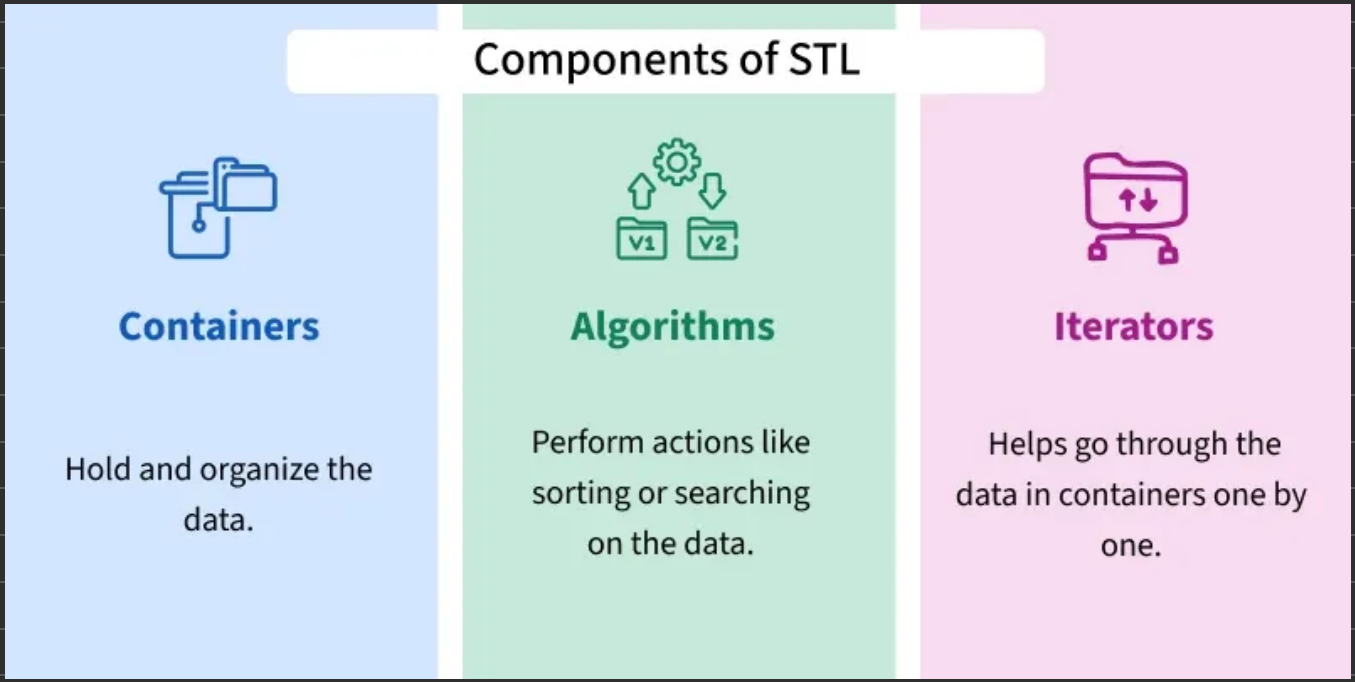
इतना मत गिरो कि उठने का यक़ीन टूट जाए,
इतना उठो कि गिरने वाला भी सोच में पड़ जाए।



Standard Template Library

STL is a collection of pre-built classes and functions



Container

Containers are the data structures used to store objects and data according to the requirement.

Containers can be further classified into 4 types:

1. Sequence Containers : Vector, Deque, List
2. Associative Containers : Set, MultiSet, Map, Multimap
3. Unordered Associated Containers : Unordered set, unordered Map,

Algorithms

STL algorithms offer a wide range of functions to perform common operations on dat

Sort()

