Data Structures Using C++: STL

- **The STL is part of the standard C++ library**
- **The STL contains many class and function templates that may be used to store, search, and perform algorithms on data structures**

Note: focussed on speed and optimisation of general purpose tasks.

- **The STL consists of:**
  - **Container classes (data structures)**
  - **Iterators**
  - **Algorithms**

Containers

- Sequence Containers - store sequences of values
  - ordinary C++ arrays
  - vector
  - deque
  - list
- Associative Containers - use "keys" to access data rather than position (Account #, ID, SSN, ...)
  - set
  - multiset
  - map
  - multimap
- Container Adapters - specialized interfaces to general containers
  - stack
  - queue
  - priority_queue

Iterators

- We need a way to iterate over the values stored in a container
- Iteration with C++ arrays:

```
const int SIZE = 10;
string names[SIZE];

for (int x=0; x < SIZE; ++x) {
   cout << names[x] << endl;
}

OR

string * end = names + SIZE;
for (string * cur = names; cur < end; ++cur) {
   cout << *cur << endl;
}
```

Iterators

- How do you iterate over the values stored in an STL container?
- For vectors and deques, you can iterate like this:

```
vector<string> names;

names.push_back("fred");
names.push_back("wilma");
names.push_back("barney");
names.push_back("betty");

for (int x=0; x < names.size(); ++x) {
   cout << names[x] << endl;
}
```

- This style of iteration doesn't work for the other container types

Iterators

- How do you know when you've reached the end of the container's values?
- All containers have a method named `end` that returns a special iterator value that represents the end of the container (similar to a null pointer)

```
set<string> names;

names.insert("fred");
names.insert("wilma");
names.insert("barney");
names.insert("betty");

set<string>::iterator it;
for (it = names.begin(); it != names.end(); ++it) {
   cout << *it << endl;
}
```

Iterators

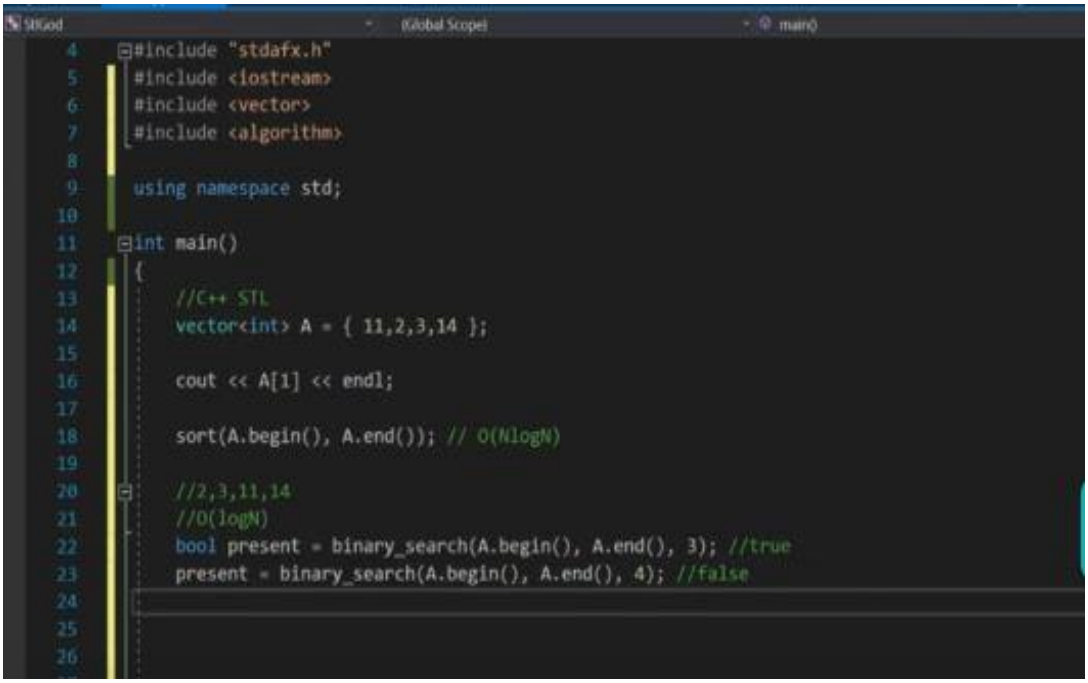
- You can also traverse a container in reverse order using reverse iterators and the `rbegin` and `rend` container methods

```
set<string> names;

names.insert("fred");
names.insert("wilma");
names.insert("barney");
names.insert("betty");

set<string>::reverse_iterator rit;
for (rit = names.rbegin(); rit != names.rend(); ++rit) {
   cout << *rit << endl;
}
```

