STL NOTES
BY AADIL

STL Introduction

Date / /
Page

STL

- Containners
- Algorithm

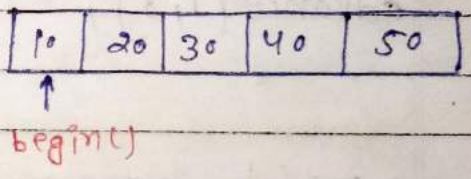
Containers:

- Simple : Pair, Vector, forward_list, List
- Container Adapters : stack, queue, Paiority queue
- Associative : set, map, unordered_set, unordered_map

Algorithms : binary_search, find, reverse, sort ------
..... many more

Iterators

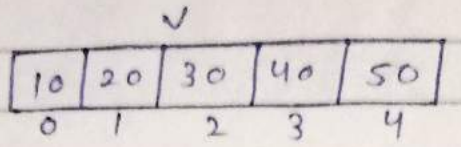
```
#include<vector>

int main()
{
    vector<int> v = { 10,20,30,40,50 };
    vector<int>::iterator i = v.begin(); // auto i;
    cout<< *i ; // 10
    i++;
    cout<< (*i) ; // 20

    i = v.end();
    i--;
    cout<< (*i) ; // 50
}
```



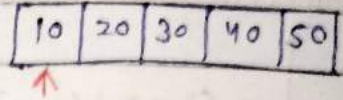
Iterators:

begin()
Prev()
advance()

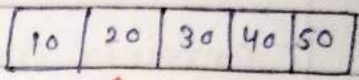
end()
next()



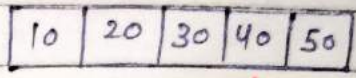
auto i = v.begin();



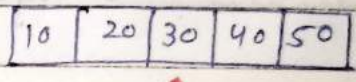
i = next(i);



i = next(i,2);

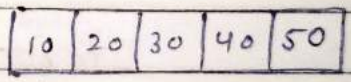


i = Prev(i)

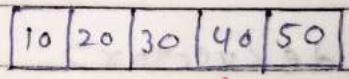


advance()

auto i = v.begin()



advance(i,3);



* next() returns the iterator where as advance modify the same past iterator.

Types of Iterator

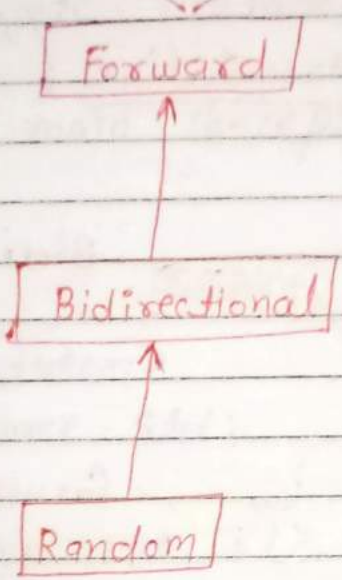
Date / /
Page

can perform only read operation by i++ or ++i

Input

can perform only write operation by i++ or ++i

Output



can perform both operation read and write

can perform read and write in forward as well as backward

all above + random access
can compare two iterator

* These are logical types based on working of iterators. there is no iterators like input iterator, output iterator.

Iterator supported by Containers

Simple

forward_list
list
vector

forward
bidirectional
Random

Associative (These are self balanc Tree

set
map
multimap
multiset

→ Bidirectional

unordered_set
unordered_map

→ forward

Adapters

queue
stack
priority_queue

Do not have iterator

* Adapters are implemented at the Top of Simple Containers.

Template in C++

- write once, use for any data types.
- like macros, processed by compiler but better than macros as type checking is performed.

- two Types :
  * function template : sort, linear search, binarySearch
  * class template : stack, queue, dequeue - - - -

- it is the main concept behind STL.

ex.

Function Template

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

template <typename T>
T mymax(T x, T y)
{
    return (x>y) ? x : y;
}

int main()
{
    cout<< mymax<int>(3,7) << endl;
    cout<< mymax<char>('c','g') << endl;
    return 0;
}
```

Class Template

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

template < typename T>

struct Pair
{
    T x,y;
    Pair (T i, T j)
    {
        x = i;
        y = j;
    }
    T getfirst()
    {
        return x;
    }

    T getsecond();
};

template<typename T>

Pair<T>::getsecond()
{
    return y;
}
```

defining function inside.

Defining function outside

Pair in C++

Date / /
Page

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

using namespace std;

int main()
{
    Pair<int, int> P1(10,20);

    Pair<int, string> P2(10,"GFG");

    cout<<P1.first<<  P1.second<<endl;

    cout<<P2.first<<  P2.second<<endl;
    return 0;
}

// we can also initialize value as follow;

    Pair<int,int> P;
    P={10,20};    or    P=make_pair(10,20);
```

* if we don't provide a value, default constructor initialize '0' as value.
* if we want another value as default we will have to implement over every template.

Comparison of pairs

```
int main()
{
    pair<int,int> P1(1,12), P2(9,12);

    cout<< (P1 == P2) << endl
        << (P1 != P2) << endl;
        << (P1 > P2) << endl;
        << (P1 < P2) << endl;
}
```

output:

```
0
1
0
1
```

* P1 == P2 return 1 only if both values of P1 and P2 are equal.

* <,> ~~only~~ compares the first value if first values are same then compares the ~~third~~ 2nd value.

Date / /
Page

Sort one array acording to other.

input: a[] = [3, 1, 2]
b[] = ['G', 'E', 'K']

output: b[] = ['E', 'K', 'G']

```
void sort( int a[], char b[], int n)
{
    Pair<int, char> P[n];

    for(int i=0; i<n; i++)
        P[i] = { a[i], b[i] };      P[i].first = a[i];
                                    P[i].second = b[i];

    sort(Pa, Pa+n);

    for(int j=0; j<n; j++)
        cout<< P[i].second << " ";
}
```

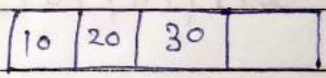
Vectors in C++

Advantage:

- Dynamic size
- Rich library function
- easy to know size
- No need to pass size
- can be returned from a function
- Bydefault initialized with default value.
- we can copy a vector to other.

ex declaration, initialization and traversal.

```
#include<vector>
int main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    for(int i=0; i<v.size(); i++)
        cout<<v[i]<<" ";
    return 0;
}
```



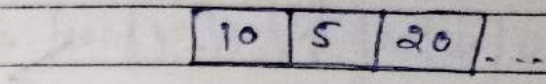
OR

```
v.at(i);
```

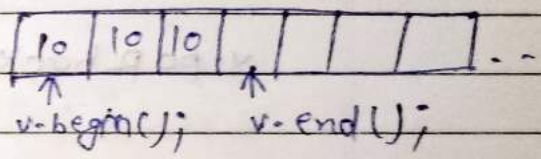
this checks for index out of bound while other does not

Alternate ways

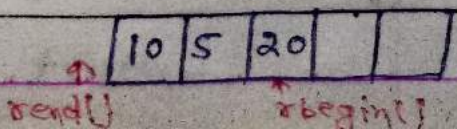
```
int main()
{
    vector<int> v{10,5,20};
    for(int x:v)
        cout<<x<<" ";

    for(int &x:v)          // to change value we need
        x=6;               // to use reference operator
}
```




```
int main()
{
    int n=3, x=10;
    vector<int> v(n,x);

    for(auto it=v.begin(); it!=v.end(); it++)
        cout<<(*it);

    return 0;
}
```



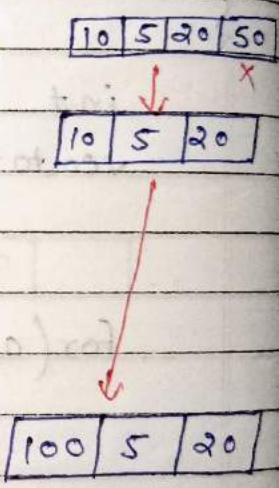

```
int main()
{
    int arr[] = {10,5,20};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> v(arr, arr+n);

    for(auto it=v.rbegin(); it!=v.rend(); it++)
        cout<<(*it);
    return 0;
}
```



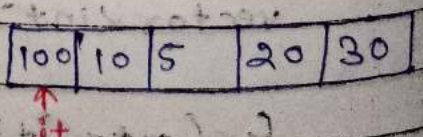
Functions in vector

push_back()
pop_back()
front() > both returns reference..
back()
insert()
erase()
clear()
resize()
empty()

```
int main()
{
    vector<int> v{10,5,20,50};

    v.pop_back();

    cout<<v.front();    // 10
    cout<<v.back();     // 20

    v.front() = 100;
}
```

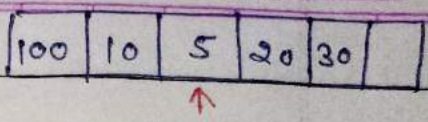



```
int main()
{
    vector<int> v{10,5,20,30};
    auto it = v.insert(v.begin(), 100);
}
```

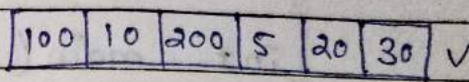


Date / /
Page

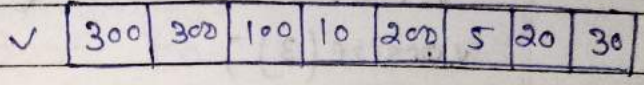
* `v.insert(v.begin()+2, 200);`



⇒



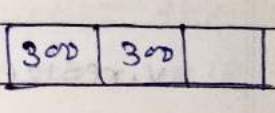
* `v.insert(v.begin(), 2, 300);`




```
vector<int> v2;
```

* `v2.insert(v2.begin(), v.begin(), v.begin()+2);`

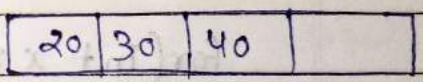
```
}
```



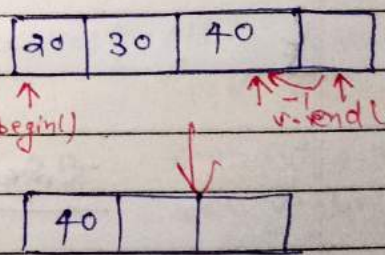