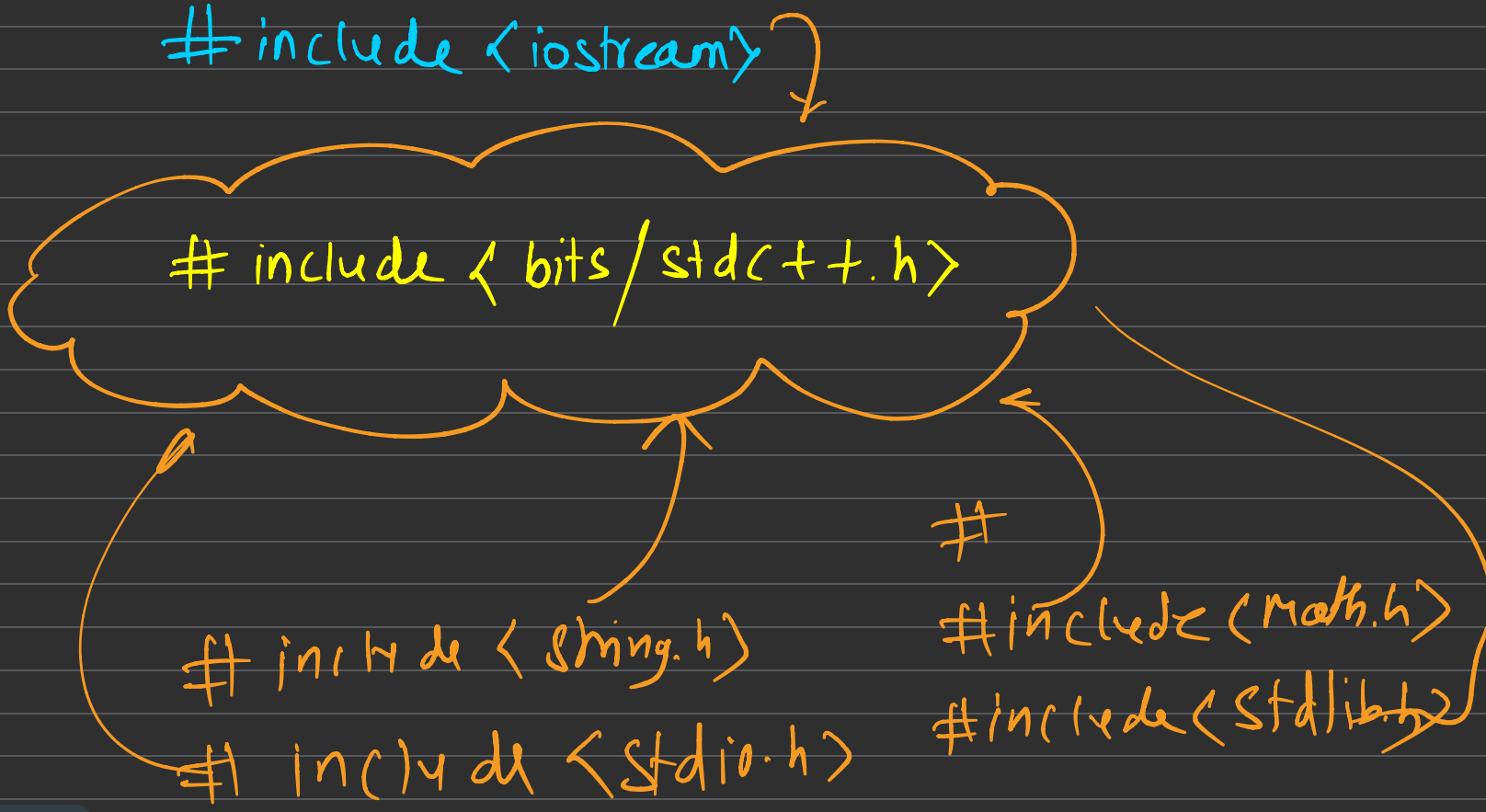
reverse()

min_element()

max_element()

count()

ACCUMULATE()



vector

header : #include <vector>

Syntax :

vector < dataType > v

vector < dataType > v (size)

vector < int > v (5)

capacity = 5

size() = 3

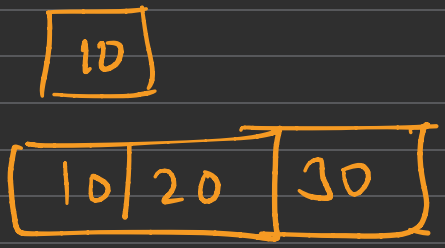
* v.push_back(data)

(10)

(20)

(30)

