Standard Template Library

1. Vector

In a simple static array, we can not redefine its size. So it is optimal to make static arrays only where the size of the array doesn't change according to the user's needs. Also, the size should be defined before the compilation because stack memory is assigned at the time of compilation of the program. For example,

- **int arr[20];** **Here, the stack memory that the arr requires is fixed**

- **int n;**
  cin >> n;
  int arr[n];
  In this case some random amount of stack memory is assigned to arr but that might be short than what the user might require.

The above problem can be solved using Vectors. **A Vector is a dynamic array that has the ability to modify its shape whenever we perform insertion or deletion in it.** Just like an array, Vector elements are placed in contiguous storage so that they can be accessed using iterators.

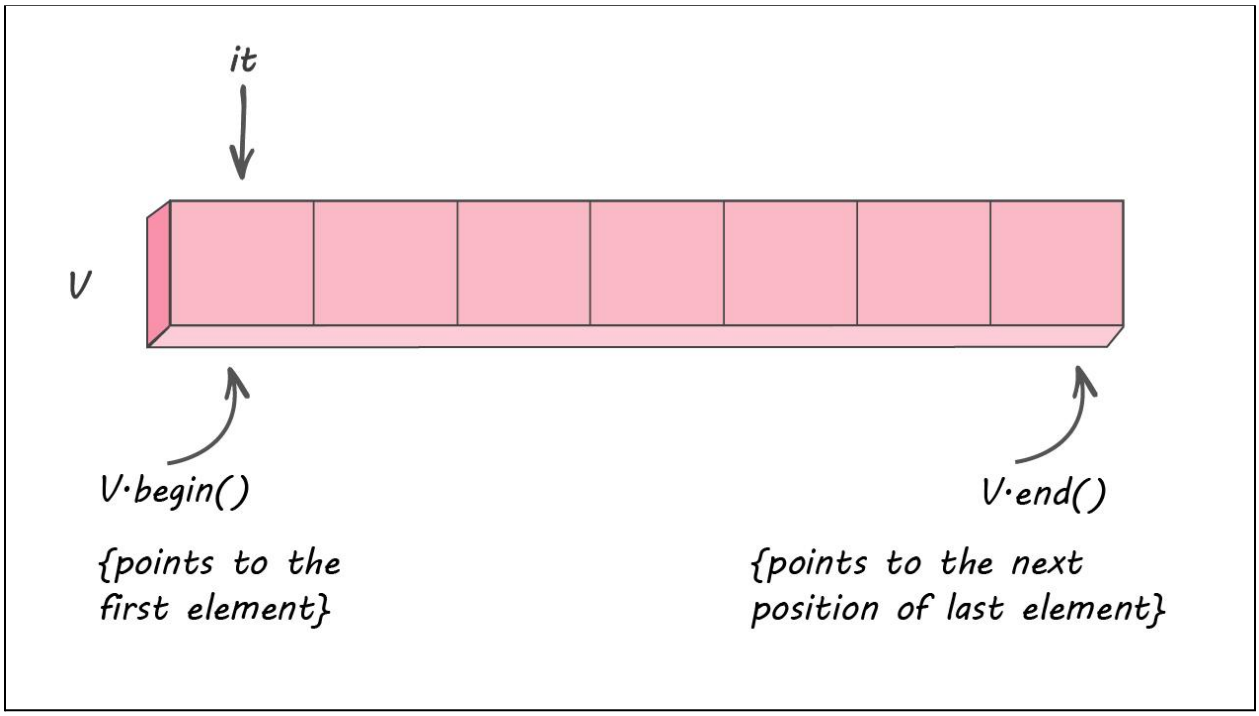
Vector is already present in the standard template library of C++ and can be included in any file using **#include<vector>**

Any new element can be inserted at the end of the vector using the **push_back()** function, which takes differential time because there might be

resizing the vector. While removing the last element takes constant time because no resizing happens.

```
#include<iostream>
#include<vector>

using namespace std;

int main(){

    // while making a vector we need not give the size of the vector
    vector<int> V ;

    // we insert the element at the last index of the vector and
    // the vector automatically reassigns its size
    for(int i=0; i<5; i++){
        V.push_back(i+1);
    }

    return 0;
}
```

There are predefined functions like **begin()**, **end()**, **size()**, etc., that can be used along with a vector.