```
int main()
{
```

* `vector<int> v{10,20,30,40};`

`v.erase(v.begin());` →



`v.erase(v.begin(), v.end()-1);`




```
}
```

```
int main()
{
    vector<int> v{10,20,30,50};
```

* `v.clear();`

```
    cout<< v.size();    // 0
    cout<< v.empty();   // true
}
```

resize

```
int main()
{
    vector<int> v{10,5,20,15};
    v.resize(3);

    for(int x:v)
        cout<<x<<" ";          // 10,5,20

    v.resize(5);

    for(int x:v)
        cout<<x<<" ";          // 10 5 20 0 0

    v.resize(8,100);

    for(int x:v)
        cout<<x<<" ";
                               // 10 5 20 0 0 100 100 100
}
```

Time complexity of functions

front(), back(), empty()
begin(), rbegin(), cbegin(), crbegin()
end(), rend(), cend(), crend()
size()
} O(1)

push_back() : O(1) → average.
pop_back() : O(1) → average. { might differ if ... changes

insert()
erase()
resize()
} O(n)

Passing vector to functions

```
void fun1(vector<int> v)
{
    v.push_back(10);
    v.push_back(20);
}

void fun2(vector<int> &v)
{
    v.push_back(10);
    v.pash_back(20);
}

void main()
{
    vector<int> v{5,7,2,9};
    fun1(v);
    for(int x:v)
        cout<<x;        // 5,7,2,9
    fun2(v);
    for(int x:v)
        cout<<x;        // 5,7,2,9,10,20
}
```

if our purpose is to just print the content then use `const vector<int> &v`

this copy each time a value to x

here also we can use `const int &x:v` when vector contain objects

* while calling fun1() all value of vector will be copied. So in case of larger size it may be very costly.
* So recommended to use &v even if you want to just print data

Sort Student by Marks

```
void sortbyMarks(int roll[], int marks[], int n)
{
    vector< pair<int,int>> v;
    for(int i=0; i<n; i++)
        v.push_back({roll[i], marks[i]});

    sort(v.begin(), v.end(), mycmp);

    for(int i=0; i<n; i++)
        cout<<v[i].first<<" "<<v.second<<endl
}

void mycmp(pair<int,int> p1, pair<int,int> p2)
{
    return p1.second > p2.second;
}
```

roll = [101, 103, 105]

marks = [80, 70, 90]

v = { (101,80), (103,70), (105,90) }

After Sorting:

v = { (105,90), (101,80), (103,70) }

Alternate implementation

Date / /
Page

```
void sortbyMarks(int roll[], int marks[], int n)
{
    vector<Pair<int,int>> v;
    for(int i=0; i<n; i++)
        v.push_back({mark[i], roll[i]});

    Sort(v.begin(), v.end(), greater<Pair<int,int>>());

    for(int i=0; i<n; i++)
        cout<<v[i].second<<" "<<v[i].first<<endl;
}
```

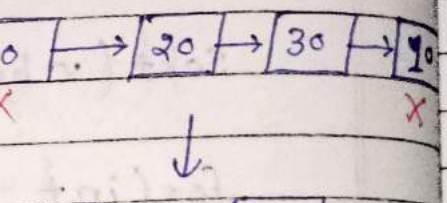
* bydefault sorting is done based on first element in the pair.

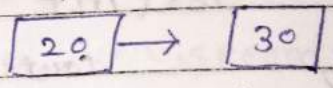
keep Index After Sorting

```
void sortwithindex(int arr[], int n)
{
    vector< pair<int, int> > v;
    for(i=0; i<n; i++)
        v.push_back({arr[i], i});   // or //make_pair(arr[i], i);

    sort(v.begin(), v.end());

    for(auto x : v)
        cout<<x.first<<" "<<x.second<<endl;
}
```

forward_list in C++ STL (singly linked list)

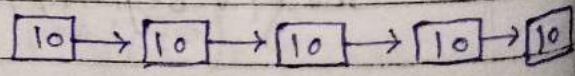
```
#include<iostream>
#include<forward_list>

int main()
{
    forward_list<int> l;
    l.assign({10,20,30,40});
    l.remove(10);

    for(auto it=l.begin(); it!=l.end(); it++)
        cout<<(*it);

    return 0;
}
```





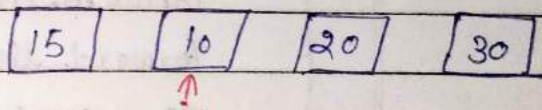
assigning one list to other

```
forward_list<int> l;
l.assign({10,20,30,10});

forward_list<int> l2;
l2.assign(l.begin(), l.end());

forwar_list<int> l3;
l3.assign(5,10);
```



* insert_after and emplace_after both are same functionality wise. but emplace_after Perform better for non-primitive type

```
#include <forward_list>

int main()
{
    forward_list<int> l = {15,20,30};
    auto it = l.insert_after(l.begin(), 10);
```




```
    it = l.insert_after(it, {2,3,5});
```

15, 10, 2, 3, 5, 20, 30
it

```
    it = l.emplace_after(it, 40);
```

15, 10, 2, 3, 5, 40, 20, 30
it

```
    it = l.erase_after(it);
```

15, 10, 2, 3, 5, 40, 30

```
}
```

* if we pass endPointer also then element from begin to end pointer will be deleted.

* erase_after() returns the pointer pointing the item next to deleted item.

More functions of forward_list

→ clear()

→ empty()

→ reverse()

→ merge()

```
forward_list<int> l1 = {10, 20, 30};
forward_list<int> l2 = {15, 25};
l1.merge(l2);
```

l1 : {10, 15, 20, 25, 30}

l2 : { }

→ sort().

Time complexity

insert_after() : O(1) for 1 item, O(m) for m item

erase_after() : " " " "

push_front() : O(1)

pop_front() : O(1)

reverse() : O(n)

sort() : O(nlgn)

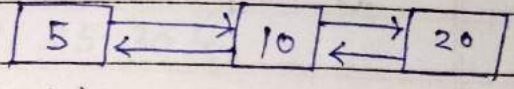
remove() : O(n)

assign() : O(1) for 1 item, O(m) for m item

list in C++ STL (Doubly Linked list)

Date / /
Page

```
#include<list>

int main()
{
    list<int> l;
    l.push_back(10);
    l.push_back(20);
    l.push_front(5);

    for(auto x:l)
        cout<<x<<" ";
    return 0;
}
```




```
int main()
{
    list<int> l = {10,2,5,20};
    l.pop_front();
    l.pop_back();

    for(list<int>::iterator it=l.begin(); it!=l.end; it++)
        cout<<(*it)<<" ";

    return 0;
}
```

insert() inserts the item before given position

```
int main()
{
    list<int> l = {10,20,30};
    auto it = l.begin();                    10,20,30
                                            ↑
    it++;                                   10,20,30
                                              ↑
    it = l.insert(it,15)                    10,15,20,30
                                              ↑
    it = l.insert(it,2,7);                  10,7,7,15,20,30
                  ↑
              frequency
    cout<<l.front();    // 10
    cout<<l.back();     // 30
}
```

```
int main()
{
    list<int> l = {10,20,30,40,20,40};
    auto it = l.begin();
    it = l.erase(it);                       {20,30,40,20,40}
                                            ↑
    l.remove(40);                           {20,30,20}
}
```

erase() takes an iterator and deletes item at the address and returns iterator pointing to next e

remove takes a value and remove all oc of given item.

* unique() removes the consecutive duplicate elements

Date / /
Page .

```
int main()
{
    list<int> l1 = {10,20,30};
    list<int> l2 = {5,15,25};
    l1.merge(l2);
}
```

l1 = {5,10,15,20,25,30}
l2 = { }

ex.

```
int main()
{
    list<int> l = {10,15,15,20,20,10}
    l.unique();
    l.sort();
    l.reverse();
}
```

{10,15,20,10}
{10,10,15,20}
{20,15,10,10}

Time complexity

front()
back()
size()
begin()
end()
erase()
push_front()
pop_front()
push_back()
pop_back()
→ O(1)

reverse()
unique()
remove()
→ O(n)

sort() → O(n log n)

Josephus Problem using list

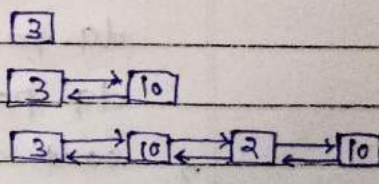
```
int getsurvivor(int n, int k)
{
    list<int> l;
    for(int i=0; i<n; i++)
        l.push_back(i);

    auto it = l.begin();

    while(l.size() > 1)
    {
        for(int cnt=1; cnt<k; cnt++)
        {
            it++;
            if(it == l.end())
                it = l.begin();
        }
        it = l.erase(it);
        if(it == l.end())
            it = l.begin();
    }
    return l.begin();
}
```

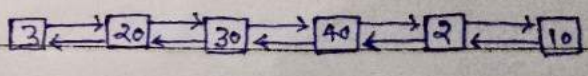
Time Comp → O(n·k)
Space → O(n)

Data structure with insert, replace, print functionality

Date / /
Page

