$n = \log n$.

Page No.
Date : / /

STL

Algorithms

① find(): linear search

find(arr, arr + size, key); → will give address of key

so int index = find(arr, arr+size, key) - arr

if element not found find will return size.

② binary_search(arr, arr + size, key);

returns boolean value.

③ lower_bound(start, end, key)
first occurence's address or greater than key

④ upper bound (start, end, key)
~~last occurence's address or less than key.~~
first element strictly greater than key.

⑤ sort(start, end)

doesnot include end but (end-1)

To sort in a order that's not ascending
create a comparator function

Page No.
Date : / /

```
bool compare ( int a, int b){
        return a<b;
    }
```

gives ascending but if we return a>=b then we'll descending.

and use as

```
sort ( start, end, compare );
```

Note)

To take a function as a parameter the syntax is

ret void bubble sort (return type (* name) (parameters))

Indian ~~money~~ exchange.

```
bool compare ( int a, int b) {
    return a<=b;
    }
int main (){
    int coins[ ] = {1, 2, 5, 10, 20, 50, 100, 200, 500, 2000};
    int n = 10;
    int money = 100; while ( money ) {
    int lb = lower_bound (coins, coins+n, money, compare ) - coins.-1;
    int m = coins [lb];
    cout << m << " ";
    money -= m;
    }
    return 0;
}
```

Page No.
Date : / /

(6) rotate (start, rotate about this el; end).

eg with vectors

```
vector<int> v{10, 20, 30, 40, 50};
rotate(v.begin(), v.begin()+3, v.end());
```

printing v we'll get

~~10~~ 40, 50, 10, 20, 30.

(7) next_permutation (start, end):
provides the next greater number.

```
int a[] = {1, 2, 3};
next_permutation(a, a+3);
```

a -> {1, 3, 2}.

(8) swap(a, b)
(9) max(a, b)
(10) min(a, b)
(11) reverse(start, end)

Page No.
Date : / /

Pairs

~~pair <int~~

```
pair <int, char> p;

p.first = 10;
p.second = 'B';

pair <int, string> ps = make_pair (100, "Audi");
```

String Class

reading

```
string s;
```

initialising

```
string s1 (string);

char a[] = {'a', 'b', 'c', '\0'};
    String s2 (a);
```

func

1. s.empty ()
2. s.append ("#string"); a s = s + string
3. s.clear()
4. s.length ()
5. s.compare (s1) = 0 if equal, else (s - s1)

char At used as str[i]

s = "I want to have apple juice";

Page No.
Date: / /

(6). int index = s.find("apple");
(7) s.erase(index, length of word);

Iterating over the string

(a) for(auto it = s.begin(); it != s.end(); it++)
cout << (*it) << endl
↳ value of it not address.

(b) for(auto c : s1){
cout << c << ".";
}

~~String sorting~~

```
int main(){
  int n;
  cin >> n;
  cin.get(); // clearing buffer.
  string s[100];
    for(int i=0; i<n; i++){
      getline(cin, s[i]);
    }
sort(s, s+n);
}
```

Page No.
Date : / /

String tokenizer

```
char s[100] = "It is a rainy day";

char *ptr = strtok(s, ' ');

cout << ptr << endl; // => It
while(ptr != NULL){
ptr = strtok(NULL, " ");
cout << ptr << endl // => is, a, rainy, day
}
```

Vectors

```
vector < &data type > v;
vector < int > b(5,10); // five int with value 10
vector <int > x(b.begin(), b.end());
vector <int> d{1,2,3,4,5};
```

Iterating over the vector

(a)
```
for(i=0; i<c.size(); i++){
c[i];
```

(b)
```
for(auto it= b.begin(); it != b.end(); it++)
cout << (*it) << '\n';
```

(c)
```
for( int n : d)
cout << n << '\n';
```

Page No.
Date : / /

More functions

① v. push_back (element)
② v. size ( )
③ v. capacity ( )
④ v. max_size.
⑤ v. pop_back
⑥ v. insert (v.begin + 3, 100) ;
or
v. insert (v.begin + 3, 4, 100); // enter 4 element with value 100
⑦ v. erase (index)
⑧ v. resize (size) // size will change but not the capacity for shrinking
⑨ v. clear ( )
⑩ v. empty ( )
⑪ v. front ( ); first element
⑫ v. back ( ); last "
⑬ v. reserve (& size); reserves ~~to~~ allocate this capacity so initial doubling can be avoided.