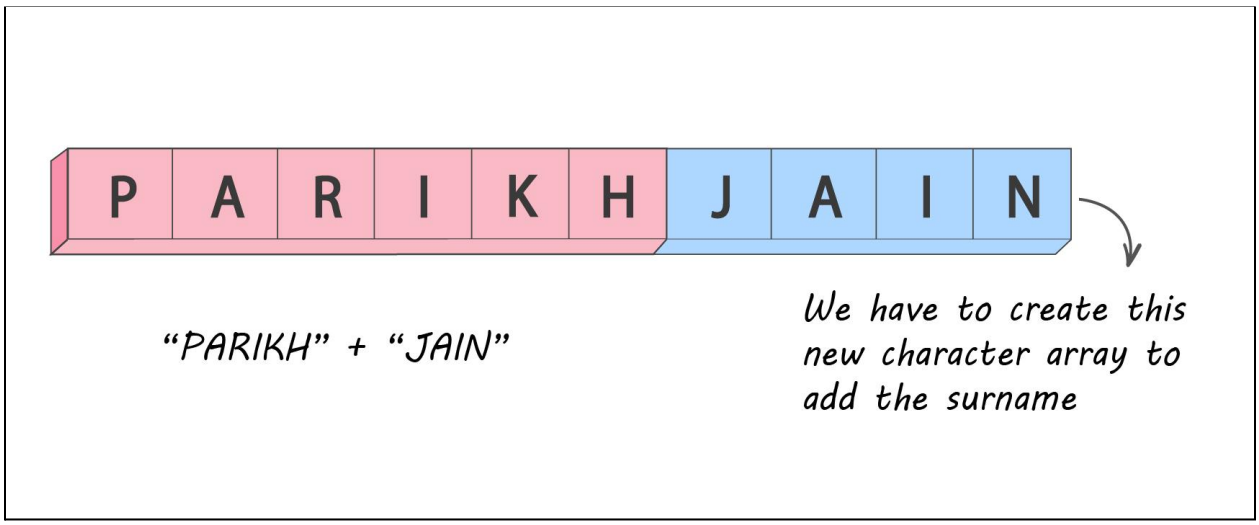


If we need to traverse the entire vector we can do it using creating an iterator:

```
// creating an iterator and traversing the vector using it
vector <int> :: iterator it;
for(it = v.begin(); it != v.end(); it++){
    cout << *it << endl;
}
```

2. Strings

A character array is a static array, and more memory can not be used at run time. Let us suppose we made a character array “PARIKH”. Now we need to add the surname but the length/size of the character array is static, so we have to make a new character array “JAIN” and then use some iteration method to get the full name.



The above problem can be made way easier with a String. A String is a class that defines objects that are represented as a stream of characters.

String class is present in the standard template library of C++, and can be included in any file using **#include<string>**

```
#include<iostream>
#include<string>
```

```
using namespace std;


int main(){

    string s = "PARIKH";
    cout << s << endl; // prints PARIKH
    s = "PARIKH JAIN";
    cout << s << endl; // prints PARIKH JAIN
    return 0;
}
```

Various functionalities of strings are explained in the code below with proper comments:

```
int main(){

    string s = "PARIKH";
    string s1(s, 2, 4);//from index 2 upto 4 characters are copied to s1 from s

    // compare string s and s1
    if(s1.compare(s) == 0){
        cout << s1 << " is equal to " << s << endl;
    } else {
        cout << s1 << " is not equal to " << s << endl;
    }

    // substring from index 1 in s upto 4 characters is stored in s2
    string s2 = s.substr(1, 4);
    cout<< s2 << endl; // prints ARIK
}
```

3. Pair Class

The Pair Class contains a pair of homogeneous or heterogeneous values. It can be understood as a container in which the pair of values are identified by **“first”** and **“second”** and can be of different or same data type.

