# Class-4-CustomDataType-CustomSorting-Set-Map

1. Set and Map
2. Custom Data Types
3. Sorting and Custom Sorting
4. Bit Manipulation

## PSEUDO-CODE/FUNCTIONS FOR SET

### Properties:

* The set stores the elements in **sorted order**.
* Set items are not stored in **linear data structure**.
  + The values in a set are **unindexed**.
* All the elements in a set have **unique** values.
* The value of the element cannot be modified once it is added to the set, though it is possible to remove and then add the modified value of that element. Thus, the values are **immutable**.
* Sets follow the **Binary search tree implementation**.
  + Can also be implemented using **Hash Table [Array]**.

1. Declare and Instantiate a Set

*set*<int> s; // TC: 1

1. Insert an Item in Set // TC: logN

*s*.insert(5);

*s*.insert(10);

*s*.insert(20);

*s*.insert(10);

1. Remove an item from set // TC: logN

s.*erase*(5); // Erase the element 5.

auto it = s.*begin*();

++it;

s.*erase*(it); // Erase using iterator.

1. Count number of elements

s.*size*();

1. Empty or Not

s.*empty*();

1. Check if the element exists in the set or not. // TC: logN

// 'find()' returns an iterator to the element '40' in the set if found,

// else returns the iterator to end.

s.*erase*(s.*find*(40));

1. Iterate over all the elements of set. // TC: N

for (auto i : s)

*cout* << i << "\n";

**[PROBLEM]** Given an array of integers, count the number of unique absolute values.

A= {2, 4, -4, 2, -3, 4, 1, 9, 7, -1}

ANS: 6

EXPLANATION

1, 2, 3, 4, 7, 9

CODE:

int arr[] = { 2, 4, -4, 2, -3, 4, 1, 9, 7, -1 };

*set*<int> s;

for (auto i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)

s.*insert*(*abs*(arr[i]));

*cout* << s.*size*() << "\n";

**[PROBLEM]** Given an array of N elements, return only those elements that occur one time

But the order must be the same as that in the original array.

int A[] = { 2, 3, 2, 4, 5, 3, 9, 1, 6, 6, 2, 4 };

**ANS**

5 9 1

THE FUNCTION PROTOPTYE

*vector*<int> occurringOnlyOnce(*vector*<int>& vec) {

}

**APPROACH 1:**

For every element check in the rest of the array, if an element is found at any other place than

the current index then ignore it. Otherwise Add it to the answer.

**TC:** n2: O(n2)

**SC:** 3 = 0(1)

**CODE:**

*vector*<int> occurringOnlyOnce(*vector*<int>& vec) {

*vector*<int> ans;

int len = vec.*size*();

for (auto i = 0; i < len; i++)

{

bool found = false;

for (auto j = 0; j < len; j++)

if (i != j && vec[j] == vec[i])

{

found = true;

break;

}

if (!found) ans.*push\_back*(vec[i]);

}

return ans;

}

**APPROACH 2**:

TC: n + nlogn + nlogn = O(nlogn) // n – Copying, nlogn – sorting, nlong – searching.

SC: n + 2 = O(n) // B to create a copy. ‘ans’ vector for returning.

**PSEUDO CODE:**

1. list<int> ans;
2. Copy array A into another array B.
3. Sort array B in asc order.
4. for i=[0, n-1]
   1. int freq = findFreqInSortedArray(B, A[i]); // Use Binary Search
   2. if(freq == 1) ans.insert(A[i])
5. return ans;

CODE:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace *std*;

int binarySearch(*vector*<int>&B, int low, int high, int a)

{

if (high < low) return -1;

int mid = (low + high) >> 1; // (low + high)/2 or low + ((high-low)/2)

if (B[mid] == a) return mid;

else if (B[mid] > a) return binarySearch(B, low, mid - 1, a);

else return binarySearch(B, mid + 1, high, a);

}

int findFrqeInSortedArray(*vector*<int>& B, int a)

{

int len = B.*size*();

int ind = binarySearch(B, 0, len - 1, a);

if (ind == -1) return 0;

int count = 1;

int left = ind - 1;

while (left > 0 && B[left] == a)

count++, left--;

int right = ind + 1;

while (right < len && B[right] == a)

count++, right++;

return count;

}

*vector*<int> occurringOnlyOnce(*vector*<int>& A) {

*vector*<int> B(A.*size*());