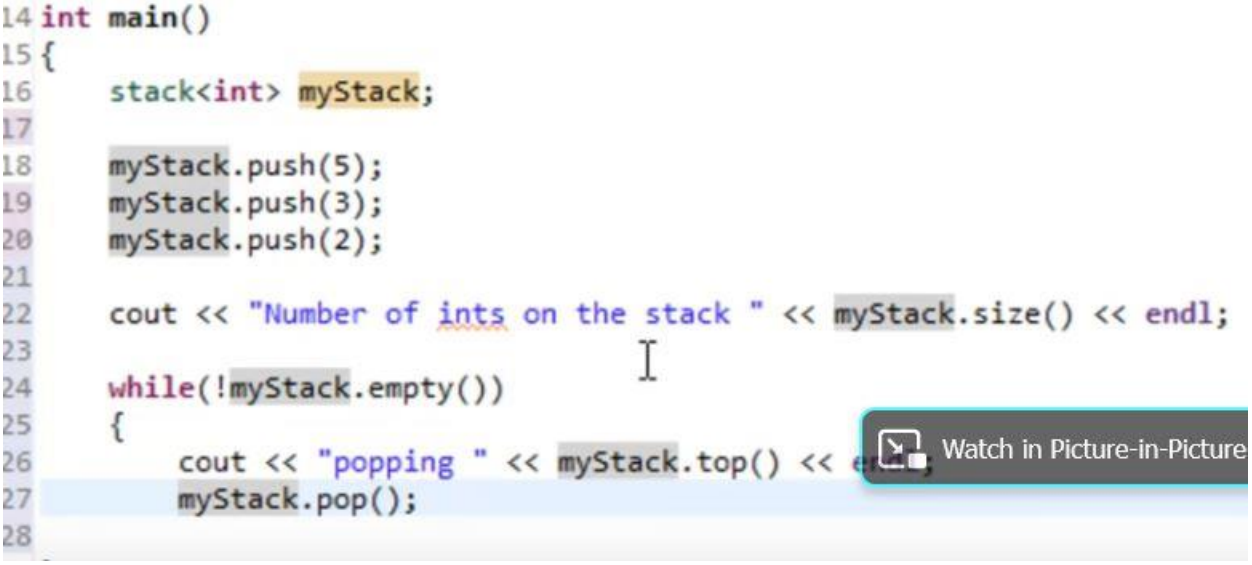
Algorithms

- The STL provides many functions that can operate on any STL container
- These functions are called algorithms
- Some STL algorithms only work on certain containers
- #include <algorithm>