Lists (double linked list)

① .push_back ( )
② .push_front ( )
③ .pop_back ( )
④ .pop_front ( )
⑤ .insert (~~Enter~~ index, element);
⑥ . erase (index)
⑦ . remove (element)
⑧ . sort ( )
⑨ . reverse ( )

Page No.
Date : / /

⑥
```
list<int> l;
list<int> l1{1,2,3,4,5};
l1.push_back(6);
for(auto s : l1){
    cout<<s<<"->";
```

To use .erase(index)

```
auto it = l1.begin();
it++;
it++;
l1.erase(it);
```

```
l1.erase(it+2);
```
✗ it's a list not a array.

Can be used for graphs in adjacency list

Stack

```
stack<int> s
```
- .push()
- .pop()
- .top()
- .empty() // to check if it is empty

Page No.
Date : / /

Queue

queue <int> q

- push
- empty ()
- ~~queue~~ front
- pop.

Deque (Doubly ended ~~que~~ queue.

deque <int> d(k) ;

- push_back()
- push_front()
- pop_back
- pop_front

Priority Queue

priority_queue <int> q ; } for max heap

- push (element)
- empty ()
- top () // ~~dis~~ return top element.
- pop ()

priority_queue <int, vector <int>, greater <int>> pq ;

Comparator class

~~max~~ heap
min

Functional object (Functor)

```
class Fun {
  public:
    void operator()(string s){
      cout << "Having fun with " << s;
    }
};

int main() {
  Fun f;
  f("C++");
  return 0;
}
```

f("C++"); → functional operator.

Output => Having fun with C++

WAP to print 3 oldest persons.

```
class Person {
public:
    string name;
    int age;
    Person(){
    }
    Person(string n, int x){
      name = n;
      age = x;
    }
};
```

Page No.
Date : / /

```
class PersonCompare {
public:
    bool operator()(Person A, Person B) {
        return A.age < B.age;
    }
};

int main() {
    int n
    cin >> n;
    priority_queue<Person, vector<Person>, PersonCompare> pq;
    for(int i=0; i<n; i++) {
        string name;
        int age;
        cin >> name >> age;
        Person p(name, age);
        pq.push(p);
    }
    for(int i=0; i<3; i++) {
        person p = pq.top();
        cout << p.name << " " << pq.age << "\n";
        pq.pop();
    }
    return 0
}
```

Page No.
Date : / /

Maps and multimaps

Map -> An associate container that stores mapping b/w key and value.

Map
↙ ↘
map unordered map.

Key -> value
a -> b
c -> d

insert ((k,v)) -> insert ()
query (k) -> find ()
delete (k) -> erase ()

Declaring

map < type1, type2 > name -> map <string, int> m

Inserting

m.insert (make_pair ("Mango", 100));
or we make the pair seperately

Or

m["Banana"] = 20;

Searching

auto it = m.find (fruit);
if (it != m.end ()){
cout << m[fruit];
}

else
cout << "not present";

m.count(fruit) -> 1 if fruit is present else 0

unique keys are stored only once

m.erase.