Pair class is present in the standard template library of C++, and can be included in any file using **#include<utility>**

Syntax for creating a pair class:

pair < first_data_type, second data_type > p(first, second);

Example :

pair < int, char > p(1, ‘a’);

To access the elements, we use variable name followed by dot operator followed by the keyword first or second:

cout << p.first << " " << p.second << endl;

```
#include<iostream>
#include<utility>

using namespace std;

int main(){

    // various ways to initialize a pair

    pair<int, char> p;
    p = make_pair(2, 'b');

    pair<int, int> p1(3, 4);

    pair<int, char> p2(1, 'a');

    // accessing elements inside a pair using first and second
```

```
    cout << p.first << " " << p.second << endl; // output is 2 b
    cout << p1.first << " " << p1.second << endl; // output is 3 4
    cout << p2.first << " " << p2.second << endl; // output is 1 a

    return 0;
}
```

4. Set

Set is a container which contains unique elements. The element itself is identified by its value. The value of the element cannot be modified once it is added to the set, though it is possible to remove and add the modified value of that element.
A set can be used in a problem where we have to know whether the element is present or not or suppose we have an array = [1, 2, 3, 4, 5, 6, 5]. Here the number 5 is occurring two times in the array but when we make a set using this array the occurrence of the number 5 is only once just like all the other numbers / elements.

Set is present in the standard template library of C++, and can be included in any file using **#include<set>**

Some functionalities of Set include:

- **begin()** – Returns an iterator to the first element in the set.
- **end()** – Returns an iterator to the theoretical element that follows the last element in the set.
- **size()** – Returns the number of elements in the set.
- **empty()** – Returns whether the set is empty.
- **find()** - Returns an iterator if the element is present.

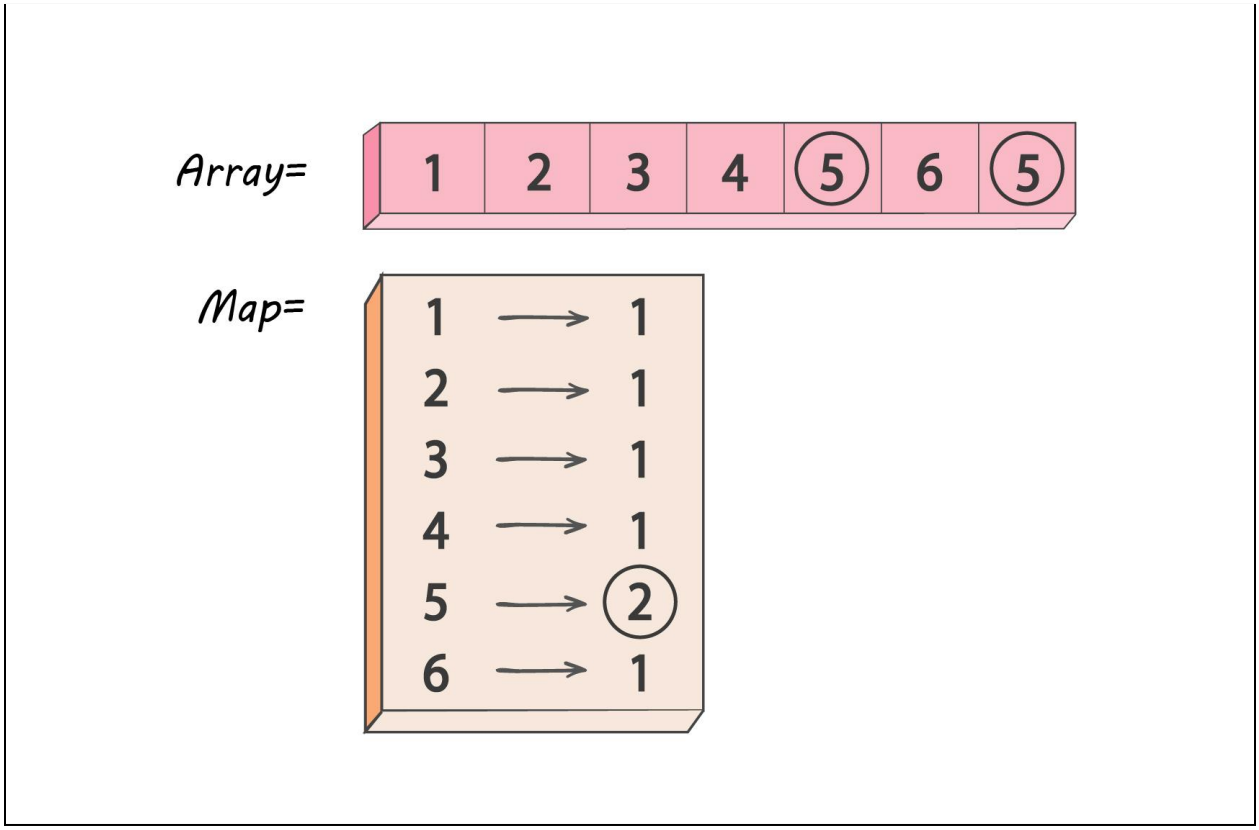
A set is generally implemented using a balanced binary search tree which takes **O(log(N))** time to find an element.

```
#include<iostream>
#include<set>
using namespace std;
```

```
int main(){

    // initializing a set
    set<int> s;
    int arr[] = {1,2,3,4,5,6,5};

    // inserting elements into a set
    for(int i=0; i<7; i++){
        s.insert(arr[i]);
    }
    // creating an iterator to traverse through the set
    set<int>::iterator it;

    for(it=s.begin(); it!=s.end(); it++){
        cout << *it << endl;
    }
    // finding elements in set
    if(s.find(7) == s.end()){
        cout << "Element not found" << endl;
    } else {
        cout << "Element found" << endl;
    }
    return 0;
}
```