```
#include<list>
#include<algorithm>
using namespace std;

list<int> l;
void insert() { l.push_back(x); }
void print() {
    for(auto x:l)
        cout<<x<<" ";
    cout<<endl;
}

void replace(int x, vector<int> &seq)
{
    auto it = find(l.begin(), l.end(), x);
    if(it==l.end())
        return;
    it = l.erase(it);
    l.insert(it, v.begin(), v.end());
}

int main()
{
    insert(3);
    insert(10);
    insert(2);
    insert(10);
    vector<int> v = {20, 30, 40};
    replace(10, seq);
    print();
    return 0;
}
```

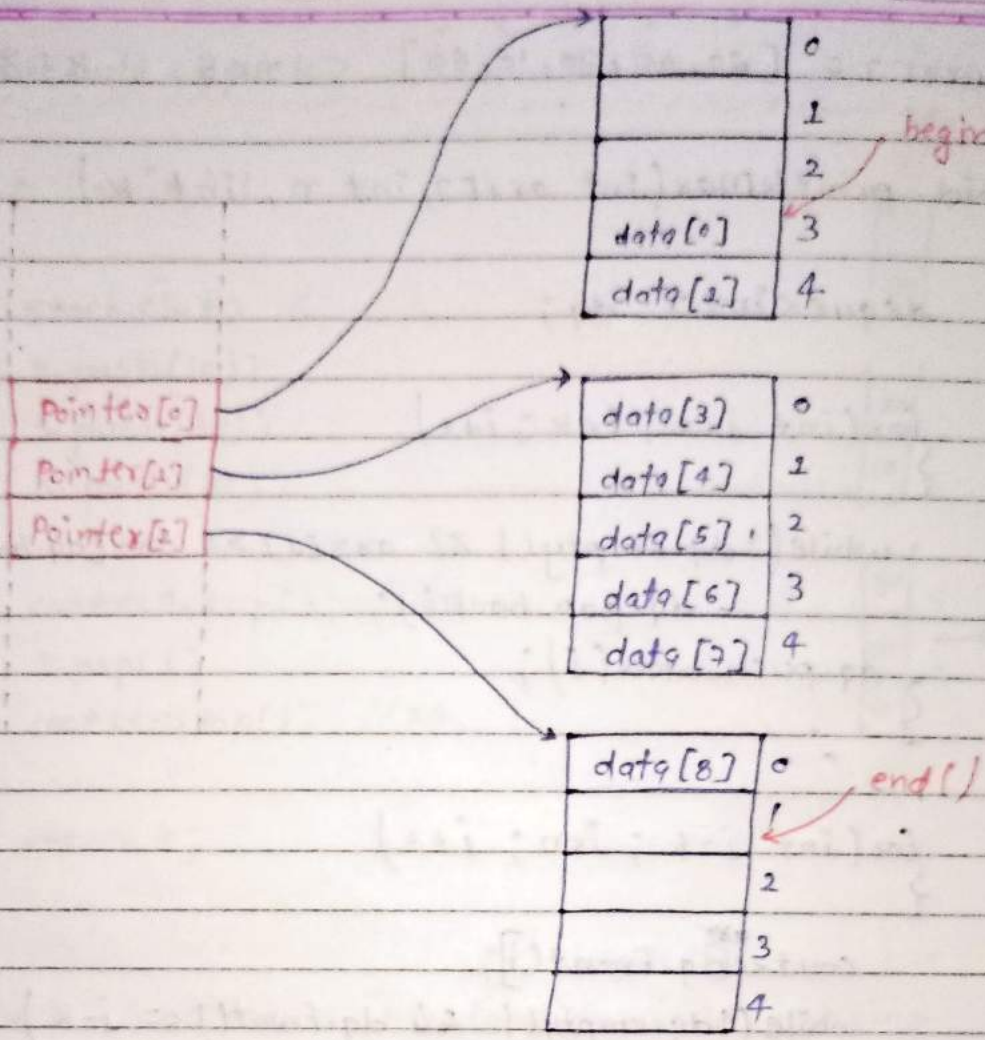




Dequeue in C++ STL

```
#include<deque>

int main()
{
    deque<int> dq = {10,20,30};
    dq.push_front(5);
    dq.push_back(50);
    for(auto it:dq)                    // 5 10 20 30 50
        cout<<it<<" ";
    cout<<dq.front()<<" "<<dq.back(); // 5 50
    return 0;
}

int main()
{
    deque<int> dq = {10,15,30,5,12};
    auto it = dq.begin();
    it++;

    dq.insert(it,20);      {10,20,15,30,5,12}

    dq.pop_front();        {20,15,30,5,12}
    dq.pop_back();         {20,15,30,5}
    cout<<dq.size();       // 4
}
```

Internal working of deque



insert() } O(n)
erase() }

push_back() }
push_front() } O(1)
pop_front() }
pop_back() }

* when these chunks become full new chunks are created and its base address is assigned to pointer array. when pointer array becomes full new pointer array with double size is created and old pointer is copied to new one.

AAA

Maximum in Subarray of Size K

arr[] = [20, 40, 30, 10, 60] n = 5 K = 3

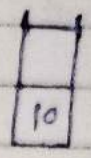
```
void printkMax(int arr[], int n, int K)
{
    deque<int> dq;

    for(int i=0; i<K; i++)
    {
      while(!dq.empty() && arr[i] >= arr[dq.back()])
            dq.pop_back();
      dq.push_back(i);
    }

    for(int i=K; i<n; i++)
    {
      cout<<arr[dq.front()];
      while(!dq.empty() && dq.front() <= i-K)
            dq.pop_front();

      while(!dq.empty() && arr[i] >= arr[dq.back()])
            dq.pop_back();

      dq.push_back();
    }
    cout<< arr[dq.front()] << " ";
}
```

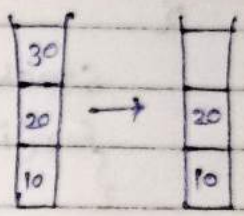
i=0 [0]
i=1 [1]
i=2 [1 2]
i=3 [1 2 3] i-k = 0
i=4 [4] i-k = 1

i = 0 1 2 3 4
output: 40 40 60

STACK in C++ STL

```
#include <stack>

int main()
{
    stack<int> s;
    s.push(10);
    s.push(20);
    s.push(30);

    cout<<s.top(); //30
    s.pop();
    cout<<s.top(); //20

    return 0;
}
```





* stack container is implemented using deque.
* containers which are implemented on top of other containers, are called Adapter Containe
* Adapter Containers works just like an interfa

Time complexity

push()	O(1)
pop()	
top()	
size()	
empty()	

Stack can be implemente using any other container which guarantee these operation in O(1).

∴ Stack can also be implemen using list or vector.

* Stack is useful for reversing forward list;

* Stack can used to solve complex problems easily.

*** Check for balanced paranthesis in String

```
bool isBalanced(string str)
{
    Stack<int> s;
    for(int i=0; i<str.size(); i++)
    {
        if(str[i]=='(' || str[i]=='{' || str[i]=='[')
            s.push(str[i]);
        else
        {
            if(s.empty==true)
                return false;
            else if(!matching(s.top(), str[i])
                return false;
            else
                s.pop();
        }
    }
    return s.empty()==true;
}

bool matching(char a, char b)
{
    return ((a=='(' && b==')') || (a=='{' && b=='}')
                || (a=='[' && b==']'));
}
```

Stock Span Problem

Date / /
Page

input: arr[] = { 15, 13, 12, 14, 16, 8, 6, 4, 10, 30 }

output: 1 1 1 3 5 1 1 1 4 10

Naive Solution

```
for(int i=0; i<n; i++)
{
    int cnt=1, j=i-1;
    while(arr[j]<=arr[i])
    {
        cnt++;
        j--;
    }
    cout<<cnt;
}
```

Optimized Solution

```
stack<int> s;
s.push(0);   cout<<1;
for(int i=1; i<n; i++)
{
    while(!s.empty() && arr[s.top()]<=arr[i])
        s.pop();

    int span = s.empty() ? i+1 : (i-s.top());
    cout<<span<<" ";
    s.push(i);
}
```

Previous Greater Element

input : {20, 30, 10, 5, 15}

output : -1 -1 30 10 30

```
void printGreater(int arr[], int n)
{
    stack<int> s;

    for(int i=0; i<n; i++)
    {
        while(!s.empty() && s.top() <= arr[i])
            s.pop();

        int prevMax = (s.empty) ? -1 : s.top();
        s.push(arr[i]);
        cout<<prevMax<<" ";
    }
}
```

Next Greater Element

Date / /
Page

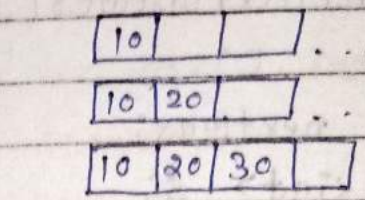
input: {5, 15, 10, 8, 6, 12, 9, 18}

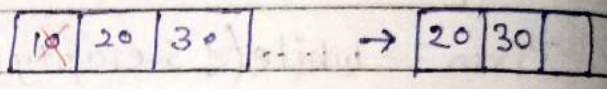
output: 15, 18, 12, 12, 12, 18, 18, -1

```
void printNextGreater(int arr[], int n)
{
    int nextMax;
    stack<int> s;

    for(int i=n-1; i>=0; i--)
    {
        while(!s.empty() && s.top()<=arr[i])
            s.pop();

        nextMax = (s.empty())? -1 : s.top();
        s.push(arr[i]);
        cout<<nextMax;
    }
}
```

Queue in C++ STL

