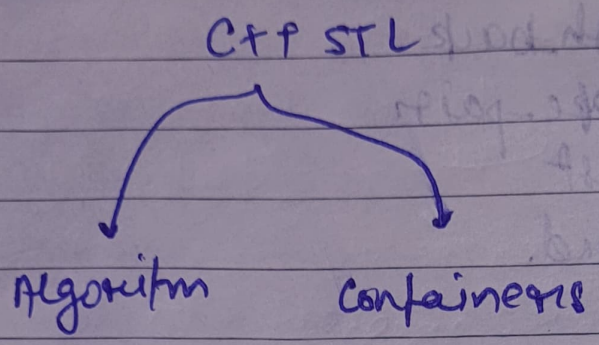
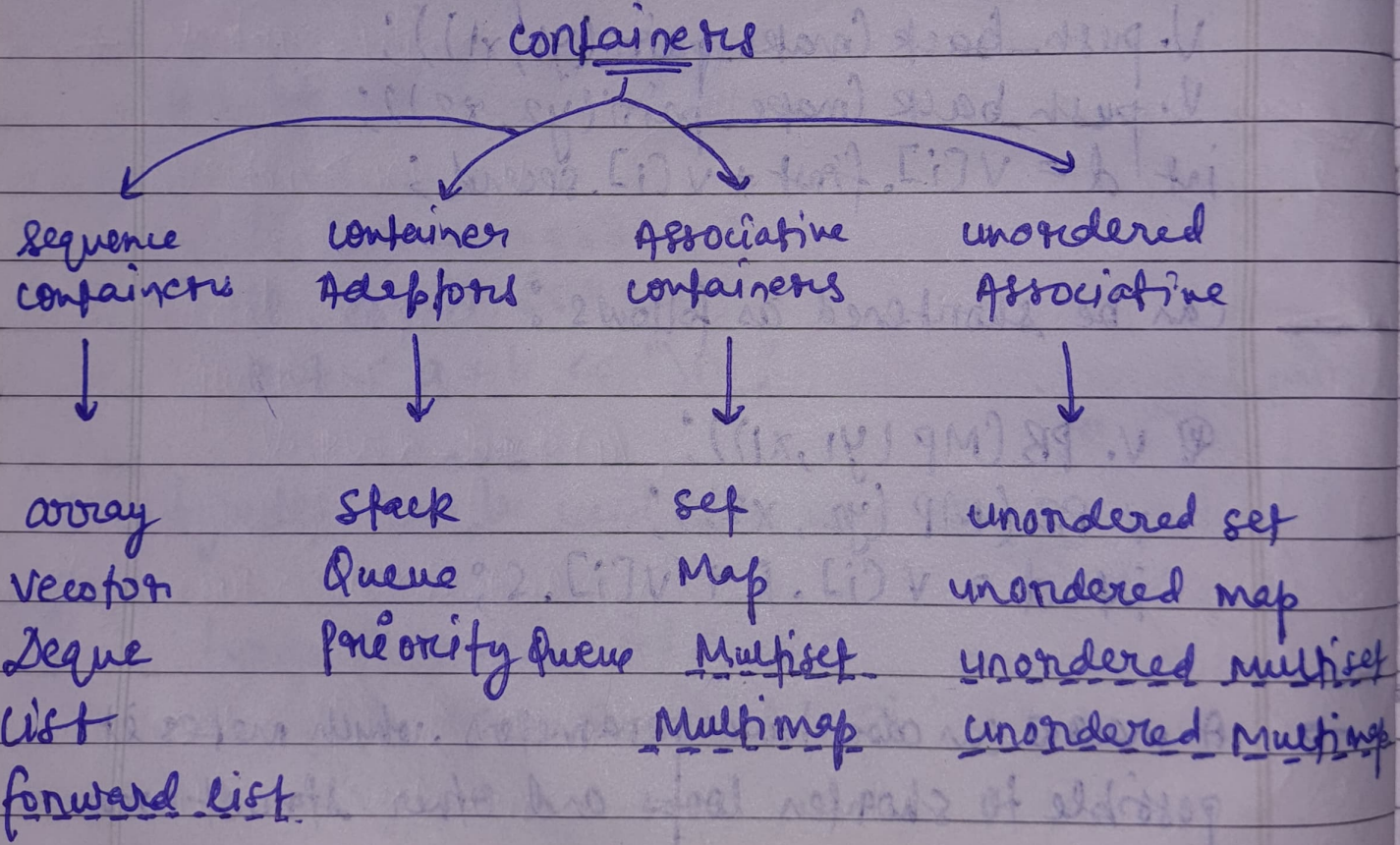
Recursion