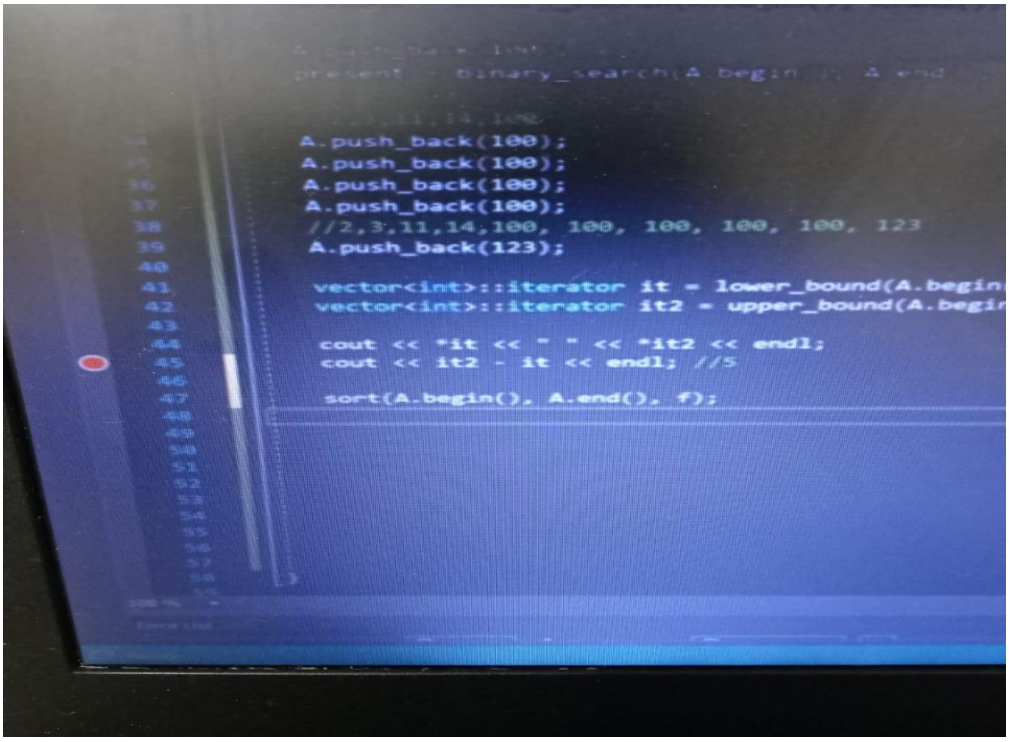
```
vector<string> names;

names.push_back("fred");
names.push_back("wilma");
names.push_back("barney");
names.push_back("betty");

unique(names.begin(), names.end());
sort(names.begin(), names.end());

vector<string>::iterator it;
for (it = names.begin(); it != names.end(); ++it) {
   cout << *it << endl;
`
```