5. Map

Consider the previous example used in sets where we made a set using the **array = [1, 2, 3, 4, 5, 6, 5]**. Now if we want to store the frequency of each element, we can not do that by using a set. Therefore, we use a map that stores their frequencies like a **key-value pair in addition to storing unique elements.**



Map is present in the standard template library of C++, and can be included in any file using **#include<map>**

Syntax to make a map:

```
map < int, int > M;
```

Some functionalities of Set include:

- **begin()** – Returns an iterator to the first element in the map
- **end()** – Returns an iterator to the theoretical element that follows last element in the map
- **size()** – Returns the number of elements in the map
- **empty()** – Returns whether the map is empty
- **pair insert(keyValue, mapValue)** – Adds a new element to the map

The **keyValue** and **mapValue** can be accessed using **M -> first** and **M -> second.**

A Map is generally implemented using a balanced binary search tree which takes **O(log(N))** time to do operations.

```
#include<iostream>
#include<map>

using namespace std;

int main(){


    int arr[] = {1,2,3,4,5,6,5};
    map<int, int> m;

    for(int i=0; i<7; i++){
        m[arr[i]] = m[arr[i]] + 1;
    }

    map<int, int>::iterator it;

    for(it=m.begin();it!=m.end();it++){
        cout<<it->first << " :" << it->second<<endl;
    }
    cout<<endl;
    m.erase(1);
    for(it=m.begin();it!=m.end();it++){
```

```
        cout<<it->first << " :" << it->second<<endl;
    }
}
```

6. unordered_map

A normal Map is implemented using a balanced binary search tree which takes **O(log(N))** time to perform operations like insert , search etc but an **unordered_map** is implemented using a hash table which takes **O(1)** time in average case to perform operations like insert, search, etc. However, in the worst case scenario it takes **O(N)** time.
unordered_map is present in the standard template library of C++, and can be included in any file using **#include<unordered_map>**

```
#include<iostream>
#include<unordered_map>

using namespace std;

int main(){


    int arr[] = {1,2,3,4,5,6,5};
    unordered_map<int,int> m;

    for(int i=0;i<7;i++){
        m[arr[i]] = m[arr[i]]+1;
    }

    unordered_map<int,int>::iterator it;
    for(it=m.begin();it!=m.end();it++){
        cout<<it->first << " :" << it->second<<endl;
    }
    cout<<endl;
    m.erase(1);
    for(it=m.begin();it!=m.end();it++){
        cout<<it->first << " :" << it->second<<endl;
    }

}
```

STL Functions

While working on a problem that requires some basic tasks like sorting, searching, etc we are not required to make a separate functionality to achieve them. Rather, these functionalities are already present in C++ STL.

