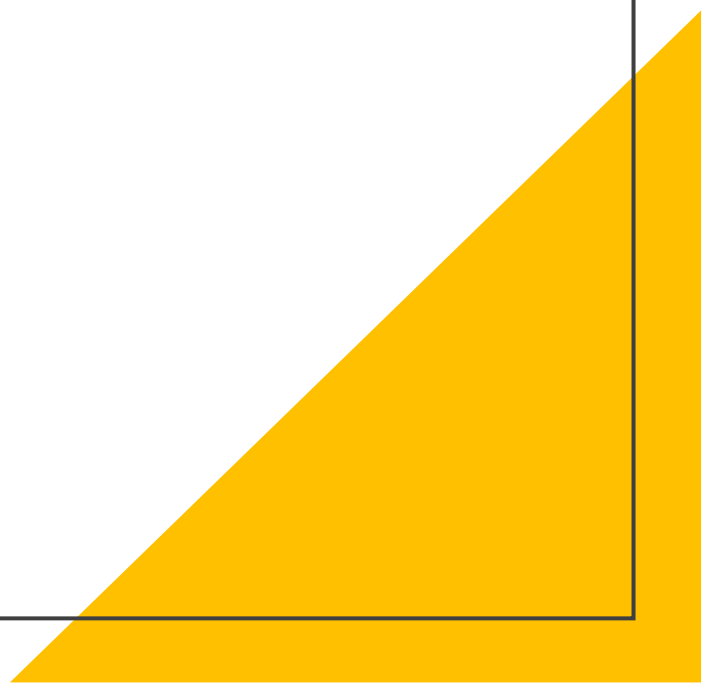


程式各種問題 與 STL 1

日月卦長



時間限制

A. Another A+B Problem

time limit per test: 1 second

memory limit per test: 1024 megabytes

input: standard input

output: standard output

A. DFS Order

time limit per test: 3 seconds

memory limit per test: 1024 megabytes

input: standard input

output: standard output

Time Limit Exceeded (TLE)

Time Limit Exceeded	2022-NCPC-Pre-B	1999ms
Time Limit Exceeded	2022-NCPC-Pre-B	2000ms

Time Limit Excee	W2-3	997ms	3MB
Time Limit Excee	W2-3	996ms	3MB
Time Limit Excee	W2-3	997ms	3MB
Time Limit Excee	W2-3	996ms	3MB

CPU 的速度 – 每秒鐘可以做幾次運算

產品名稱	狀態	推出日期	核心數量	最大超頻	處理器基礎頻率	快取記憶體	散熱設計功耗	處理器繪圖 ‡
<input checked="" type="checkbox"/> Intel® Core™ i5-7300HQ 處理器 (6M 快取記憶體，最高 3.50 GHz)	Launched	Q1'17	4	3.50 GHz	2.50 GHz	6 MB	45 W	Intel® HD Graphics 630

3.5GHz → 每秒最多執行 3.5×10^9 個指令

測試：你的電腦要跑多久？

```
double speed(int iter_num) {  
    const int block_size = 1024;  
    volatile int A[block_size];  
    auto begin = chrono::high_resolution_clock::now();  
    while (iter_num--)  
        for (int j = 0; j < block_size; ++j)  
            A[j] += j;  
    auto end = chrono::high_resolution_clock::now();  
    chrono::duration<double> diff = end - begin;  
    return diff.count();  
}
```

如何計算程式的速度

大概就好(使用複雜度)

- 簡單快速
- 忽略不必要的常數

精確計算每個指令

- 非常難算

複雜度 – small o notation

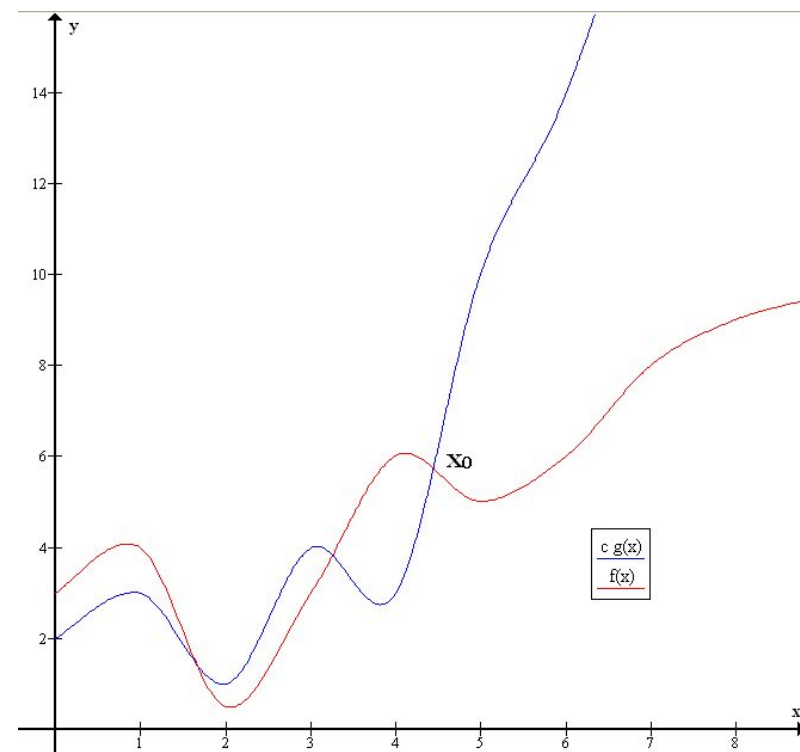
- $\forall k > 0, \exists n_0, \forall n > n_0: |f(n)| < k \times g(n)$