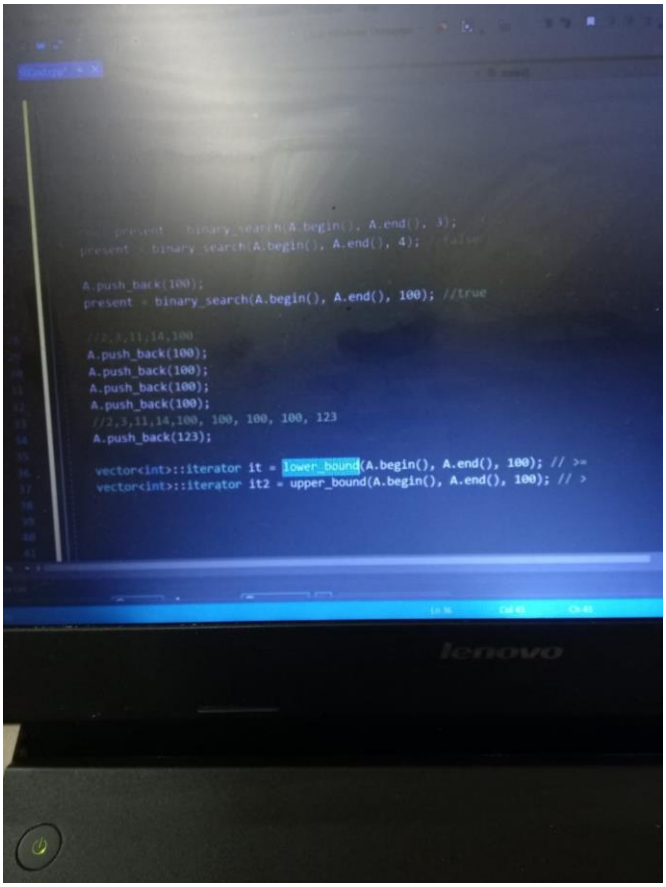


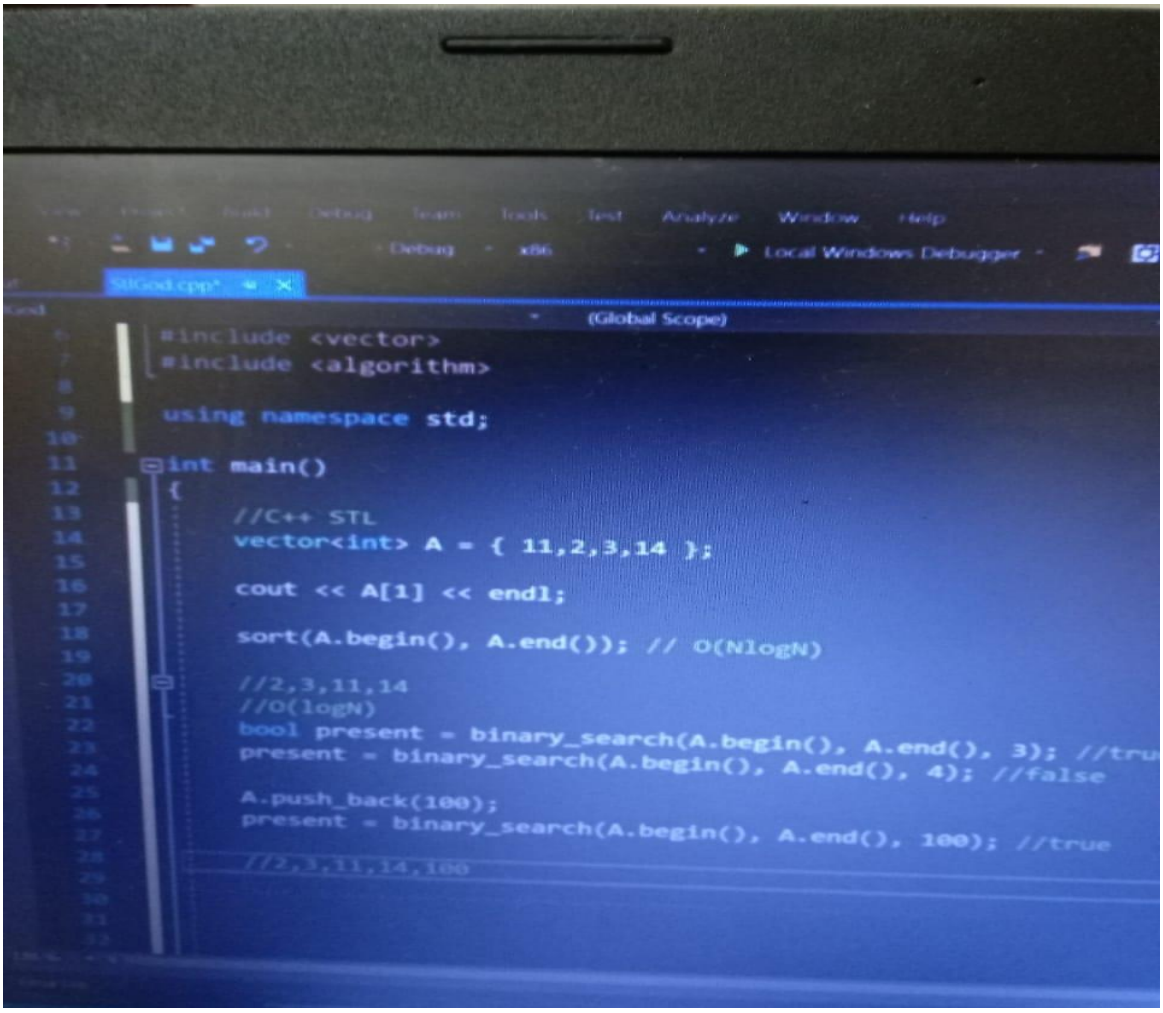
```cpp
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    //C++ STL
    vector<int> A = { 11,2,3,14 };

    cout << A[1] << endl;

    sort(A.begin(), A.end()); // O(NlogN)

    //2,3,11,14
    //O(logN)
    bool present = binary_search(A.begin(), A.end(), 3); //true
    present = binary_search(A.begin(), A.end(), 4); //false
```