```
#include <queue>

int main()
{
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);

    cout<<q.front()<<" "<<q.back()<<endl;

    q.pop();

    cout<<q.front()<<" "<<q.back()<<endl;

    return 0;
}
```





output:
10 30
20 30

```
int main()
{
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);
    while(q.empty() == false)
    {
        cout<<q.front();
        q.pop();
    }
}
```

* Queue can be implemented on any underlying container that provide these function in O(1). (list, dequeue (default))

empty()
size()
front()
back()
push_back()
pop_front()

Time Complexity

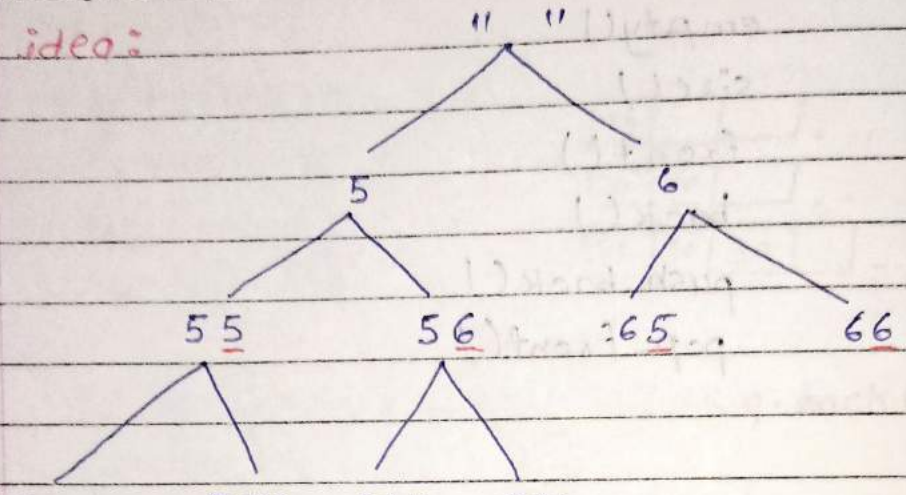
push()
pop()
front()
back()
empty()
size()

O(1)

Generate Number with given digit

Q. generate first n numbers containing the digit 5, 6.

idea:




```
void printFirstN(int n)
{
    queue<string> q;
    q.push("5");
    q.push("6");

    for(int cnt=0; cnt<n; cnt++)
    {
        string curr = q.front();
        cout<< curr << " ";
        q.pop();
        q.push(curr+"5");
        q.push(curr+"6");
    }
}
```

q | 5 | 6 | | | | |

q | 6 | 55 | 56 | | | cnt = 0

q | 55 | 56 | 65 | 66 | | cnt = 1

q | 56 | 65 | 66 | 555 | 556 | cnt = 2

Reverse first k item of Queue

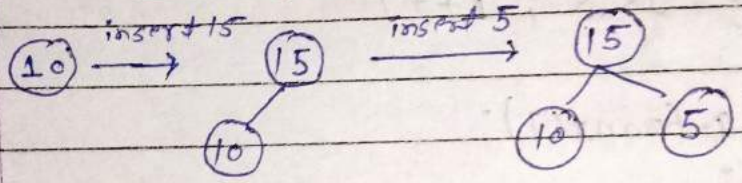
Date / /
Page

```
void reversek(queue<int> &q, int k)
{
    if(q.empty() || k > q.size || k <= 0)
        return;

    stack<int> s;
    for(int i=0; i<k; i++)
    {
        s.push(q.front());
        q.pop();
    }

    while(!s.empty())
    {
        q.push(s.top());
        s.pop();
    }

    for(int i=0; i<q.size()-k; i++)
    {
        q.push(q.front());
        q.pop();
    }
}
```

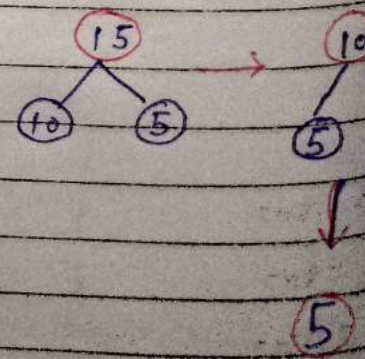
Priority_queue in C++

* priority_queue is always implemented using heap data structure.
* in c++ priority_que is implemented using max heap.

working of Max heap:



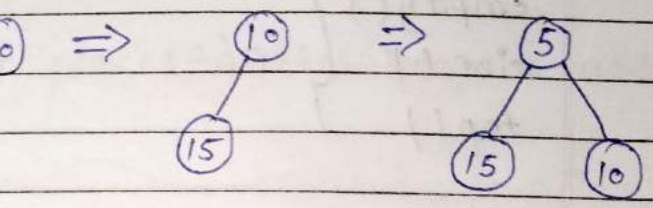

```
#include <queue>

int main()
{
    priority_queue<int> pq;
    pq.push(10);
    pq.push(15);
    pq.push(5);

    cout<<pq.top();    // 15

    while(!pq.empty())
    {
        cout<<pq.top()<<" ";
        pq.pop();
    }

    return 0;
}
```



output: 15 10 5

How to create a Priority_queue with Minimum at the top (Min_heap based)

```
int main()
{
    priority_queue<int, vector<int>, greater<int>> pq;

    pq.push(10);
    pq.push(15);
    pq.push(5);
```




```
    cout << pq.top(); // 5

    while(!pq.empty())
    {
        cout << pq.top() << " ";
        pq.pop();
    }
    return 0;
}
```

creating pq from array or vector

```
int main()
{
    int arr[] = {10, 5, 15};
    vector<int> v{10, 5, 15};

    priority_queue<int> pq(arr, arr+3);
    priority_queue<int> pq2(v.begin(), v.end());
}
```

* this way is preferable than pushing each element one by one.

* another way of keeping smaller elem at top is first do arr[i] = arr[i] and then create pq. while printing cout << pq.top()

* for pairs by default comparison is done for 1st value:

Time complexity

push() } pop() } logn

empty() size() top() } O(1)

Custom Comparator

```
struct person
{
    int age;
    float height;
    person(int a, float h)
    {
        age = a;
        height = h;
    }
};

struct mycmp
{
    bool operator()(Person const &P1, Person const &P2)
    {
        return P1.height < P2.height;
    }
};

int main()
{
    priority_queue<Person, vector<Person>, mycmp>
}
```

Application of priority queue :-

- Dikstra Algorithm
- Prim Algorithm
- Huffman Algorithm
- Heap Sort
- - - - any other places where heap is used

K Largest element

Arr = [5, 15, 10, 20, 8, 25, 18] K = 3
0 1 2 3 4 5 6

(1) Build a Min Heap of first K item. O(K)

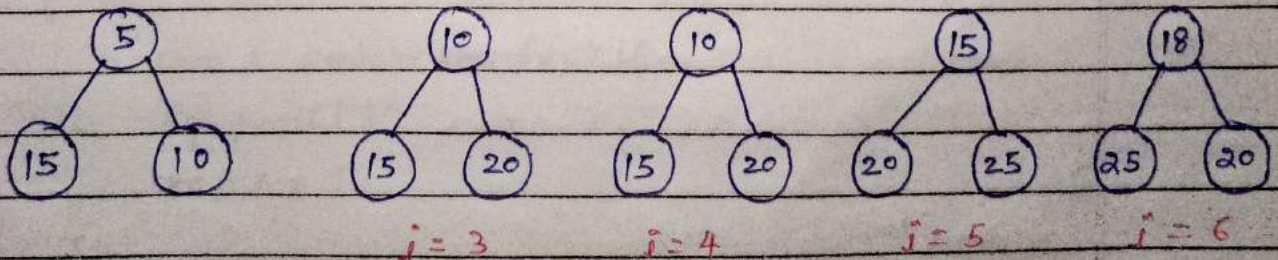
(2) Traverse from $(K+1)^{th}$ element

(a) Compare current element with top of heap. if smaller than top, ignore it. O((n-k)*logK)

(b) else remove the top element and insert the current element in the Min Heap.

(3) Print contents of Min Heap. O(K)

1st step 2nd step



overall Time Complexity = O(K + (n-K) * logK)

K largest Element

```
void KLargest(int arr[], int n, int K)
{
    priority_queue<int, vector<int>, greater<int>>
        pq(arr, arr+K);

    for(int i=K; i<n; i++)
    {
        if(arr[i] > pq.top())
        {
            pq.pop();
            pq.push(arr[i]);
        }
    }

    while(!pq.empty())
    {
        cout<<pq.top()<<" ";
        pq.pop();
    }
}
```

Purchasing Maximum Items

Date / /
Page

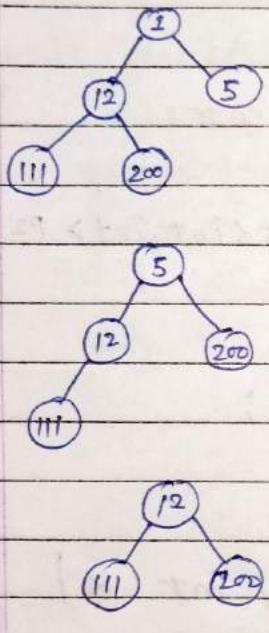
Input: arr[] = {1, 12, 5, 111, 200}

sum = 10

Output: 2

Method - 1 (Sorting)

Method - 2 (Heap)



sum = 10 res = 0

since top is smaller than sum pop it

sum = 9 res = 1

since top is smaller than sum, Pop it

sum = 4 res = 2

top is not smaller than sum, return res

```
int purchaseMax(int arr[], int n, int sum)
{
    priority_queue<int, vector<int>, greater<int>> pq(arr, arr+n);
    int res=0;
    while (sum>=0 && !pq.empty() && pq.top()<=sum)
    {
        sum = sum - pq.top();
        pq.pop();
        res++;
    }
    return res;
}
```

Find K most frequent number

* if multiple items has same frequency, give preference to smaller items.

input: arr[] = {10, 5, 20, 5, 10, 10, 30} K=

output: 10 5

input: arr[] = {20, 40, 30, 20, 30, 40, 60, 60} K

output: 20 30 40

Method-1: using unordered_map and vector

```
bool compare(Pair<int,int> P1, Pair<int,int> P2)
{
    if(P1.second == P2.second)
        return P1.first < P2.first;
    return P1.second > P2.second;
}
void printKfrequent(int arr[], int n, int K)
{
    unordered_map<int,int> m;
    for(int i=0; i<n; i++)
        m[arr[i]]++;
    vector<Pair<int,int>> v(m.begin(), m.end());
    sort(v.begin(), v.end(), compare);

    for(int i=0; i<K; i++)
        cout<<v[i].first<<" ";
}
```

m = {(10,3), (30,2), (20,2)}

v = {(10,3), (20,2), (30,2)}

O(n log n)

output: 10,20

Method-2 (using unordered_map and Priority_queue)