Deleting a key

m.erase(fruit);

traversing

I
```
for(auto it = m.begin(); it != m.end(); it++){
cout << it->first << it->second<<endl;
```

II
```
for(auto p : m)
cout << p.first << p.second;
```

Page No.
Date : / /

Multimap

Some key can have diff values.

initialisation

multimap < char, string > m;
inserting similar to map

```
auto it = m.lower_bound('b');
auto it2 = m.upper_bound('d');
for (auto it3 = it1; it3 != it2; it3++)
```

input-
b batman
a apple
b boat
a angry
c cat
d dog
e elephant

will printing from batman to elephant.

searching

```
auto f = m.find('a');
if (f -> second == 'angry'){
    print
}
else f++;
```

Note -> STL containers are by default passed by value i.e. the value to avoid this use & at the formal parameter.

Page No.
Date : / /

Unordered maps (implementation of Hashmaps)

Declaration.

unordered_map < string, int > um;

All the f^n of map are applicable here.

Use case

```
class Student {
public:
string first_name;
string last_name;
string roll_no;

Student(string f, string l, string no) {
    first_name = f;
    last_name = l;
    Roll_no = no;
}

bool operator==(const Student &s) const {
return (roll_no == s.roll_no) ? true : false;

class Hashfn {
public
  size_t operator()(const Student &s) const {
  return s.first_name.length() + s.last_name.length();
  }
};
```

```
int main() {

    unordered_map < Student, int, Hashfn > s_map;

    Some entries in the fn.
    Student s1 ("Iresh", "Agrawal", "083");
    {
    s_map[s1] = 100;
    }

iterate
        for (auto s : s_map) {
            cout << s.first.firstname << " " << s.first.rollno << endl;
                    s.first.firstname      s.first.rollno.

    return 0;
}
```

Phone book use case

```
main {
    unordered_map < string, vector < string > phonebook;

phonebook["Iresh"].push_back("9110");
    ~ ——— ———
    ↳
    ↳

for (auto p : phonebook) {
    cout << p.first << " -> ";
        for (string s : p.second) {
            cout << s << ", ";
        }
}
```

Page No.
Date: / /

trie data structure

```
class node {
  public:
    char data;
    unordered_map <char, Node*> child;
    bool terminal;

    Node (char d) {
      data = d;
      terminal = false;
    }
};

class trie {
    Node * root
    int cnt;
    public:

    Trie ( ) {
      root = new Node ('\0');
      cnt = 0;
    }
  void insert (char * w) {
    Node * temp = root
    for (int i=0, w[i] != '\0'; i++) {
      char ch = w[i];
      if ( temp -> child. count (ch)) {
        temp = temp -> child [ch];
      }
    }
```

```
        else {
            Node* n = new Node(ch);
            temp->child[ch] = n;
            temp = n;
        }
    }
    temp->terminal = true;
}

bool find(char* w){
    node* temp = root;
    for(int i=0; w[i]!='\0'; i++){
        char ch = w[i];
        if(temp->children.count(ch)==0){
            return false;
        }
        else{
            temp = temp->children[ch];
        }
    }
    return temp->terminal;
};

int main(){
    trie t;
    char words[][10] = {"a", "apple", "not", "mess"};
    char w[10];
    cin>>w;
    for(int i=0; i<4; i++){
        t.insert(words[i]);
    }
```

Page No.
Date : / /

```
if (t.find(w)){
        cout << "Present";
    }
    else{
        cout << "Absent";
    }
    return 0;
}
```

Sets → Ordered
→ Unordered.

Declaration

```
set<int> s;
```

Initialisation

```
s.insert(element);
```

iterating

```
for(set<int>::iterator it=s.begin(); it!=s.end(); it++){
    cout << *(it) << " ";
}
```

deletion

- `s.erase(element);`

or

```
auto it = s.find(element);
s.erase(it);
```

Page No.
Date : / /

Policy Based data structures

header files

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
```

initialisation

```
tree <int, null_type, less<int>, rb_tree_tag, tree_order_
                                    statistics_node_update>
        pbds st;                                    pbds
```

functions

```
st
pbds.insert();
pbds st.find_by_order(no.);
st.order_of_key(key);
```