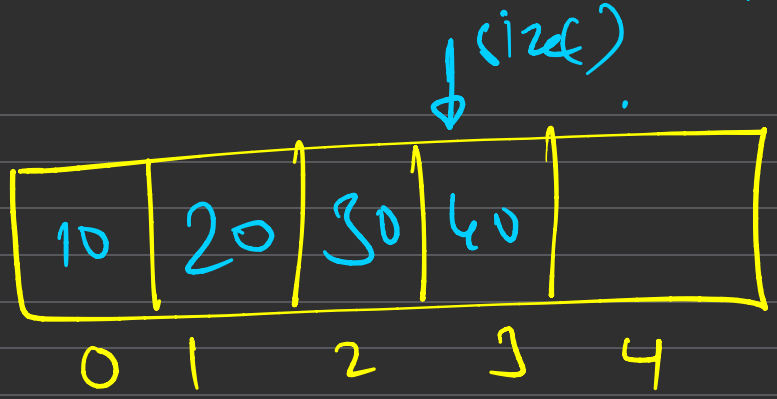
(40)



* v.pop_back() ⟶ Remove last element

* v.size() ⟶ size of vector

* v.empty() { T if empty
F



size = 0 ~~0~~ 2

pop_back

vector<int> v(5)

v.push_back(11)

(20)

v[(size)] = x

size++

Pairs

< val1, val2 >
< val2, val3 >

header = include <pair>

Syntax:

pair < dataT1 , dataT2 > p

p.first = dataT1

p.second = dataT2

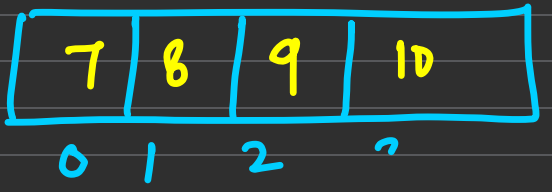
pair < int, string > p[5]

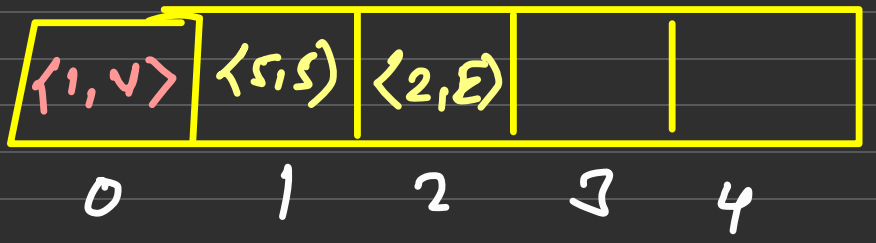
(Roll, Name)

p[0].first = 1
p[0].second = "Urushali"

P[1].first = 5
p[1].second = "Sanchita"

p[2].first = 2
p[2].second = "Eshita"






```
for ( i=0 ; i < p.size() ;i++)
{
    cout << p(i). first
    cout << p(i). second

}
```

Set

- mainly used to store data

- data is in sorted order (increasing)
- duplicate not allowed

Unordered Set

- data can be in any order
- duplicate not allowed

Set

header = include <set>

Syntax:

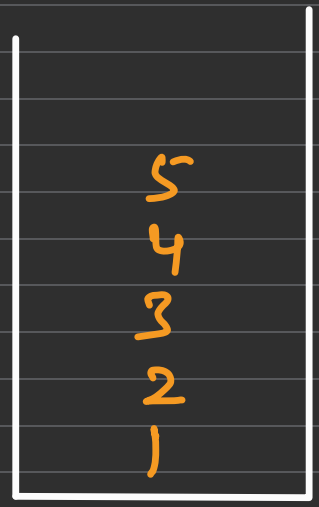
- set <dataType> s (increasing)
- unordered_set <dataT> us

- s.size() → mainly gives No. of Elements
- s.empty() → Either Empty

- s.insert(data) → insert data in set

```
s.insert(5)
s.insert(2)
s.insert(1)
s.insert(4)
s.insert(3)
s.insert(2)
```

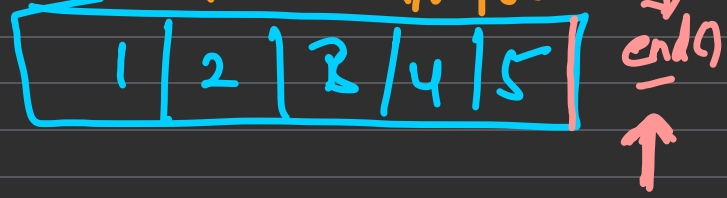
ignore



- Traversing set

```
for ( auto i : s)
{
    cout << i
}
```

Addr = 100 400 600 700 800



- s.find(data)

auto x = s.find(4)

cout << (*x)

auto y = s.find(20)

