Q. Design a data structure that perform following operations.

- bool search(x)
- void insert(x)
- void delete(x)
- int getFloor(x): ~~returns value smaller~~ return largest value smaller than or equal to x. If x is not present and smallest return INT_MIN.
- int getCeiling(x): return smallest value greater than or equal to x. If x is not present and largest return INT_MAX.

→ Now we have to choose a DS which can easily perform these operations and we have given containers.

- vector
- pair
- list
- forward list
- stack
- queue
- priority_queue
- dequeue
- set

as we can see, first three operations are easily implemented by many containers but for last two we need containers in which data is already sorted. so, we will use set for this.

ex. -

~~insert(10), insert(20), insert(15), insert(5), insert(25), search(15), search(15)~~

insert(10)	{10} ← set
insert(20)	{10, 20}
insert(15)	{10, 15, 20}
insert(5)	{5, 10, 15, 20}
insert(25)	{5, 10, 15, 20, 25}
search(15)	→ true
delete(15)	{5, 10, 20, 25}
search(15)	→ false
getfloor(6)	→ 5 // as 5 is smaller than 6

getfloor(5) → 5 // as 5 is itself present

getceling(6) → 10 // as 10 is greater than 6 and nearest.

getceiling(25) → 25

getceiling(100) → INT_MAX // as there is no 100 in set and neither any greater element

getfloor(1) → INT_MIN

Code -

```
Set<int> S;

void Insert(int x){ S.insert(x); }

bool search(int x){ return (S.find(x)!=S.end()); }

void delete(int x){ S.erase(x); }

int getceiling(int x){
    auto it=S.lower_bound(x);
    if(it==S.end()) return INT_MAX;
    else return (*it);
}
```

```
int getfloor(int x){
 auto it = s.lower_bound(x);

 if(it == s.begin()) // searching first ele.
 {                   //or lowest element
   if(*it == x)      //which is not present
     return(*it);
   else
     return INT_MIN;
 }

 else
   {
     if(it != s.end() && *it == x)
                return *it;
     it--;
     return *it;
   }
}
```

Q. First greater height -

We have given vector 'v' containing height of students. For every students height we have to find smallest height greater than given current height in its left side of it, and store it in another vector.

- And if There is no greater height then current height print '-1' for that.

ex. -

V = {3, 2, 5, 4, 1}

Output = -1, 3 -1 5 2

explanation - For 3 there is no element in left so -1, for 2 there is 3 which is in his left and greater so 3, for 5 no one in his left greater than 5 so -1. for 4, 5 is ans for 1 all 2, 3, 4 and 5 are in left and greater than 1 but smallest is 2 so 2.

approach -

As we can see if we sort the array and find upper bound of numbers then we get some of ans right but for some cases like.

'3' in

V={3, 2, 5, 4, 1}

we get wrong result as, for 3 there is no element in left but if we sort whole array/vector at once we get '4' as upperbound of 3.

So, we don't need to sort whole vector at once but we need a container such that we input each no one by one from left and it automatically store it in sorted order.

By doing so, if we enter '3' and find upper_bound it give end() iterator and we will put '-1' for that and as we input numbers one by one in that container it will automatically sort it so, we no need of

to sort the containers, and by using upper_bound we can find greater elements than given element in left of it.

So, The only container we know of who can do so is 'set'.

Code-

```
vector<int> greater_height(vector<int>v){
    vector<int> output;
    set<int> s;
    for(int i=0;i<v.size();i++){
        s.insert(v[i]);
        auto it=s.upper_bound(v[i]);
        if(it==s.end()){
            output.push_back(-1);
        }
        else{
            output.push_back(*it);
        }
    }
    return output;
}
```

Q. Ceiling on right -

We have been given an array and we have to find ceiling of each number on right of it.

if v[i] is a number, we have to find another number in left greater than or equal to v[i] if there is any, and if not put -1 for that.

ex. -

arr[] = {10, 100, 200, 30, 120, 120}

O/p - {30, 120, -1, 120, 120, -1}

approach - we will use set container and will insert one element at a time from right side of array. we find lower_bound of given element. if it's s.end() put -1 for that if (*it) == v[i], put v[i] in output else also put v[i] in output

Code-

```
void ceil_right(int arr[], int n){
    set<int> s;
    int res[n];

    for(int i=n-1; i>=0; i--){
        auto it = s.lower_bound(arr[i]);
        if(it == s.end())
            res[i] = -1;
        else
            res[i] = *it;
        s.insert(arr[i]);
    }
    for(int i=0; i<n; i++){
        cout<<res[i]<<" ";
    }
}
```

8. Pair Sum existance -

You are given an array of distinct integers of size n and a sum value. You have find if two numbers in array exist or not such that sum of these numbers is equal to sum value. Return 1 if yes or 0 if not.

ex. -

1 3 2 6 4 5 8 9 10 7 , sum = 14

O/p - 1 (10+4)

approach -

We first need to sort an array or we can use set as all numbers are distinct (if not distinct use multiset).

now, make two iterators pointing to first and last element of set. (smallest and greatest).

If sum of these two iterators is greater than sum decrease right iterator

If sum of these two iterator values is less than sum increase left side iterator and if sum of these two iterator values is = sum return 1.

Code-

```
int sumexists(int arr[], int n, int sum){

    Set<int> s(arr, arr+n);
    auto it1 = s.begin();
    auto it2 = s.end();
    it2--;
    while(it1 != it2){
      if(*it1 + *it2 > sum)
          it2--;
      else if(*it1 + *it2 < sum)
          it1++;
      else
          return 1;
    }
    return 0;
}
```

Q. Design a Data Structure for item and their prices which can perform these operations. (no duplicate)

- add(price, item)
- find(price)
- print_sorted()
- print_Greater(price) :- prints all items with price greater than given price.
- print_Smaller(price) :- prints all items with price smaller than given price.