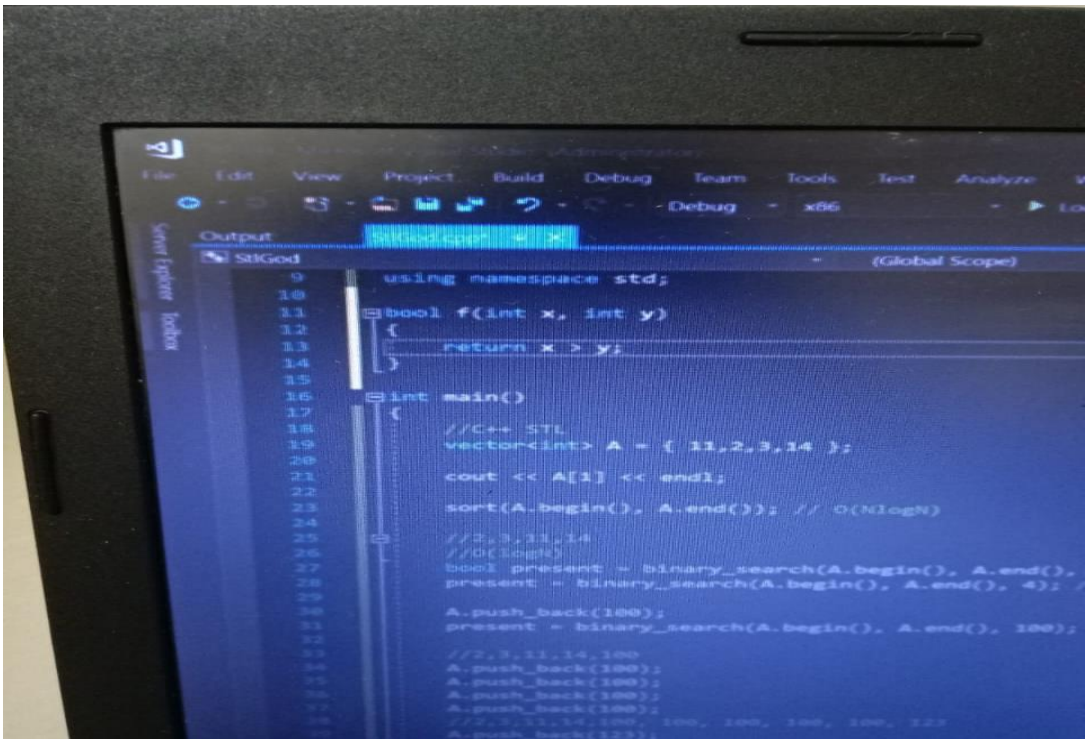


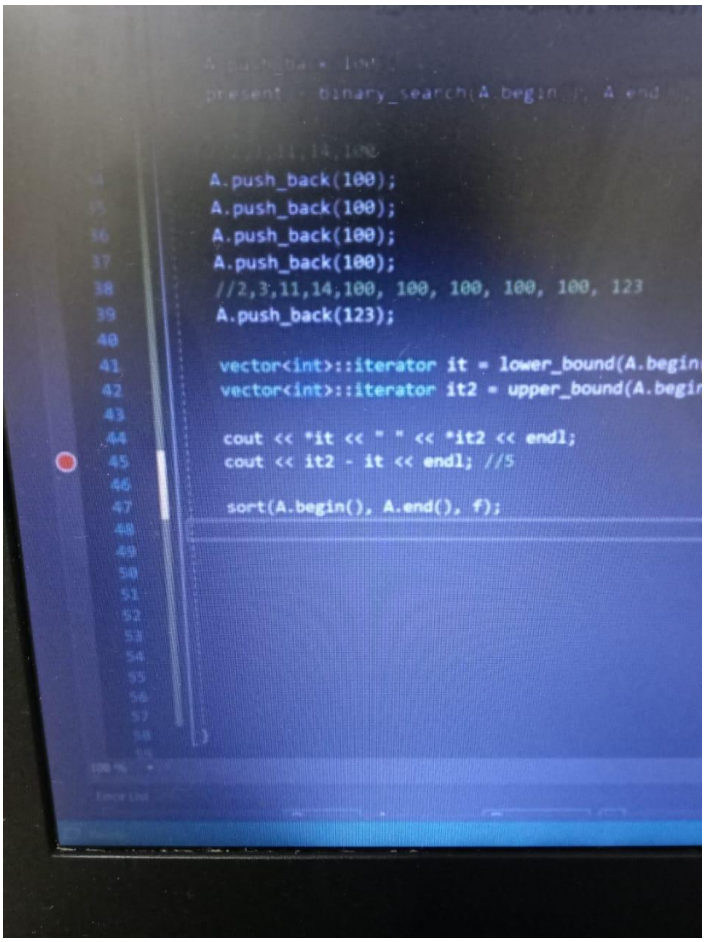
```cpp
int main()
{
    stack<int> myStack;

    myStack.push(5);
    myStack.push(3);
    myStack.push(2);

    cout << "Number of ints on the stack " << myStack.size() << endl;

    while(!myStack.empty())
    {
        cout << "popping " << myStack.top() << e
        myStack.pop();
```