*vector*<int> ans;

*copy*(A.*begin*(), A.*end*(), B.*begin*());

*sort*(B.*begin*(), B.*end*());

int len = A.*size*();

for (auto i = 0; i < len; i++)

{

int freq = findFrqeInSortedArray(B, A[i]); // Use Binary Search

if (freq == 1) ans.*push\_back*(A[i]);

}

return ans;

}

int main(void)

{

*vector*<int> A = { 2, 3, 2, 4, 5, 3, 9, 1, 6, 6, 2, 4 };

for (auto i : occurringOnlyOnce(A))

*cout* << i << " ";

return 0;

}

## PSEUDO-CODE/FUNCTIONS FOR MAP/DICTIONARY

* Pair is a class having two members named first and second.

template<typename T, typename U>

class pair{

T fimt;

U second;

};

1. Declare and Instantiate a Map.
   1. *map*<int, int> m; // Declaring a map of int key and int value // TC: 1
2. Insert an Item in Map // TC: **LogN**

m.*insert*(*pair*<int,int>(5, 6)); // 5 -> key, 6 —> value

m.*insert*(*make\_pair*(2, 3)); // make\_pair() is less verbose than std::pair.

m[3] = 5; // key = 3, value = 5, c++, py, c#, JS, JAVA does not support.

1. Remove an item from map // TC: **LogN**

m.*erase*(3); // key is the only param given, it removes both key and value.

1. Count of Elements // TC: 1

m.*size*();

1. Empty Or Not

m.*empty*(); // Returns boolean answer.

1. Check if a Key is there or not in the Map. // TC: **LogN**

if (m.*find*(5) != m.*end*())

*cout* << m[5]; // Prints value 6.

1. How to iterate over all the values of a Map. // Map Does not support INDEXING. // TC: N

for (*auto* x : m) // for each item X in map m

*cout* << x.*first* << " " << x.*second*;

**APPROACH 3**: Using MAP

TC: n\*logn + n\*logn = O(nlogn) // n -> iterating over array + logn -> map insert.

SC: 1(for i) + number of unique elements in A = 1 + worst case n

(all distinct elements in A) = O(n)

Assumption: Although map pair has two fields, we are counting it as 1.

CODE:

*vector*<int> occurringOnlyOnce(*vector*<int> a)

{

*vector*<int> ans;

*map*<int, int> freq;

for (auto i = 0; i < a.*size*(); i++)

freq[a[i]] += 1;

for (auto i = 0; i < a.*size*(); i++)

if (freq[a[i]] == 1) ans.*push\_back*(a[i]);

return ans;

}

EXERCISES:

<https://leetcode.com/problems/sum-of-unique-elements/>

<https://leetcode.com/problems/unique-number-of-occurrences/>

<https://leetcode.com/problems/4sum-ii/>

**Map and Set are useful for solving SEARCH PROBLEMS.**

## CUSTOM DATA TYPES and CUSTOM SORTING

Questions related to intervals…

<https://practice.geeksforgeeks.org/problems/minimum-platforms-1587115620/1>

<https://practice.geeksforgeeks.org/problems/n-meetings-in-one-room-1587115620/1>

<https://leetcode.com/problems/merge-intervals/>

Given list of START and END times for your meetings

|  |  |
| --- | --- |
| 9:00 AM | 11 :00 AM |
| 10:30 AM | 12:00 PM |
| 7:00 AM | 8:00 AM |
| 12:50 PM | 1:00 PM |

Question 1: Are there any conflicting meetings?

Question 2: Give busy slot groups.

Question 3: Min meeting rooms required => Max Overlapping Intervals Count

Question 4: Find out free slot groups.

class Meeting {

*string* startTime;

*string* endTime;

};

*vector*<Meeting> allMeetings;

* Built-In Sort provides us an easy way to provide comparators for our own DATA TYPES.

Or

* Built-In Sort provides us an easy way to provide a comparator for built-in DATA TYPES if we want a different order of sorting.