=> We will use set containers for this purpose -

```
map<int, string> m;
• void add (string name, int price){
      m.insert({price, name});
  // m[price] = name;
  }

• string find (int price){
    auto it = m.find(price);
    if(it == m.end()) return "";
    else return it->second;
  }
```

- void print_sorted(){
  for(auto x:m)
    cout<<x.second<<" "
      <<x.first<<endl;
}

- void print_greater(int price){
  auto it=m.upper_bound(price);
  while(it!=m.end())
  {
    cout<<it->second<<" "
      <<it->first<<endl;
    it++;
  }
}

- void print_smaller(int price){
  auto it=m.lower_bound(price);
  for(auto it2=m.begin(); it2!=it; it2++){
    cout<<it2->second<<" "
      <<it2->first<<endl;
}

Q. Count greater element for every array element.

->

We have given an array and we have to find number of greater element from given element.

ex. -

arr[] = {2, 8, 5, 12, 1, 8}

O/p = 4, 1, 2, 0, 5, 1

There is a naive approach of $O(n^2)$ for each element in array, we search whole array and count greater element

but efficient approach is using map of $O(n \log n)$.

approach -

We will make a map with keys as array element and values as count of their occurance.

By doing so we have occurance of every digit in incresing order of value of array element.

arr[] = {2, 8, 5, 12, 1, 8}

map m = {{1, 1}, {2, 1}, {5, 1}, {8, 2}, {12, 0}}

Now, as you can see, for any element in map, no of greater element for that element is sum of count of element in it's right.
for example -

for 12 there is no ele. in right so zero.
for 8 there is one element and it's value part is 1 so 1

for 5, 8 and 12 is in it's right and sum of their value part is 3 so 3.

So, we have to change the map so that in each key value pair value contain number greater element.
To do so sum put sum of value part to the right of it, in current value.

we make variable

int total_val = 0;

and run a loop from .begin() to .end() and for each pair in map we

- first store value part of pair in local variable say

  int curr_val = it->second.

- then store total val (up to it right side) into current value.

  it->second = total_val.

- update total value as

  total_value += curr_val.

After doing so we run a loop over array and for each element print it's value part in map.

map after changing.

m = {{1,5}, {2,4}, {5,3}, {8,1}, {12,0}}

Code -

```
Void print_greater_no(int arr[], int n){

    map<int, int> m;
    for(int i=0; i<n; i++){
        m[arr[i]]++;
    }

    int total_val = 0;

for(auto it = m.begin(); it != m.end(); it++){

int cur_val = it->second;
    it->second = total_val;
    total_val += cur_val;
}

for(int i=0; i<n; i++){
    cout<< m[arr[i]] << " ";

}
```

Q. Design a data structure for item and their prices which can perform these operation (duplicates allowed) -

- add(price, item):
- find(price) → print all pair with same price
- print sorted()
- print Greater(price) → print all pair greater than given price.
- print_Smaller(price) → print all pair Smaller price than given price.

→

```
multimap<int, string> m;

void add(int price, string item){
    m.insert({price, item});
}

void find(int price){
  auto it = equal_range(price);
  if(it == m.end()){
    printf("no item \n"); }
  else{
      for(auto ptr=it.first; ptr!=it.second; ptr++)
    {   cout<< ptr -> second<<"  "
            << ptr -> first<<endl;
    }
  }
}
```

```
void print_sorted(){
    for(auto x: m)
        cout<<x->second<<" "
            <<x->first<<endl;
}

void print_greater(int price){
    auto it=m.upper_bound(price);
    if(it==m.end)
        printf("no greater element\n");
    else{
        while(it!=m.end()){
            cout<<it->second<<" "
                <<it->first<<endl;
            it++;
        }
    }
}

void print_smaller(int price){
    auto it=m.lower_bound(price);
    if(it==m.end() || it->first!=price)
    for(auto it2=m.begin();it2!=it;it2++)
        cout<<it2->second<<" "
            <<it2->first<<endl;
}
```

Q. Print distinct element in array.

→

arr[] = {7, 5, 2, 7, 9, 2, 8};

O/P - 7 5 2 9 8

we are considering order doesn't matter.

approach 1:- naive approach $O(n^2)$

for each element arr[i] we traverse in left of it and if arr[i] is not comes before, we print it.

approach 2:- $O(n \log n)$

we use set for this. It will print element in sorted order. We first insert all element of array in set.

```
set<int> S;
for(int i=0 ; i<n ; i++)        -O(n)
    S.insert(arr[i]);           -O(log n)
```

So total T.C. is $O(n \log n)$

approach 3- O(n), using unorded_set.

We input all array element in unorded_set and then print unordered_set it will O(n) as in unordered_set insertion is O(1) operation.

```
unordered_set<int> S;
for(int i=0;i<n;i++)          <- O(n)
    S.insert(arr[i]);          O(1) avg.

for(auto x:S)
    cout<<x<<" ";
```

If we have to print all distinct element of array but in the same order they appear in array.

for this also we can use unordered_set. we check if the given element is not present in unordered_set we print it and insert it in unordered_set.

```
void print(int arr[], int n){
    unordered_set<int> s;

    for(int i=0;i<n;i++){
        auto it = s.find(arr[i]);
        if(it == s.end()){
            cout<<arr[i]<<" ";
            s.insert(arr[i]);
        }
    }
}
```