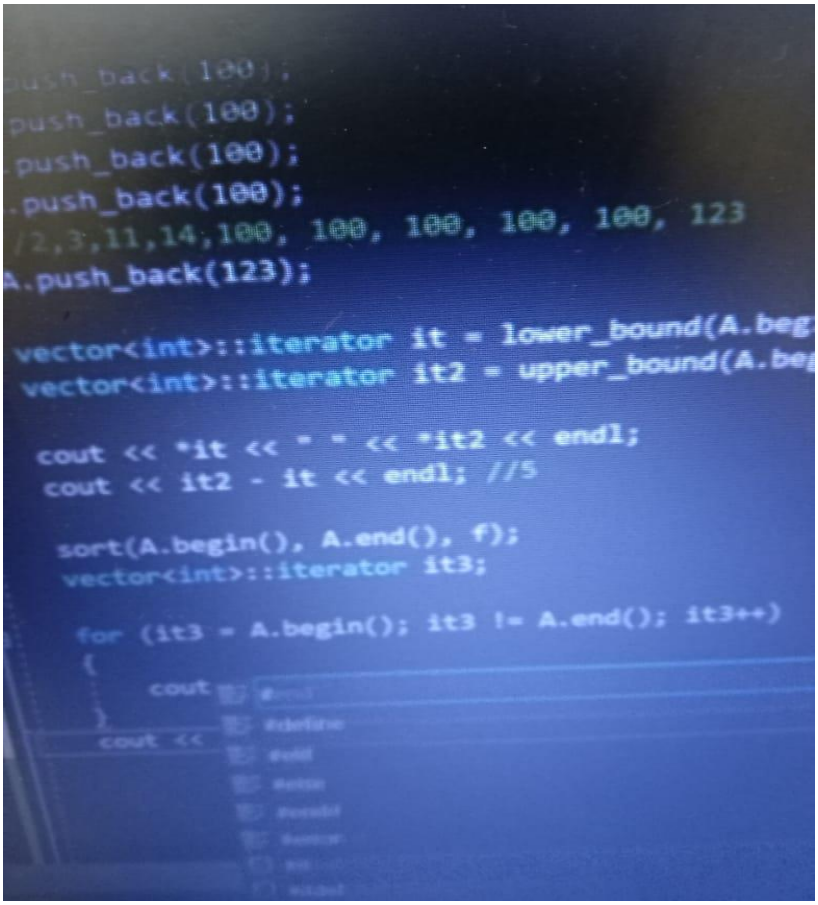

A.push_back(100);
A.push_back(100);
A.push_back(100);
A.push_back(100);
//2,3,11,14,100, 100, 100, 100, 100, 123
A.push_back(123);
vector<int>::iterator it = lower_bound(A.begin
vector<int>::iterator it2 = upper_bound(A.begi
cout << *it << " " << *it2 << endl;
cout << it2 - it << endl; //5
sort(A.begin(), A.end(), f);