INTRODUCTION:

## Bubble Sort

1. Compare Adj Elements in Every Iteration.
2. Each Iteration Bubbles One Element to its right position (MAX).

// SORT IN ASCENDING

void bubbleSort(*vector*<int> &a) { // Passed By Ref, Use & in c++

for (int iter = 0; iter <= a.*size*()-1; iter++) {

for (int i = 0; i < a.*size*()-1; i++) // Lenient

if (a[i] > a[i + 1]) { // Adjacent Only

int tmp = a[i]; a[i] = a[i + 1]; a[i + 1] = tmp;

}

}

}

// SORT IN DESCENDING

void bubbleSort(*vector*<int> &a) { // Passed By Ref, Use & in c++

for (int iter = 0; iter <= a.*size*()-1; iter++) {

for (int i = 0; i < a.*size*()-1; i++) // Lenient

if (a[i] < a[i + 1]) { // Adjacent Only

int tmp = a[i]; a[i] = a[i + 1]; a[i + 1] = tmp;

}

}

}

GENERALIZATION OF **COMPARE** FUNCTION

template<typename T>

bool compareType1(T a, T b) {

return a < b;

}

template<typename T>

int compareType2(T a, T b) {

return a - b;

}

template<typename T>

void bubbleSort(*vector*<T> &a) {

for (int iter = 0; iter <= a.*size*() - 1; iter++) {

for (int i = 0; i < a.*size*() - 1; i++) // Lenient

if (compareType1(a[i + 1], a[i])) { // Adjacent Only

int tmp = a[i]; a[i] = a[i + 1]; a[i + 1] = tmp;

}

}

}

# Custom Sorting - User Defined Data Types

* Many Coding Problems Require us to define a data type which is built using primitive data types. Data Types that are not built-in into the language and we need to define/create ourselves are called **User Defined Data Types or Custom Data Types**.
* For Example

*Given a List of Students (each having* ***name, age, gender****), Print the name of student who is* ***eldest****.*

* In Above question, if we notice, **STUDENT** is not in-built data type of a language. We need to define that data type (custom or user defined). And then deal with it.

We MUST be comfortable to achieve following for a **custom/user-defined data type**:

1. How to declare/define a custom data type.
2. How to declare a variable/object of this data type.
3. How to declare array of custom data types.
4. How to pass custom data type to a function.
5. How to pass array of custom data type to a function.
6. How to return custom data type from a function.
7. How to return an array of custom data type from a function
8. How to sort array of custom data type. ----- THIS IS MOST IMPORTANT THING TO LEARN.

## How to declare/define a custom data type.

// CREATING OUR OWN DATA TYPE

class Student {

*string* name;

int age;

char gender;

public:

Student() { }

Student(const *string*& n, const int& a, const char& g) {

name = n;

age = a;

gender = g;

}

*string* getName(void) const { return name; }

int getAge(void) const { return age; }

char getGender(void) const { return gender; }

};

## How to declare a variable/object of this data type.

Student aStudent;

## How to declare array of custom data types.

Student anArray[10]; // Declaring

Student arrOfStudents[] = { {"SONAM", 11, 'F' },

{"RAJAT", 20, 'M' },

{"SAGAR", 21, 'M' }};

## How to pass custom data type to a function.

void printStudent(const Student& s) {

*cout* << s.getName() << " " << s.getAge() << " " << s.getGender() << "\n";

}

## How to pass array of custom data type to a function.

void printStudents(const *vector*<Student>& arr) {

for (auto i : arr)

printStudent(i);

}

## How to return custom data type from a function.

Student CreateStudent(*string* name, int age, char gender)

{

Student newStudent(name, age, gender);

return newStudent;

}

## How to return an array of custom data type from a function.

*vector*<Student> CreateStudents()

{

*vector*<Student> listOfStudents = {{"SONAM", 11, 'F' },

{"RAJAT", 20, 'M' },

{"SAGAR", 21, 'M' } };

return listOfStudents;

}

## How to sort array of custom data type. ----- THIS IS MOST IMPORTANT THING TO LEARN.

* Let’s learn sorting the custom data type with a problem.