Standard template library C++





Page No.
Date :

(A) Sequence Containers

① **Array** :- STL array is also Static array.

* array declaration (create)

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

* ~~capacity~~ a.size(); → return size of the array

* ~~at~~ function

```
cout<<"Element at 2nd Index"<< a.at(2)<<endl;
```

↓ return no. at 2 index.

* empty function

```
cout<<"Empty or not "<< a.empty()<<endl;
```

↓ Return boolean (0 or 1)

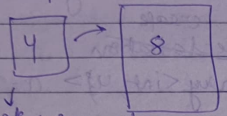
* front() & back() operation

ie

```
cout<<"First Element -> "<< a.front()<<endl;
cout<<"Last Element"<<a.back()<<endl;
```

V.size() → total kitne elements hai
V.capacity() → total kitni memory allocate hui h
↳ kitna space assign hua h element ke liye

② **Vector** :- dynamic array → pehle maan vector ka size 4 hota jaise hi aap 5th element daaloge ye vector apna size double kar dega.



ye ek naya vector banayega double size ka, saare elements copy kar dega aur purana dump ho jaayega.

* Vector initialization

```
vector<int> v;   → initially 0 memory allocate hogi.
```
↳ also `vector<int> a(5,1);` → size, element

* add elements
  pushback
  `v.push_back(1);`

* `cout << "Element at 2nd Index" << v.at(2) << endl;`

* `cout << "front" << v.front() << endl;`
  `cout << "last element" << v.back() << endl;`

* to remove element
  `v.pop_back();` → remove from last

* clear → vector khali hoga but size 0 hoga capacity abhi bhi hogi waisi

```
v.clear();
```

* copy
  `vector<int> last(a);` → a ke saare element last vector mein aa jaayenge

③ **Deque** :- doubly ended queue

→ aap dono end (begin & last) mein push/pop yaani deletion/insertion dono perform kar skte ho.

→ they are not any continous memory stored data but they are multiple fixed static array (thoda complex h)

→ It is dynamic and random access possible [], at() function use kar sakte h.

* `deque<int> d;`

* `d.push_back(1);`
* `d.push_front(2);`
  ] add

* `d.pop_back();`
* `d.pop_front();`
  ] Remove

* d.at(1)

* cout<<"front "<< d.front()<<endl;
  cout<<"back "<< d.back()<<endl;

* cout<<"Empty or not "<< d.empty()<<endl;

* d.erase(d.begin(), d.begin()+1); → range batane parega.

④ List :- double linked list → 2 iterator honge shuru & last ke liye.

→ Random access not permissible (at() & [] use nai kar paenge) → waha tk pahuchne ke liye jana parega.

* list<int> l;

* l.push_back(1);
  l.push_front(2);

* for(int i:l) {
    cout<<i<<" ";
  } → To print.

* l.erase(l.begin()); → time complexity zyada hoga
  → jb iterator daga usko delete ho jaaega.

Page No.
Date :

* l.size();

* list<int> n(0); } → to copy

(B) Container ~~Adaptors~~

(i) Stack :- last in first out

* initialization

Stack<string> s;

* s.push("love");
  s.push("babbar");
  s.push("kumar");

* s.pop(); ← after this.

* cout<<"Top Element" <<s.top() <<endl; → print "babbar"

* cout<<"size of stack" <<s.size() <<endl;

* cout<<"Empty or not" <<s.empty() <<endl;

② Queue :- first in first out.

