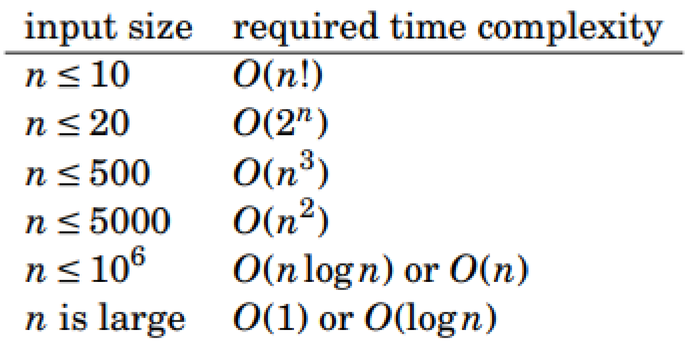
OCI

C++ Template

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

int main(){
        sync_with_stdio(0);
        cin.tie(0);
}
```

Time Complexity



input size	required time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Sorting

Bubble Sort

```
for(int i=0;i<n;i++){
        for(int j=0;j<n-1;j++){
                if(array[j]>array[j+1]){
                        swap(array[j],array[j+1]);
                }
        }
}

// O(n^2)
```

Sorting in C++

```
vector <int> v= {4,2,4,3,5,8,3};
sort(v.begin(),v.end()); // arranges the vector in increasing order
sort(v.rbegin(),v.rend()); // arranges the vector in decreasing order

int a[]={4,2,4,3,8,4};
sort(a,a+n) // where n is the size of the array

// O(n*log(n))
```

Binary Search

Implementation

```
int a=0,b=n-1;
while(a<=b){
        int k = (a+b)/2;
        if(array[k]==x){
                // x found at index k
        }
        if(array[k]>x){
                b=k-1;
        }
        else{
                a=k+1;
        }
}
```

C++ functions

Bounds

```
auto k = lower_bound(v.begin(),v.end(),x);
if(k<v.size() && v[k]==x){
        // x found at index k
}

auto k = lower_bound(array,array+n,x)-array;
if(k<n && array[k]==x){
        // x found at index k
}
```

```
// The following code counts the number of elements whose value is x

auto a = lower_bound(v.begin(),v.end(),x);
auto b = upper_bound(v.begin(),v.end(),x);
cout<<b-a<<"\n";
```

Smallest Solution

```
int x = -1;

for(int b=z;b>=1;b/=2){
        while(!ok(x+b)){
                x+=b;
        }
}
int k = x+1;
```

Data structures

Dynamic Array

```
vector <int> v;

v.push_back(3); // [3]
v.push_back(5); //[3,5]

cout<<v[0]; // 3
cout<<v[1]; // 5

for(int i=0;i<v.size();i++){
        cout<<v[i]<<"\n"; //print all elements in the dynamic array
}

// other way to do is

for(auto x : v){
        cout<<x<<"\n";
}

cout<<v.back(); // returns the last element in vector
v.pop_back(); // removes the last element

// Initialize a vector
```

```
vector <int> v (10,0) //size 10 , initial value 0
```

String

```
string a = "hatti";
string b = a+a;

cout<<b; //"hattihatti"
b[5] = 'v';
cout << b << "\n"; // hattivatti

string c = b.substr(k,x); //returns the substring that begins in at position k and has lenght x
cout<<c<<"\n"; //tiva

find("tiva") ; // finds the position of the first ocurrence of a substring
```

Sets

```
set<int> s;
s.insert(3);
s.insert(2);

cout<<s.count(3)<<"\n"; // 1
cout<<s.count(4)<<"\n"; // 0

s.erase(3);

// An important property of sets is that all their elements are distinct.
```

Multisets

```
multiset <int> s;
s.insert(5);
s.insert(5);
s.insert(5);
cout<<s.count(5)<<"\n"; //3

s.erase(5);
cout<<s.count(5)<<"\n"; //0

s.erase(s.find(5));
```

```
cout<<s.count(5)<<"\n"; //2
```

Maps

```
map<string,int>m;
m["monkey"]=4;
m["banana"]=3;

cout<<m["banana"]<<"\n"; //3

cout<<m["aarfaf"]<<"\n"; // 0
//If the value of a key is requested but the map does not contain it, the key is automatically added to the map with a default value.

if(m.count("aarfaf")){ //count checks if a key exist in a map
        //key exist
}

for(auto x : m){
        cout<<x.first<<" "<<x.second<<"\n";
}
```

Iterators

```
//Ranges
sort(v.begin(),v.end());
reverse(v.begin(),v.end());
random_shuffle(v.begin(),v.end());
```

Stack

```
stack <int> s;
s.push(3);
s.push(2);
s.push(5);
cout<<s.top(); //5
s.pop(); //delete 5

//First in Last Out
```

Queue

```
queue <int> q;
q.push(3);
q.push(2);
q.push(5);
cout<<q.front(); //3
q.pop(); //delete 3;

// First in First Out
```

Priority queue

```

//By default, the elements in a C++ priority queue are sorted in decreasing order,
and it is possible to find and remove the largest element in the queue.

priority_queue <int> q;
q.push(3);
q.push(5);
q.push(7);

cout<<q.top(); // 7
q.pop();
```

Graphs

```
vector<int> adj[N];

adj[1].push_back(2); //this means that the node 1 has an edge with node 2


//the following loop goes through all nodes to which we can move from node s

for( auto u : adj[s]){ //
        // process node u
}
```

DFS

```
vector <int> adj[N];
bool visited [N];
```

```
void dfs(int s){
        if(visited[s]){
                return;
        }
        visited[s]=true;
        //process node s;
        for(auto u : adj[s]){
                dfs(u);
        }
}
```

BFS

```
queue <int> q;
bool visited [N];
int distance [N];

visited[x]=true;
distance[x]=0;
q.push(x);
while(!q.empty){
        int s = q.front; q.pop;
        //procces node s;
        for(auto u : adj[s]){
                if(visited[u]){
                        continue;
                }
                visited[u]=true;
                distance[u]=distance[s]+1;
                q.push(u);
        }
}

// Connectivity check
```

Dijkstra

```
for(int i=1;i<=n;i++)distance[i]=INF;
distance[x]=0;
q.push({0,x});

while(!q.empty()){
        int a = q.top().second;q.pop();
        if(processed[a])continue;
        processed[a]=true;
        for(auto u : adj[a]){
                int b=u.first, w=u.second;
                if(disntace[a]+w<distance[b]){
                        distance[b]=distance[a]+w;
                        q.push({-distance[b],b});
                }
        }
}
```