**[PROBLEM]** Given details of N students (NAME AGE GENDER) format, read them in given order and print output in …

1. REVERSE ORDER.
2. ALPHABETICAL SORTED ORDER.
3. SORTED ACCORDING TO AGE.

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace *std*;

// CREATING OUR OWN DATA TYPE

class Student {

*string* name;

int age;

char gender;

public:

Student() { }

Student(const *string*& n, const int& a, const char& g) {

name = n;

age = a;

gender = g;

}

*string* getName(void) const { return name; }

int getAge(void) const { return age; }

char getGender(void) const { return gender; }

};

void printStudent(const Student& s) {

*cout* << s.getName() << " " << s.getAge() << " " << s.getGender() << "\n";

}

void printStudents(const *vector*<Student>& arr) {

for (auto i : arr)

printStudent(i);

}

void printReverse(const *vector*<Student>& arr) {

for (int i = arr.*size*() - 1; i >= 0; i--) {

printStudent(arr[i]);

}

}

bool compareName(const Student& a, const Student& b)

{

return a.getName() < b.getName();

}

/\*

5

ARIVIND 14 M

DARSHAN 13 M

CHANDANA 12 F

BHEEM 11 M

ESHWAR 10 M

\*/

int main(void)

{

*ios\_base*::*sync\_with\_stdio*(false);

*cin*.*tie*(nullptr);

*cout*.*tie*(nullptr);

int n;

*cin* >> n;

*vector*<Student> arr;

*string* name;

int age;

char gender;

for (int i = 0; i < n; i++) {

*cin* >> name >> age >> gender;

arr.*push\_back*(Student(name, age, gender));

}

*cout* << "Printing the input list of students.\n";

printStudents(arr);

*cout* << "Printing in reverse order.\n";

printReverse(arr);

*sort*(arr.*begin*(), arr.*end*(), [](Student& a, Student&b) -> bool

{ return a.getName() < b.getName(); }); // ALPHABETICAL SORTING.

*cout* << "Printing as per ALPHABETICAL SORTING.\n";

printStudents(arr);

*sort*(arr.*begin*(), arr.*end*(), [](Student& a, Student& b) -> bool

{ return a.getAge() < b.getAge(); }); // ACCORDING TO AGE

*cout* << "Printing ACCORDING TO AGE.\n";

printStudents(arr);

return 0;

## }

**[PROBLEM]** Given few color names, SORT colors based on their string length (shorter first,

longer later). If two colors are of same length, put the Lexicographically bigger one first.

*string* colors[] ={ "green", "red", "pink", "grey", "yellow", "purple", "blue", "black" };

**OUTPUT:**

red pink grey blue green black yellow purple

**CODE:**

bool compare(*string* a, *string* b)

{

if (a.*length*() < b.*length*())

return a.*length*() < b.*length*();

else

return a.*compare*(b) == 1 ? true : false;

}

template<typename T>

void bubbleSort(*vector*<T> &a) {

for (int iter = 0; iter <= a.*size*() - 1; iter++) {

for (int i = 0; i < a.*size*() - 1; i++) // Lenient

if (!compare(a[i], a[i + 1])) { // Adjacent Only

T tmp = a[i]; a[i] = a[i + 1]; a[i + 1] = tmp;

}

}

}

**[PROBLEM]** Given a list of integers, sort them based on their reverse values, if two numbers have the same reverse values, put the bigger one first.

*vector*<int> A = { 123,41,70,9,124,90 };

**ANS:**

70, 90, 9, 41, 123, 124

int reverse(int a) {

int ra = 0;

while (a != 0) {

ra = ra \* 10 + a % 10;

a /= 10;

}

return ra;

}

bool compare(int a, int b) {

int ra = reverse(a); int rb = reverse(b);

if (ra == rb) return a < b;

return ra > rb;

}

template<typename T>

void bubbleSort(*vector*<T> &a) {

for (int iter = 0; iter <= a.*size*() - 1; iter++)

for (int i = 0; i < a.*size*() - 1; i++)

if (compare(a[i], a[i + 1])) {

int tmp = a[i]; a[i] = a[i + 1]; a[i + 1] = tmp;

}

}

## SORTING BASED PROBLEMS ON LEETCODE

<https://leetcode.com/problems/sort-an-array/>

<https://leetcode.com/problems/find-target-indices-after-sorting-array/>

<https://leetcode.com/problems/custom-sort-string>

<https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/>

<https://leetcode.com/problems/reorder-data-in-log-files/>

<https://leetcode.com/problems/sort-colors/>

<https://leetcode.com/problems/sort-characters-by-frequency/>

<https://leetcode.com/problems/sort-array-by-increasing-frequency/>

<https://leetcode.com/problems/sort-array-by-parity/>

<https://leetcode.com/problems/sort-array-by-parity-ii/>

<https://leetcode.com/problems/sort-the-matrix-diagonally/>

<https://leetcode.com/problems/rearrange-words-in-a-sentence/>