if (y == s.end())
{
    Not found
}
else

Multiset

- Allows duplicate
- Sorted data

- s.insert(val)
- s.size()
- s.empty()
- s.find(val) { end()
  Add

```
for (auto i : s)
    cout << i
```

set <int> s

multiset <int> ms

unordered_set <int> us

Unordered Set

```
unordered_set <int> us
```

Map

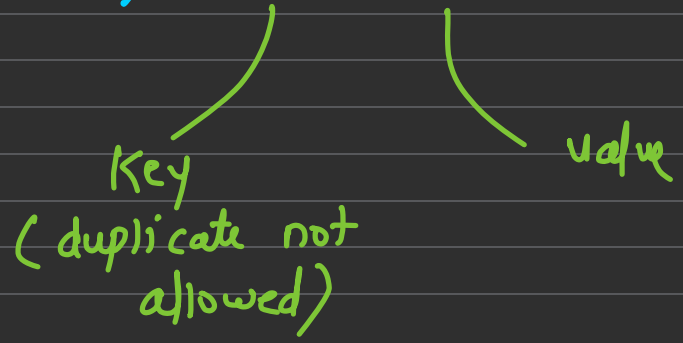
mainly used to store (key, value) pair

* map does not allow duplicate key
* Data is in increasing order → key

header file = #include <map>

Syntax:

map < data1 , data2 > m

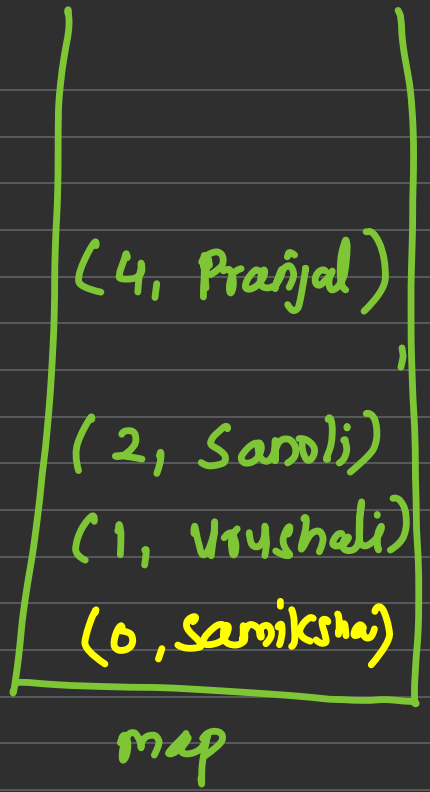


map < int , string > m

m.insert({ 2 , "sanoli" })

m[1] = " Vrushali"

m[0] = " Samiksha"



* Traversing

```
for (auto i : m)
{   cout << i.first
    cout << i.second
}
```

→ key

→ value

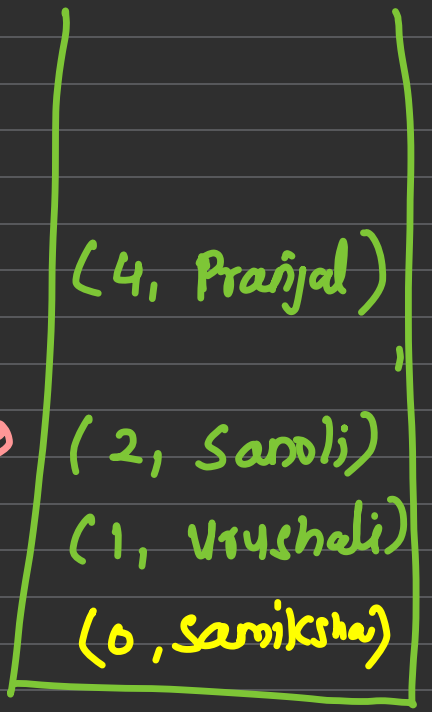
* find :

mainly check whether given Key is present or not

```
auto find = mp.find(2)
cout << find -> first
cout << find -> second
```

Key

find →




```
(*find).first
(*find).second
```

```
auto find = mp.find(20)
if (find == mp.end())
{
    Not found
}
```