```
struct Mycmp
{
  bool operator()(pair<int,int> P1, Pair<int,int> P2)
  {
    if(P1.second == P2.second)
        return P1.first > P2.first;
    return P1.second < P2.second;
  }
};

void printKMaxFreq(int arr[], int n)
{
    unordered_map<int,int> m;
    for(int i=0; i<n; i++)
        m[arr[i]]++;                                   } O(n)
    priority_queue<Pair<int,int>, vector<Pair<int,int>>,
                   mycmp> pq(m.begin(), m.end());
    for(int i=0; i<K; i++)
    {                                                  O(K log n)
        cout<< pq.top().first;
        pq.pop();
    }
}
```

O(n)

*** in sort function we expect a function template as Comparator
But in priority_queue constructor class template is expected.

K most frequent in Linear time

* if multiple element have same frequency give preferance to first occuring element

```
void printkfrequent(int arr[], int n)
{
    unordered_map<int, int> m;
    for(int x : arr)
        m[x]++;

    vector<int> freq[n+1];

    for(auto x : m)
        freq[x.second].push_back(x.first);

    int count = 0;
    for(int i = n; i >= 0; i--)
    {
        for(int x : freq[i])
        {
            cout << x << " ";
            count++;
            if(count == k)
                return;
        }
    }
}
```

Ex. arr[] = {10, 5, 20, 5, 10, 10, 30} n = 7

m = {{10,3}, {5,2}, {2,1}, {30,1}}

Freq[] = {{}, {20}, {30}, {5}, {10}, {}, {}, {}}

```
void printKfrequent(int arr[], int n)
{
    unordered_map<int,int> m;
    for(int x: arr)
        m[x]++;

    vector<int> freq[n+1];

    for(int i=0; i<n; i++)
    {
        int f = m[arr[i]];
        if(f != -1)
        {
            freq[f].push_back(arr[i]);
            m[arr[i]] = -1;
        }
    }

    . . . . .

    . . . . .

}
```

ex. arr[] = {10, 40, 40, 10} K = 1

m = {{40, 2}, {10, 2}}

freq = {{}, {}, {10, 40}, {}, {}}

output: 10

Set in C++ STL

* By default stores in increasing order.

```
#include<set>
int main()
{
    set<int> s;
    s.insert(10);
    s.insert(5);
    s.insert(20);

    for(int x: s)
        cout<<x<<" ";
    return 0;
}
```

5 10 20

* for decreasing order

```
int main()
{
    set<int, greater<int>> s;
    s.insert(10);
    s.insert(5);
    s.insert(20);

    for(int x:s)
        cout<<x<<" ";
    return 0;
}
```

20 10 5

* duplicate values are ignored.

Finding an element

```
int main()
{
    set<int> s;
    s.insert(20);
    s.insert(5);
    s.insert(10);

    auto it = s.find(10);

    if(it == s.end())
        cout<<"Not Found";
    else
        cout<<"Found";
}
```

find(): return iterator to element if it is available else return iterator to end()

clear(): removes all element of set

count(): returns 1 if item is present else 0

erase(): to remove a particular item either by providing value or by providing iterator to the element.
we can erase set by providing range also

ex.
```
auto it = s.find(7);
s.erase(it, s.end());
```

Date
Page

upper_bound() and lower_bound()

```
set<int> s;
s.insert(10);
s.insert(5);
s.insert(20);
```

```
it = s.upper_bound(5);
```
5 10 20 (↑ 10)

```
it = s.upper_bound(6);
```
5 10 20 (↑ 10)

```
it = s.upper_bound(25);
```
5 10 20 (↑ past end)

```
it = s.lower_bound(10);
```
5 10 20 (↑ 10)

```
it = s.lower_bound(4);
```
5 10 20 (↓ 5)

* set is built on top of self balancing binary search tree (Red Black Tree).

Time Complexity

begin(), end()
rbegin(), rend()
cbegin(), cend(), crbegin(), crend()
size(), empty()
} O(1)

insert()
find()
count(), lower_bound()
upper_bound(), erase()
} O(log n)

erase(it) —— Amortized O(1)

ex.

```
struct Test
{
    int x;
    bool operator< (const Test &t)
    {
        return (this->x < t.x);
    }
};

int main()
{
    set<Test> s;
    s.insert({5});
    s.insert({20});
    s.insert({10});

    for(Test t: s)
        cout<<t.x<<" ";
    return 0;
}
```

output: 5 10 20

Application of Set:

: sorted stream of data.

: doubly Ended priority Queue

* set iterators are bidirectional.

Design a DS with floor and ceil

operation:

bool search(x)
Void insert(x)
void delete(x)
int getfloor(x) : largest val smaller or equal
int getceiling(x): smaller val greater or equal

```
set<int> s;
void insert(int x) { return s.insert(x); }

void delete(int x) { s.erase(x) }

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

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

Ceiling of every element on Right

Naive method

```
void printCeiling( int arr[], int n)
{
    for(int i=0, i<n ; i++)
    {
        int diff = INT_MAX;
        for(int j=i+1; j<n; j++)
            if(arr[j] >= arr[i])
                diff = min(diff, arr[j]-arr[i]);

        if(diff == INT_MAX)
            cout<< -1;
        else
            cout<< arr[i] + diff << " ";
    }
}
```

arr[] = { 100 (0), 50 (1), 30 (2), 60 (3), 55 (4), 32 (5) }

i = 0	diff = INT_MAX	-1
i = 1	diff = 5	55
i = 2	diff = 2	32
i = 3	diff = INT_MAX	-1
i = 4	diff = INT_MAX	-1
i = 5	diff = INT_MAX	-1

Date / /
Page

Efficient method

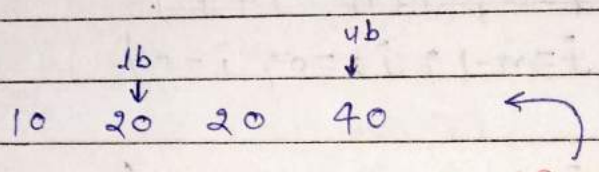
```
void printCeiling(int arr[], int n)
{
    set<int> s;
    int res[n];
    for(int i=0; i<n; i++)
    for(int i=n-1; i>=0; i--)
    {
        auto it = s.lower_bound(arr[i]);
        if(it == s.end())
            res[i] = -1;
        else
            res[i] = *it;
        s.insert(arr[i]);
    }
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

O(nlogn) → Time
O(n) → space

Multiset

* only difference b/w set and multiset is set allow single instance of key whereas multiset allows multiple instance of a key.

* erase() will erase all occurances of provid item.



set

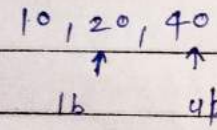
s.lower_bound(20) //20

s.upper_bound(20) //40

multiset

m.lower_bound(20) // 20

m.upper_bound(20) // 40




```
int main()
{
    multiset<int> ms;
    ms.insert(20);
    ms.insert(10);
    ms.insert(20);
    ms.insert(40);

    auto it = ms.equal_range(20);
    cout<<(*it).first<<" "<<(*it).second;
}
```

auto → Pair<multiset<int>::iterator, multiset<int>::iterator>

Map in C++ STL

Date / /
Page

* Similarity b/w set and map is, both use Red-Black Tree internally.

* difference is set is used to store only key whereas map is used to store key-value pair.

```
#include<map>
int main()
{
    map<int,int> m;

    m.insert({10,200});
    m[5] = 100;
    m.insert({3,300});

    m.insert({3,800});   // this key-value will be ignored

    for(auto &x : m)
        cout<<x.first<<" "<<x.second<<endl;

    return 0;
}
```

Functions

operator []
at()
insert()
size()
begin()
end()
empty()
clear()

output:

```
10 200
5 100
3 300
```

`cout<<m[10];` if m[10] is not available, it will insert {10,0} in map

* m.at(10) is same as m[10] but m.at(20) will throw exception (out of range) if key is not present

* minimum is always at m.begin() and maximum at m.at(m.size-1)

```
int main()
{
    map<int, string> m;
    m.insert({5, "gfg"});

    m.insert({2, "ide"});

    m.insert({1, "Practice"});

    for(auto it = m.find(2); it != m.end(); it++)
        cout << (*it).first << " " << (*it).second << "\n";

    if (m.find(5) == m.end())
        cout << "Not Found";
    else
        cout << "Found";
}
```

O/P:
2 ide
5 gfg

* count() can also be used to check whether a key is found or not. it returns 0 or 1

* lower_bound() finds a value if it is pres or greater value. if provided key is greater than greatest then it returns e

* upper_bound() returns next key.

* erase() removes the element. it can take key, iterator or range of iterator.

* Time complexity is same as set. [] at() } O(log

Applications:
- sorted stream of data with (key, value) pair.
- doubly ended priority queue of items with (key, val) pair

Count greater Elements

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

    int cum_freq = 0;
    for(auto it = m.rbegin(); it != m.rend(); it++)
    {
        int freq = it->second;
        it->second = cum_freq;
        cum_freq += freq;
    }

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

Alternate method:

(i) arr[] = {10, 2, 8, 5, 8}

(ii) Copy arr to tem[] and sort temp[]
temp[] = {2, 5, 8, 8, 10}

(iii) traverse through arr and use binary search in temp[] to find greater count.

Multi Map

* allows duplicate keys.
* [] is not applicable.