* queue<string> q;

* q.push("love");
  q.push("Babbar");
  q.push("Kumar");

* q.pop();

* cout<<"size before pop" << q.size() << endl;

* cout<< "first Element" << q.front() << endl;

③ Priority Queue :- → ek aisi queue jiska first element hamesha greatest hoga.

initialization.

* // max-heap → sabse pehle max element aaega
  priority_queue<int> maxi;

  // min_heap → sabse pehle min element aaega.
  priority_queue<int, vector<int>, greater<int> > mini;

Page No.
Date :

* data pust

```
maxi.push(3);
maxi.push(2);
maxi.push(0);
maxi.push(1);
 n= maxi.size();
for(int i=0; i<n; i++){

    cout<<maxi.top()<<" ";
    maxi.pop();

}cout<<endl;
```

(C) Associative Containers :-

(1) Set

→ only unique element store kar skta h (5 times ko delete plus bhi ek hi baar store hoga.)

→ no modify (data daal chuke h phir modify nai kar skte ya delete ya delete kr den.

→ Element sorted order mein milenge.

unordered set → thoda fast hota h (bs element sorted nai hote)

* initialization

set<int> s;

* add elements.

s.insert(5);
s.insert(1);

→ 1 5 store in sorted way.

* print

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

* erase → jo iterators dega use delete kar dega.

s.erase(s.begin());

* to find → we use count() → bataega element h ke nahi

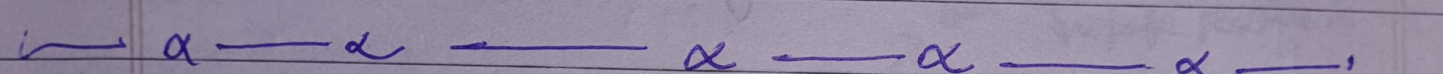
s.count(5); → return in boolean

element ka iterator de dega.

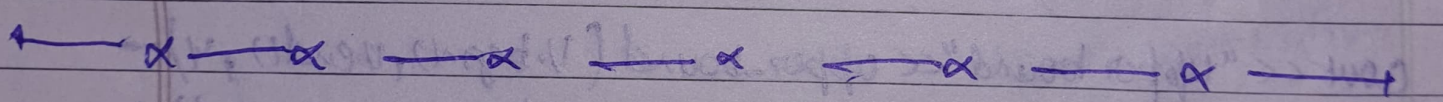
s.find(5);

present hoga toh iterator

(2) Map → see previous pages.



```
int gfg[] = {5, 6, 7, 7, 6, 5, 5, 6};
vector<int> v(gfg, gfg+8); // 5 6 7 7 6 5 5 6
sort(v.begin(), v.end()); // 5 5 6 6 7 7
```



Algorithms

1. Binary search
2. lower/upper bound
3. min/max
4. reverse/rotate
5. sort/swap etc.

vector<int> v; → // 1, 3, 6, 7

① Binary search

cout<<"finding 6"<<

binary_search(v.begin(), v.end(), 6)

<<endl;

② upper bound / lower bound.

cout<<"lower bound"<< lower_bound(v.begin(), v.end(), 6) -
v.begin()<<endl;

cout<<"upper bound"<< upper_bound(v.begin(), v.end(), 4) -
v.begin()<<endl;

③ int a = 8;
int b = 5;

cout<<"max:"<< max(a,b);
cout<<"min:"<< min(a,b);

④ swap(a,b); → also comparisons pe bhi kar sakte h

Page No.
Date :

⑤ String abcd = "abcd";

reverse (abcd.begin(), abcd.end()); → dcba

⑥ Rotate.

rotate (v.begin(), v.begin+1, v.end()); → 7,6,3,1

(1 baar rotate karne h)

to print vector

```
for (i:v)
{
  cout<<i<<" ";
}
```

⑦ Sort → based on introsort

(3 algo ke combination se bante h)

1) Quick sort
2) heap sort
3) insertion sort.

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