1. sort()

The sort function in STL is given a start pointer and an end pointer and it sorts any array in **O(log(N))** time complexity.
It can be used in any file using **#include<algorithm>.**

For example, arr = [1, 3, 2, 5, 7, 6]

sort(arr, arr+6);	**Sorts the array in ascending order**
sort(arr, arr+6, greater<int>);	**Sorts the array in descending order**

```
int main(){

    int arr[] = {1, 3, 2, 5, 7, 6};
    sort(arr, arr+6);
    for(int i=0; i<6; i++){
        cout << arr[i] << " ";
    }

    return 0;
}
```

We can also sort the array of objects on the basis of one of the properties of a class or structure to whom the object belongs to. For that we are required

to pass a **compare function** along with other parameters and have to define this compare function.

```
#include<iostream>
#include<algorithm>

using namespace std;

// creating a structure Interval
// with a start time and end time
struct Interval{
    int st;
    int et;
};
// compare function to pass in the sort function
// to sort on the basis of start time
bool compare(Interval i1,Interval i2){
    return i1.st < i2.st; // > for descending
}

int main(){

    Interval arr[] = {{6, 4}, {3, 4}, {4, 6}, {8, 13}};
    sort(arr, arr+4, compare);

    for(int i=0; i<4; i++){
        cout << arr[i].st << " : " << arr[i].et << endl;
    }

    return 0;
}
```

2. binary_search

Binary search can be applied only on a sorted array and its time complexity is **O(log(N)).**

Syntax : **binary_search(arr, arr+n, element to be searched);**

3. lower_bound

lower_bound() returns an iterator pointing to the first element in the range [first, last). The function returns the index of the next smallest number just greater than or equal to that number we are searching for. If there are multiple values that are equal to the element to be found, lower_bound() returns the index of the first such value.

Syntax : **lower_bound(arr, arr+n, element);**

Examples:

1. **Input: 10 20 30 40 50**
   Output: lower_bound for element 30 at index 2

2. **Input: 10 20 30 40 50**
   Output: lower_bound for element 35 at index 3

4. upper_bound

upper_bound() returns an iterator pointing to the first element in the range [first, last) that is greater than the element that needs to be found, or last if no such element is found. The elements in the range shall already be sorted or at least partitioned with respect to the element.

Syntax : **lower_bound(arr, arr+n, element);**
Examples :

1. **Input : 10 20 30 30 40 50**
   Output : upper_bound for element 30 is at index 4

2. **Input : 10 20 30 40 50**
   Output : upper_bound for element 45 is at index 4

```
int main(){

    int arr[] = {1,3,2,5,7,6};

    sort(arr,arr+6);

    for(int i=0;i<6;i++){
        cout<<arr[i] << " ";
    }
    cout<<endl;
    cout << binary_search(arr,arr+6,2) << endl;

    cout<<lower_bound(arr,arr+6,3) - arr << endl;

    cout<<upper_bound(arr,arr+6,3) - arr << endl;

    return 0;
}
```

STL Examples

1. Remove Duplicates from Array

Input Format :

Line 1: Contains the size of array

Line 2: M integers which are elements of the array separated by space

Output :

Resultant array

Sample Input :

7

2 4 3 3 5 3 4

Sample Output :

2 3 4 5

Explanation :

Approach 1 : This problem can be easily solved using a Set. We will first make a set using the inbuilt STL and a **result** array. Then we will traverse through our input array and start adding the elements in the set. While traversing the input array if we encounter any duplicates, that is if the element has already been added in the set , then we will not add the element to the result array otherwise we will.