```
#include <map>
int main()
{
    multimap<int, int> mm;
    mm.insert({10,20});
    mm.insert({5,50});
    mm.insert({10,25});

    cout<<mm.count(10); // 2
    mm.erase(10);
    cout<<mm.count(10); // 0

    return 0;
}
```

let values are

key	val
10	5
20	3
20	2
50	1

mm.lower_bound(20)

mm.upper_bound(20);

* equal_range(20) will return iterator of lower_bound and upper_bound.

* other most of the functions are same.

Design a DS for item and price (Duplicates allowed)

```
multimap<int, string> m;

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

void find(int P)
{
    auto ip = m.equal_range(P);
    for(auto it = ip.first; it != ip.second; it++)
        cout<<it->second<<" "<<it->first;
}

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

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

void printsmaller(int P)
{
    auto it1 = m.lower_bound(P);
    for(auto it2 = m.begin(); it2 != it1; it2++)
        cout<<it2->second<<" "<<it2->first<<endl;
}
```

Unordered_set in C++ STL

* unordered_set is based on Hashing whereas set is based on Red-Black Tree.

* in unordered_set there is no order among the elements.

```
#include<unordered_set>
int main()
{
    unordered_set<int> s;
    s.insert(10);
    s.insert(5);
    s.insert(15);
    s.insert(20);
    for(int x:s)
        cout<<x<<" ";
    return 0;
}
```

* we can't predict the output of this program as it will differ compiler to compiler.

* duplicates are not allowed.

Imoportant functions :

insert()
begin()
end()
size()
clear()
find()
count() - returns 0 or 1
erase() → item/iterator or range of iterator

Date / /
Page

Time complexity

begin(), end(), rbegin(), rend() } O(1)

insert(), erase(), find(), count() } O(1) on average

size(), empty() } O(1)

Applications :-

anywhere when following operation required on key quickly.

- Search
- insert
- delete

Print distict Element

```
void printdistinct(int arr[], int n)
{
    unordered_set<int> s;
    for(int i=0; i<n; i++)
    {
        if(s.find(arr[i]) == s.end())
        {
            cout<<arr[i]<<" ";
            s.insert(arr[i]);
        }
    }
}
```

input: { 5, 6, 5, 5, 8 }
output: 5, 6, 8

Print repeating element

```
void PrintRepeating(int arr[], int n)
{
    unordered_set<int> s;
    for(int i=0; i<n; i++)
    {
        if(s.find(arr[i]) == s.end())
            s.insert(arr[i]);
        else
            cout<<arr[i];
    }
}
```

input: { 5, 3, 7, 5, 8, 3 }
output: 5 3

Pair Sum

Date / /
Page

input: arr[] = { 3, 2, 8, 15, -8 } sum = 17

output : true

```
bool isPair(int arr[], int n)
{
    unordered_set<int> s;
    for(int i=0; i<n ; i++)
    {
        if (s.find(-arr[i]+sum) == s.end())
                return true;
        else
                s.insert(arr[i]);
    }
    return false;
}
```

Intersection of two array (order based on first array)

let b[] is first array

```
void PrintIntersection(int arr[], int m, int b[], int n)
{
    unordered_set<int> s;
    for(int i=0; i<m ; i++)
        s.insert(arr[i]);                    O(m+n)

    for(int i=0; i<n ; i++)
        if( s.find(b[i]) == s.end())
            cout<< b[i] << " ";
}
```

b[] = { 10, 15, 20, 25, 30, 50 }

arr[] = { 30, 5, 15, 80 }

S = { 30, 5, 15, 80 }

output : 15, 30

Maximum distinct element in a Subset

input: {1, 1, 2, 1, 3, 1} K = 2

{1, 1, 1}, {1, 2, 3}

output: 3

input: {1, 1, 2, 1, 3, 1} K = 3

{1, 1}, {2, 3}, {1, 1}

OR {1, 2}, {1, 3}, {1, 1}

output: 2

Algorithm:

(i) count distinct element. Let count be d

(ii) if d >= n/k

return n/k

(3) else

return d

```
int maxdistinct(int arr[], int n, int k)
{
    unordered_set<int> s;
    for(int i=0; i<n; i++)
        s.insert(arr[i]);
    int d = s.size();

    if( d ≥ n/k)
        return n/k;
    else
        return d;
}
```

Longest Consecutive Subsequence

Date / /
Page

input : arr[] = {1, 9, 3, 4, 2, 10, 13}
output : 4

Method - 1 :

sort array → {1, 2, 3, 4, 9, 10, 13}
(1, 2, 3, 4 → 4; 9, 10 → 2)

```
int findLongest(int arr[], int n)
{
    sort(arr, arr+n);
    int res = 1, cur = 1;

    for(int i=1; i<n; i++)
    {
        if(arr[i] == arr[i-1] + 1)
            cur++;
        else
        {
            res = max(res, cur);
            cur = 1;
        }
    }
    res = max(res, cur);
    return res;
}
```

O(n log n)

optimal Solution

Algorithm:

(i) initialize res = 0
(ii) create a hash table h
(iii) insert all element into h
(iv) for(i=0; i<n ; i++)
{
    if arr[i]-1 is present in ~~set~~ h
        cur = 1
        while(h contains arr[i]+cur)
            cur ++
        res = max(cur, res)
(v) return res

```
int longestsequence(int arr[], int n)
{
    int res = 0;
    unordered_set<int> h;
    for(int i=0; i<n; i++)
        h.insert(arr[i]);
    for(int i=0; i<n ; i++)
    {
        if( h.find(arr[i]-1) != h.end() )
        {
            cur = 1;
            while( h.find(arr[i]+curr) != h.end()
                cur++;
            res = max(res, cur);
        }
    }
    return res;
}
```

it looks like n^2 but it is $O(n)$

Subarray with 0 Sum

Date / /
Page

input: {1, 4, 13, -3, -10, 5}
o/p : Yes

input: {1, 4, -3, 1, 2}
output: Yes

input: {5, 6, 0, 8} output : Yes

idea: take prefix sum → {1, 4, 13, -3, -10, 5}

↓

{1, 5, 18, 15, 5, 10}

if elements are repeating means there is a subarray having sum is 0

cumulative sum is again being 5 means sum of these elements are 0

But we nee to handle this case also → {3, -1, -2, 5}

↓

{3, 2, 0, 5}

```
bool issumzero(int arr[], int n)
{
    unordered_set<int> s;
    int pre_sum = 0;
    for(int i=0; i<n; i++)
    {
        pre_sum = pre_sum + arr[i];
        if(s.find(Presum) != s.end())
            return true;
        if(Pre_sum == 0)
            return true;
        s.insert(Pre_sum);
    }
    return false;
}
```

unordered_map in C++ STL

* map is based on Red-Black Trees but unordered_map is based on hashing.
* there is no order
* stores key-value pair

```
#include<unordered_map>
int main()
{
    unordered_map<string,int> um;
    um.insert({"STL",15});
    um["gfg"] = 20;
    um["ide"] = 30;

    for(auto x:m)
        cout<<x.first<<" "<<x.second<<endl;
    return 0;
}
```

* rest of the functions are almost same and rarely used.

Time Complexity

O(1):
- begin()
- end()
- size()
- empty()

O(1) on average:
- count()
- find()
- erase()
- insert()
- []
- at

* [] (index operator) → try to access a value; if not present it assign value 0 at that

Design a DS for storing and accesing Balance and id

```
set(1,100)          get(1)  // 100
set(1,800)          get(1)  // 800
set(2,500)
set(5,100)          get(5)  // 100
```

O(1) Solution

```
unordered_map<int,int> m;

int get(int id)
{
    return m[id];
}

void set(int id, int bal)
{
    m[id] = bal;
}
```

Frequency of array Element O(n)

```
int printfreq(int arr[], int n)
{
    unordered_map<int,int> um;
    for(int i=0; i<n; i++)
        um[arr[i]]++;

    for(auto x: um)
        cout<<x.first<<" "<<x.second<<endl;
}
```

Winner of an election

* if two candidate have same number of votes return lexigraphically less smaller candidate

```
string findWinner(string arr[], int n)
{
    unordered_map<string, int> um;
    for(int i=0; i<n; i++)
        um[arr[i]]++;

    int max_freq = 0;
    string winner;

    for(auto x : m)
    {
        if(x.second > max_freq)
        {
            max_freq = x.second;
            winner = x.first;
        }
        else if(x.second == max_freq &&
                x.first < winner)
            winner = x.first;
    }
    return winner;
}
```

Count distinct element in every window

input: arr[] = {10, 10, 5, 3, 20, 5} K = 4

3 4 3

Naive

```
void printdistinct(int arr[], int n, int k)
{
    for(int i=0; i<=n-k; i++)
    {
        int count=0;
        for(int j=0; j<k; j++)
        {
            bool flag=false;
            for(int p=0; p<j; p++)
            {
                if(arr[i+j]==arr[i+p])
                {
                    flag=true;
                    break;
                }
            }
            if(flag==false)
                count++;
        }
        cout<<count<<" ";
    }
}
```

* no of windows = n-k+1 where k = size of window

Efficient solution O(n)

Algorithm:

1. create a frequency map of first K items
2. print size of frequency map.
3. for(i = K ; i<n ; i++)
   a. decrease frequency of arr[i-K]
   b. if the frequency of arr[i-K] becomes remove it from map.
   c. if arr[i] does not exist in the m insert it else increase the frequen
   d. print size of the map.

ex. {10, 20, 10, 10, 30, 40} K = 4

i = 0 1 2 3 4 5

(i)

10	3
20	1

(ii) 2

(iii)

10	2
20	1
30	1

i = 4

3

10	2
30	1
40	1

i = 5

3

```
void printdistinct(int arr[], int n, int k)
{
    unordered_map<int, int> freq;
    for(int i=0; i<k; i++)
        freq[arr[i]]++;

    cout<< freq.size();

    for(int i=k; i<n; i++)
    {
        freq[arr[i-k]]--;
        if(freq[i-k]==0)
            freq.erase(arr[i-k]);
        freq[arr[i]]++;

        cout<< freq.size()<<" ";
    }
}
```

ALGORITHM

Non-Mutating Algorithms

find() O(n)

```
#include<algorithm>
int main()
{
    vector<int> = { 5,10,7,10,20 };
    auto it = find(v.begin(),v.end(), 10);

    if( it == v.end() )
            cout<< "Not Found"
    else
            cout<< "Found at" << (it - v.begin());
    return 0;
}
```

```
int main()
{
    int arr[] = {5,10,12,8,7,9,4};
    int *ptr = find(arr, arr+6, 12);

    if( ptr == (arr+6) )
            cout<< "Not Found";
    else
            cout<< "Found at"<< (ptr - arr);
}
```

* it is highly recommended to use contain specific find() because it is optimized for particular containers.

lower_bound ()

Date / /
Page

* Returns an iterator having address of element greater than equal to given value in a given sorted range.

* it uses binary search internally.