* erase()

```
auto find = mp.find(2)
mp.erase(find)
```

Unordered Map

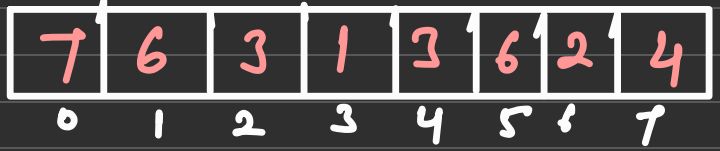
Syntax:

```
unordered_map <data1, ,data2>um
```

header:

```
#include <unorder_map>
```

Count occurrance of every element of array



7 → 1
6 → 2
3 → 2
1 → 1
2 → 1
4 → 1

map < int , int > m

```
for (i=0 ; i<8 ; i++)
{
    mp[a[i]]++
}
```

i = 7

mp(4) ++

0 ++

(4,1)
(2,1)
(1,1)
(3,2)
(6,2)
(7,1)

map

Sort() → mainly used to sort array / string

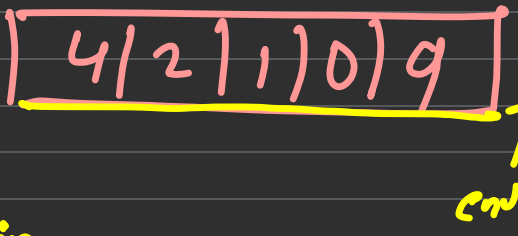
* int arr[] = {1, 4, 2, 0, -1}; N = 5

→ sort(arr, arr+N)

O(N.log N)

* vector<int> a = {4, 2, 1, 0, 9};

→ sort(a.begin(), a.end())



* String s = "Pramod"

→ sort(s.begin(), s.end())

reverse()

* `int arr[] = {1, 2, 3, 4}`

  `reverse(arr, arr+n)`

* `vector<int> a = {4, 3, 2, 1}`

  `reverse(a.begin(), a.end())`

count()

accumulate() → gives you sum of elements of array

int arr[] = {1, 2, 3, 4}

→ int sum = accumulate(arr, arr+n, 0)

vector<int> a = {1, 2, 3, 4}

→ int sum = accumulate(a.begin(), a.end(), 0)

min_element() → Returns min element from array

int arr[] = {3, 4, 6, 2, 0, 1}

int min = *min_element(arr, arr+n)

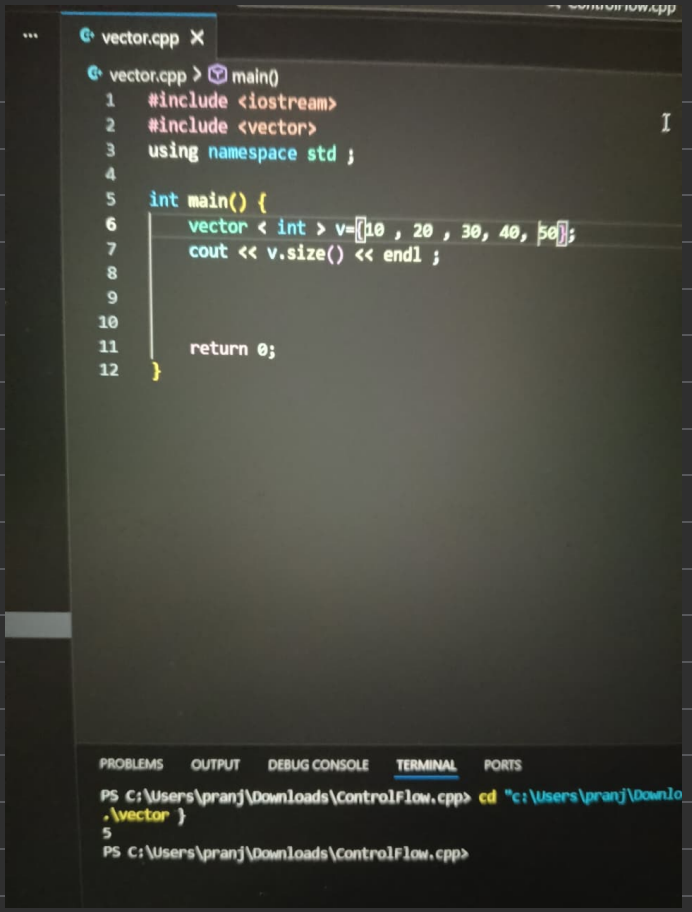
int max = *max_element(arr, arr+n)

X

vector

* min_element(arr.begin(), arr.end)

max_element()



int arr[] = { 1, 2, 3, 4 }