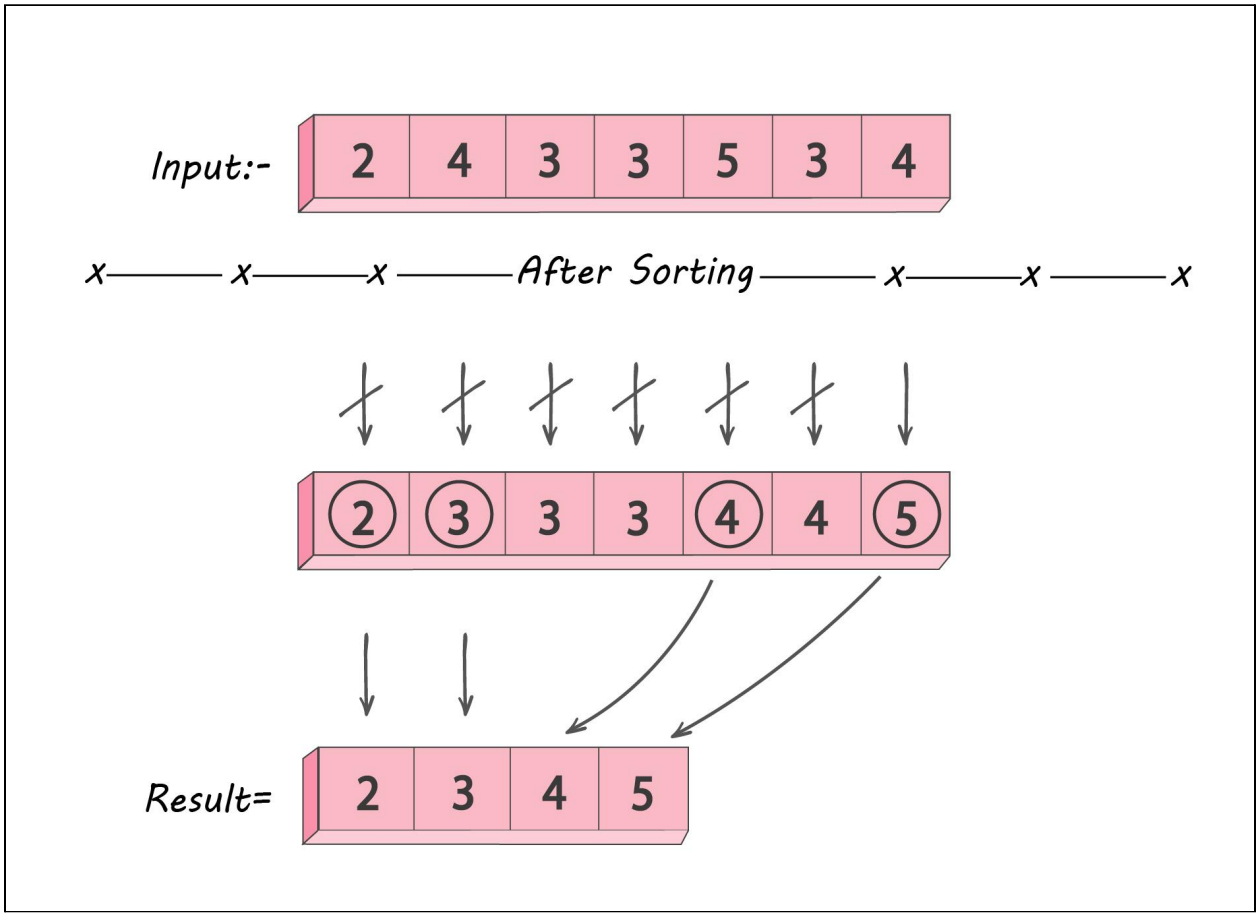
```
vector<int> removeDuplicates(vector<int> input){
```

```
    set<int> s;
    vector<int> result;

    for(int i=0; i<input.size(); i++){

        if(s.find(input[i]) == s.end()){
            s.insert(input[i]);
            result.push_back(input[i]);
        }
    }

    return result;

}
```

Time Complexity: O(N)
Space Complexity: O(N)

Approach 2 : Sort the elements in the input array so that all the duplicate elements will be together. Then traverse through the array and compare elements with the element present at the previous index; if they are equal do not add them in the result array otherwise add them.

For example :




```
vector<int> removeDuplicates(vector<int> input){
    vector<int> result;
    sort(input.begin(), inpt.end());

    result.push_back(input[0]);

    for(int i=1; i<input.size(); i++){

        if(input[i] != input[i-1]){
            result.push_back(input[i]);
        }
    }
    return result;
}
```

Time Complexity : O(n)

2. First Non Repeating Character in a String

If there is no repeating character then print the first character of the string.

Input Format :

Line 1: A String

Output :

First non repeating character

Sample Input :

aDcadhc

Sample Output :

D

Explanation : This problem can be solved using a **map** because with a map we can store the elements along with their frequencies.

```
char nonRepeatingCharacter(string str){

    map<char, int> frequency;
    for(int i=0; i<str.length(); i++){
        char currentChar = str[i];
        frequency[currentChar++];
    }

    for(int i=0; i<str.length(); i++){
        if(frequency[str[i]] == 1){
            return str[i];
        }
    }
    // if there is no unique character in the string
    return str[0];
}
```