```
#include<algorithm>

int main()
{
    int arr[] = {10,20,20,30,40,50};
    auto it = lower_bound(arr, arr+6, 20);

    if((it==(arr+6)) || ((*it) != 20))
        cout<< "Not Found";
    else
        cout<< "found";
    return 0;
}
```

* it takes 3 args begin, end and item

arr[] = { 2, 8, 10, 12, 15, 18 }
↑ begin ↑ end

Time complexity : O(logn) for Random access containers.

n → no of elements in given Range.

upper_bound()

* Return iterator to the first greater element in a sorted range.

```
#include<algorithm>
int main()
{
    vector<int> v = { 10,20,20,20,30,40 };
    auto it = upper_bound(v.begin(), v.end(), 20);

    if( it != v.begin() && (*(it-1) == 20) )
            cout<< "found" << (it - v.begin());
    else
            cout<<"Not found";
}
```

```
arr[] = { 10,20,20,20,30,40 };
```

upper_bound(v.begin(), v.end(), 20);

Time complexity: O(log n) for random access container.

* lower_bound, upperbound() work for sorted data only.

* lower_bound, upper_bound both uses it = advance(iterator, n), n = num of elem to move

is_permutation()

Date / /
Page

* used to check whether two array, vector ---- are equal or not.

* it can also be used to check whether 2 strings are anagram or not but it is not recommeded as it is time costly. for this purpose.

```
#include<algorithm>
int main()
{
    vector<int> v1 = {10,20,30};
    vector<int> v2 = {20,10,30};
```

(i)

```
    if(ispermutation(v1.begin(), v1.end(), v2.begin())
        cout<<"Yes";
    else
        cout<<"No";
```

(ii)

```
    vector<int> v3 = {10,20,10,30};
    vector<int> v4 = {10,30,20,30};

    if(is_permutation(v3.begin(), v3.end(), v4.begin()))
        cout<<"Yes";
    else
        cout<<"No";
}
```

output: Yes (i)
No (ii)

max_element() and min_element()

```
vector<int> v = {10,5,30,40,70,98,50};
auto it1 = max_element(v.begin(), v.end());
auto it2 = min_element(v.begin(), v.end());

cout<<(*it1)<<" "<<(*it2); // 98 5

int arr[] = {5,6,20,50,90,7}
cout<< *min_element(arr, arr+6); // 5
cout<< *max_element(arr, arr+6); // 90
```

ex.

```
struct point
{
    int x,y;
    point(int i, int j) { x=i; y=j; }
};

bool mycmp(point P1, Point P2)
{
    return P1.x < P2.x;
}

int main()
{ vector<point> v = {{5,4},{2,300},{10,50}};

    auto it = max_element(v.begin(),v.end(), mycmp);

    cout<< it->x<<" "<<((*it).y)<<endl;
}
```

$\theta(n)$

count()

Date / /
Page

```
#include<algorithm>

int main()
{
    vector<int> v={5,10,5,8,7,5,8};
    cout<<count(v.begin(),v.end(),5);      //3

    int arr[]={5,8,5,7,9};
    cout<<count(arr,arr+5,5);              //2

    list<int> l={10,5,10,7,10,20};
    cout<<count(l.begin(),l.end(),10);     //3

    string s="Leetcode";
    cout<<count(s.begin(),s.end(),'e');    //3
```

binary_search()

we can pass custom comparator

```
vector<int> v={5,8,10,12,15};
binary_search(v.begin(),v.end(),10);
```

internal working

```
it = lower_bound(begin, end, x)
if(it != end && !(x<*it))
        return true;
else
        return false;
```

fill()

```
{
vector<int> v = { 10, 20, 30, 50, 80 };
fill(v.begin(), v.end(), 5);
for(x:v)
    cout<<x<<" ";
```

5 5 5 5 5

```
fill(v.begin()+1, v.end()-1, 8);
}
```

5 8 8 8 5

* Similarly we can fill array, list, string etc. but list does not allow ptr+ or ptr-1

rotate()

```
{
vector<int> v = { 5, 8, 12, 18, 22, 25 }
rotate(v.begin(), v.begin()+2, v.end());
```

{ 12, 18, 22, 25, 5, 8 }

* it can be used for string, deque

accumulate()

```
{
    vector<int> v = {10,20,30};
    int init_res = 0;
    cout << accumulate(v.begin(), v.end(), init_res); //60

    init_res = 100;
    cout<< accumulate(v.begin(), v.end(), init_res, minus<int>());

    init_res = 1; int arr[] = {5,4,10};
    cout<< accumulate(arr, arr+3, init_res, myfun); //200
}

int myfun(int x, int y)
{
    return x*y;
}
```

rand()

* the number generated is not truly random bcoz it is generated by some fixed number of steps.

range => 0 to RAND_MAX
where RAND_MAX is a variable
and RAND_MAX ≥ 32767

* it uses linear congrential generator

$$x_{n+1} = (ax_n + b) \% m$$

```
#include<cstdlib>
#include<ctime>
int main(){ srand(time(Null)); //to generate unique num each time
            for(int i=0; i<5; i++)
            {
                cout<<rand()%100<<" ";
            }
}
```

this will return current time

to get a num b/w 0 to 99

all elems are distinct

Median of a Row-wise Sorted Matrix

Naive Solution

(i) put all element in a array. O(r*c)
(ii) Sort the array. O(rc log rc)
(iii) return middle element of sorted array. O(1)

Optimized: O(r * log(max-min) * log c)

```
int MatMed(int arr[], int r, int c)
{ int min = arr[0][0], max = arr[0][c-1];
  for(int i=1; i<r; i++)
  { if(arr[i][0] < min) { min = arr[i][0]; }
    if(arr[i][c-1] > max) { max = arr[i][c-1]; }
  }
  int medPos = (r*c + 1) / 2;
  while (min < max)
  {
    int mid = (min+max) / 2, midPos = 0;
    for(int i=0; i<r; i++)
    { int pos = Arrays.binarySearch(arr[i], mid) + 1;
      midPos = midPos + Math.abs(pos);
    }
    if(mid < medPos)
        mid = mid + 1;
    else
        max = mid;
  }
  return min;
}
```

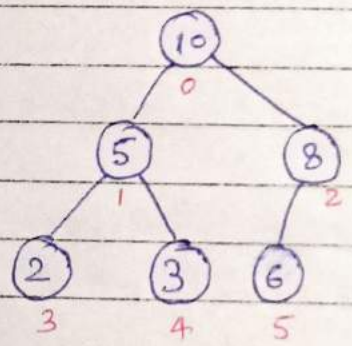
Mutating STL Algorithms

sort();

sort(arr, arr+n)
sort(arr, arr+n, greater<int>()); //decreasing ord

make_heap() - O(n)

* it takes container as input.
* By default max heap is created.



left(i) = 2i+1
left(0) = 2x0+1 = 1

right(i) = 2i+2

Parent(i) = $\left(\frac{(i-1)}{2}\right)$

```
int main()
{
    vector<int> v = {15,6,7,12,30};
    make_heap(v.begin(), v.end());
    cout<< v.front();
}
```

output = 30

to create min heap

make_heap(v.begin(), v.end(), greater<int>());

* make heap converts the vector into heap

* pop_heap( ) moves the min/max item at the which is treated as deleted item.
* push_heap( ) again takes the items at the end into consideration.

ex.
```
int main()
{
    vector<int> v = { 15, 6, 7, 12, 30 };
    make_heap(v.begin(), v.end(), greater<int>());
    cout<<v.front();    // 6

    pop_heap(v.begin, v.end(), greater<int>());
    cout<<v.front();    // 7

    v[4] = 2;
    push_heap(v.begin(), v.end(); greater<int>());
    cout<<v.front();    // 2
}
```

Sort_heap()

```
int main()
{
    vector<int>  v = { 15, 6, 7, 12, 30 };
    make_heap(v.begin(), v.end(), greater<int>());
    for( auto x : v )
        cout<<x<<" ";                    6 12 7 15 30

    Sort_heap(v.begin(), v.end(), greater<int>());
    for(auto x : v )
        cout<<x<<" ";                    30 15 12 7 6
}
```

Merge 2 sorted array in-Place.

input: a[] = { 3, 20, 40 }
b[] = { 2, 10, 12 }

output: a[] = { 2, 3, 10 }
b[] = { 12, 20, 40 }

```
void merge(int a[], int b[], int m, int n)
{
    for(int i=0; i<m ; i++)
    {
        if( a[i] > b[0] )
        {
            pop_heap(b, b+n, greater<int>());
            swap(a[i], b[n-1]);
            push_heap(b, b+n, greater<int>());
        }
    }
    sort(b, b+n);
}
```

Time → O((m+n) log n)

Date / /
Page

merge()

```
int main()
{
    vector<int> v1 = {10,20,40};
    vector<int> v2 = {5,15,30};
    vecto<int> v3(v1.size()+v2.size());

    merge(v1.begin(),v1.end(),v2.begin(),v2.end()
                                    v3.begin());

    for(int x:v3)
        cout<<x<<" ";          5 10 15 20 30 40
}
```

* before calling merge() make sure both the arrays or vectors or other container are sorted.

* list has it's own merge() function.

next_permutation() O(n)

* it ~~returns~~ makes the lexicographically next permutation of a sequence.

ex. input: {1, 2, 3, 4, 5}
output: {1, 2, 3, 5, 4}

input: {5, 4, 3, 2, 1}
output: {5, 4, 3, 2, 1}

```
int main()
{
    vector<int> v = {5,4, {1,2,5,4,3};
    next_permutation(v.begin(), v.end());
    for(int x:v)
        cout<<x<<" ";
}
```

output: 1 3 2 4 5

Internal Working Algo

1. Traverse from right, find the first element that is not decreasing order. let it be x.
2. find the smallest greater element on right of x. let it be y.
3. swap x and y.
4. Reverse the subsequence after new sequence of y. ex → 1 2 5 4 3 → 1 3 5 4 2

x = 2 y = 3 → 1 3 2 4 5

reverse()

```
int main()
{
    vector<int> v = { 1,2,3,4,5,6,7 };
    reverse(v.begin(), v.end());
    for(int x:v)
        cout<<x<<" ";          // 7 6 5 4 3 2 1

    vector<int> v = { 1,2,3,4,5,6,7,8 };
    reverse(v.begin()+3, v.end());
    for(int x:v)
        cout<<x<<" ";          // 1 2 3 8 7 6 5 4

    string s = "books";
    reverse(s.begin(), s.end());
    cout<<s;                   // skoob
}
```

* it can reverse only those containers which have bidirectional iterator

prev_permutation()

* it makes largest permutation of given seq which is smaller then given sequence. it returns boolean value.