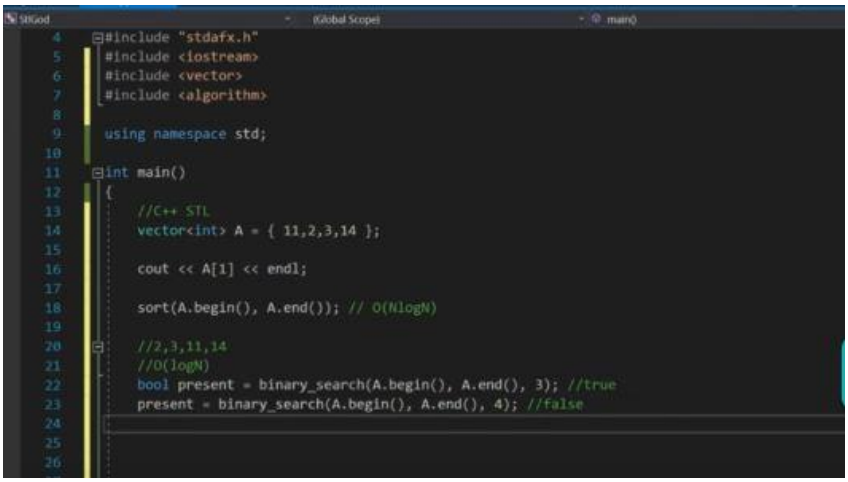

present = binary_search(A.begin(), A.end(), 4);
A.push_back(100);
present = binary_search(A.begin(), A.end(), 100); //true
//2,3,11,14,100
A.push_back(100);
A.push_back(100);
A.push_back(100);
A.push_back(100);
//2,3,11,14,100, 100, 100, 100, 123
A.push_back(123);
vector<int>::iterator it = lower_bound(A.begin(), A.end(), 100); // >=
vector<int>::iterator it2 = upper_bound(A.begin(), A.end(), 100); // >


Debug
x86
Local Windows Debugger
(Global Scope)
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
//C++ STL
vector<int> A = { 11,2,3,14 };
cout << A[1] << endl;
sort(A.begin(), A.end()); // O(NlogN)
//2,3,11,14
//O(logN)
bool present = binary_search(A.begin(), A.end(), 3); //tru
present = binary_search(A.begin(), A.end(), 4); //false
A.push_back(100);
present = binary_search(A.begin(), A.end(), 100); //true
//2,3,11,14,100


Debug
x86
Output
(Global Scope)
using namespace std;
bool f(int x, int y)
return x > y;
int main()
//C++ STL
vector<int> A = { 11,2,3,14 };
cout << A[1] << endl;
sort(A.begin(), A.end()); // O(NlogN)
//2,3,11,14
//O(logN)
bool present = binary_search(A.begin(), A.end(),
present = binary_search(A.begin(), A.end(), 4);
A.push_back(100);
present = binary_search(A.begin(), A.end(), 100);
//2,3,11,14,100
A.push_back(100);
A.push_back(100);
A.push_back(100);
A.push_back(100);


A.push_back(100);
A.push_back(100);
A.push_back(100);
A.push_back(100);
//2,3,11,14,100, 100, 100, 100, 100, 123
A.push_back(123);
vector<int>::iterator it = lower_bound(A.begin
vector<int>::iterator it2 = upper_bound(A.begi
cout << *it << " " << *it2 << endl;
cout << it2 - it << endl; //5
sort(A.begin(), A.end(), f);


push_back(100);
push_back(100);
push_back(100);
push_back(100);
2,3,11,14,100, 100, 100, 100, 100, 123
A.push_back(123);
vector<int>::iterator it = lower_bound(A.beg
vector<int>::iterator it2 = upper_bound(A.beg
cout << *it << " " << *it2 << endl;
cout << it2 - it << endl; //5
sort(A.begin(), A.end(), f);
vector<int>::iterator it3;
for (it3 = A.begin(); it3 != A.end(); it3++)
{
cout
}
cout <<


SBGod
(Global Scope)
main()
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
//C++ STL
vector<int> A = { 11,2,3,14 };
cout << A[1] << endl;
sort(A.begin(), A.end()); // O(NlogN)
//2,3,11,14
//O(logN)
bool present = binary_search(A.begin(), A.end(), 3); //true
present = binary_search(A.begin(), A.end(), 4); //false