```
int main()
{
    vector<int> v = { 5, 4, 1, 2, 3 };
    prev_permutation (v.begin, v.end() );
    for( int x : v )
        cout << x << " ";
    return 0;
}
```

output : 5 3 4 2 1

* it return false when we pass min possible permutation.

prev_permutation (123); it will return false

Internal Working Algorithm

(i) Traverse from right, find the first element that is not in increasing order. let it be x

(ii) find largest elem on right of x that is smaller than x. let it be y.

(iii) Swap x and y

(iv) Reverse elem after new y

Sample Problems based on sort()

Fractional Knapsack

input:

weight	50	20	30
values	600	500	400

Capacity = 70

output: 1140

```
bool mycmp(pair<int,int> a , pair<int,int> b)
{
    double r1 = (double) a.first/a.second;
    double r2 = (double) b.first/b.second;
    return r1>r2;
}

double Knapsak(int W , Pair<int,int> arr[], int n)
{
    sort(arr ,arr+n,mycmp);
    double res=0.0;
    for(int i=0; i<n; i++)
    {
        if (arr[i].second ≤ W)
        {
            res += arr[i].first;
            W = W - arr[i]-second;
        }
        else
        {
            res += arr[i].first * ((double) W/arr[i].second);
            break;
        }
    }
    return res;
}
```

in subset of m element max-min should be min

Chocolate Distribution Problem

input: {7, 3, 2, 4, 9, 12, 56} m = 3
output: 2

input: {3, 4, 1, 9, 56, 7, 9, 12} m = 5

output: 6 [3 4 7 9 9] → 9 - 3 = 6

```
int minDiff(int arr[], int n, int m)
{
    if(m > n)
        return -1;
    sort(arr, arr+n);          O(n log n)
    int res = arr[m-1] - arr[0];
    for(int i=1; i<=n-m; i++)
        res = min(res, (arr[i+m-1] - arr[i]));
    return res;
}
```

arr[] = {7, 3, 2, 4, 9, 12, 56}

↓ sort()

0	1	2	3	4	5	6
2	3	4	7	9	12	56

res = arr[2] - arr[0]
= 4 - 2 = 2 → min

i = 1 → arr[3] - arr[1]
= 7 - 3 = 4

i = 2 → arr[4] - arr[2]
= 9 - 4 = 5

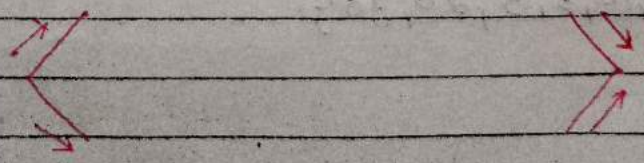
Sort Elements by Frequency if frequency is same put smaller first

```
bool mycmp(pair<int,int> p1, pair<int,int> p2)
{
    if(p1.second == p2.second)
        return p1.first < p2.first;
    return p1.second > p2.second;
}

void sortbyfreq(int arr[], int n)
{
    unordered_map<int,int> m;
    for(int i=0; i<n; i++)
        m[arr[i]]++;

    vector<pair<int,int>> v(m.begin(), m.end());
    sort(v.begin(), v.end(), mycmp());

    for(auto x : v)
        for(int j=0; j<x.second; x++)
            arr[i++] = x.first;
}
```

* Tips for comparator function

< → keep the smaller element first

> → keep the greater element first



Sort element by frequency

(1) No ordering in case of Tie

```
void sortbyfreq(int arr[], int n)
{
    unordered_map<int, int> m;
    for(int i=0; i<n; i++)
        m[arr[i]]++;

    vector<int> freq[n+1];
    for(auto x:m)
        freq[x.second].push_back(x.first);

    int index=0;
    for(int i=n; i>0; i--)
    {
        for(int x: freq[i])
            for(int j=0; j<i; j++)
                arr[index++] = x;
    }
}
```

ex.

arr[] = {10, 5, 20, 5, 10, 10, 30}

m = {{10,3}, {5,2}, {20,1}, {30,1}} not in same order

freq = {{}, {20,30}, {5}, {10}, {}, {}}
(0, 1, 2, 3, 4, 5)

arr[] = {10, 10, 10, 5, 5, 20, 30}

(ii) in case of Tie keep first appearing elem first.

```
void sortbyfreq(int arr[], int n)
{
    unordered_map<int,int> m;
    for(i=0; i<n; i++)
        m[arr[i]]++;

    vector<int> freq[n+1];

    for(int i=0; i<n; i++)
    {
        int f = m[arr[i]];
        if(f!=-1)
        {
            freq[f].push_back(arr[i]);
            m[f] = -1;
        }
    }
    int ind=0;
    for(int i=n; i>0; i--)
        for(int x: freq[i])
            for(int j=0; j<i; j++)
                arr[index++] = x;
}
```

arr[i], d[i] < 2359

Meeting Maximum Guests

input : arr[] = {900, 940}
dept[] = {1000, 1030}

output : 2 (9:40 - 10:00)

input : arr[] = {800, 700, 600, 500}
dep[] = {840, 820, 830, 530}

output : 3 (8:00 to 8:20)

concept for Eficient Solution :

arr[] = {900, 600, 700}
dep[] = {100, 800, 730}

↓

arr[] = {600, 700, 900}
dep[] = {730, 800, 100}

600	A	1
700	A	2
730	D	1
800	D	0
900	A	1
100	D	0

```
int MaxGuest(int arr[], int dep[], int n)
{
    sort(arr, arr+n);
    sort(dep, dep+n);
    int i=1, j=0, res=1, curr=1;

    while(i<n && j<n)
    {
        if(arr[i] <= dep[j])
        {
            curr++;
            i++;
        }
        else
        {
            curr--;
            j++;
        }
        res = max(res, curr);
    }
    return res;
}
```

O(nlogn)

String class in C++

Advantage over c-style strings

- no need to worry about size
- can assign a string later.
- support of operator like +, <, >, ==, <= and >=
- Rich library
- can be converted to cstyle String.

```
int main()
{
    string str = "gfg";
    cout << str.length(); // 3
    str = str + "isbest"; // gfgisbest
                          // 0 1 2 3 4 5 6 7 8

    cout << str.substr(1,5); // fgisb

    cout << str.find("is"); // 3

    int ind = str.find("python");
    if (ind == string::npos)
        cout << "Not found";
    else
        cout << "first occurance at " << ind;
}
```

* if provided string is not there in original string find returns string::npos .

Date / /
Page

* ==, <, >, <=, >= ... operators compares the string character by character.....

```
int main()
{
    string s1 = "abc";
    string s2 = "abc";

    if (s1 == s2)
        cout<<"same";
    else if (s1 < s2)
        cout<<"s1 is smaller";
}
```

* to read space seperated strings use getline()

```
getline(cin, str);

getline(cin, str, '$');
```

→ stop when a $ occurs in string

Find one extra character

input: s1 = "abcd" , s2 = "abecd"
output: e

Method I : sort and compare char by char for i=0 to smaller sting length

Method II : (counting by Hashing)

```
char findextra (string s1, string s2)
{
    int count[256] = {0};
    for(char x : s2)
        count[x]++;
    for(char x : s1)
        count[x]--;
    for(i=0; i<256; i++)
        if(count[i] == 1)
            cout << (char) i;
}
```

Method III : (using Bitwise)

```
char findextra(string s1, string s2)
{
    int n = s1.length();
    for(i=0; i<n; i++)
        res = res ^ s1[i] ^ s2[i];
    res = res ^ s2[n];
    return (char) res;
}
```

Check if a string is panagram

Date / /
Page

* a string is panagram if it contains all 26 letters (capital or small).

```
bool isPangram(string s)
{
    int n = s.length();
    if(n<26)
        return false;
    bool visited[26] = {false};

    for(int i=0; i<n; i++)
    {
        char x = s[i];
        if(x>='a' && x<='z')
            visited[x-'a'] = true;
        if(x>='A' && x<='Z')
            visited[x-'A'] = true;
    }
    for(int i=0; i<26; i++)
        if(visited[i]==false)
            return false;
    return true;
}
```

__builtin_popcount()

* A GCC compiler specific function to count set bits.

```
__builtin_popcount(n);
__builtin_popcountl(n);     // for long int
__builtin_popcountll(n);    // for long long int
```

Tuple in C++

* can store heterogeneous data.

```
int main()
{
    tuple<string, int, string> t = make_tuple("abc", 10, "xyz");
    cout<< get<0>(t) << get<1>(t) << get<2>(t);
    get<0>(t) = "turtle";
```