- 那麼我們就可以說

$$f(x) = o(g(x))$$

- Ex: $n^2 + n = o(n^3)$

- 回去翻 Algorithm 課本



複雜度 – big O notation (最常見)

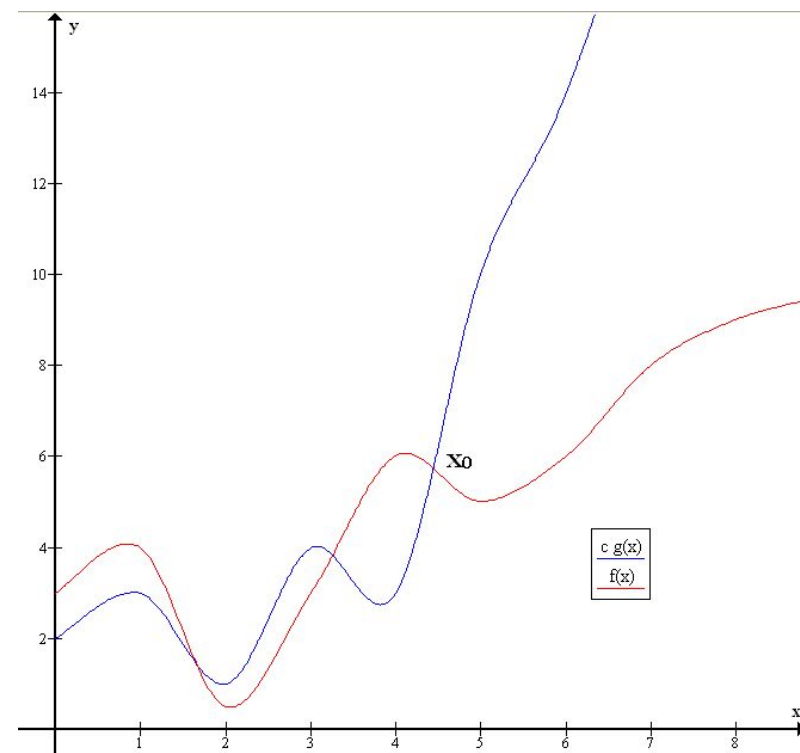
- $\exists k > 0, \exists n_0, \forall n > n_0: |f(n)| \leq k \times g(n)$

- 那麼我們就可以說

$$f(x) = O(g(x))$$

- Ex: $n^2 + n = O(n^2)$

- 回去翻 Algorithm 課本



複雜度 – big Ω notation (一般不會討論)

- $\exists k > 0, \exists n_0, \forall n > n_0: |f(n)| \geq k \times g(n)$

- 那麼我們就可以說

$$f(x) = \Omega(g(x))$$

- Ex: $n^2 + n = \Omega(n)$

複雜度 – big Θ notation (最嚴格)

- 若 $f(x) = O(g(x))$, 且 $g(x) = \Omega(f(x))$
則

$$f(n) = \Theta(g(n))$$

- Ex: $n^2 + n = \Theta(n^2)$

複雜度的比較

- 若 $f(x) = O(g(x))$, 但 $g(x) \neq O(f(x))$
則可以說 $O(f(n)) < O(g(n))$
- $O(n^n) > O(n!) > O(3^n) > O(2^n) > O(n^4) > O(n^2) > O(n \log n)$

複雜度一些定理

1. $O(a_0 + a_1n + a_2n^2 + \cdots + a_kn^k) = O(n^k)$

Ex: $O(10n^2 + 999n + 7122) = O(n^2)$

2. $O(f(n) + g(n)) = \max\{O(f(n)), O(g(n))\}$

Ex: $O(n^2 + 2^n) = O(2^n)$

3. $O(f(n) \times g(n)) = O(O(f(n)) \times O(g(n)))$

Ex: $O((99n^2 + n) \times (7n^3 + n^2 + 2n + 2)) = O(n^5)$

常數時間

- 如果某個步驟執行的基本運算次數是固定的
我們稱其為常數時間
- 變數存取、基本變數運算 ($+-\times\div$) 等通常會被當成常數時間
- 記為 $O(1)$
- $O(1) + O(f(n)) = O(f(n))$
- $O(1) \times O(f(n)) = O(f(n))$

範例 code

```
int n, g[105][105];
scanf("%d", &n);
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        scanf("%d", &g[i][j]);
    }
}
for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (g[i][k] + g[k][j] < g[i][j])
                g[i][j] = g[i][k] + g[k][j];
```

- 這個程式的會先有 $n^2 + 1$ 次的輸入
然後有 n^3 次的運算
- 總複雜度為 $O(n^2 + 1) + O(n^3) = O(n^3)$
- 其實大部分的程式都可以直接用看的就能
看出他的時間複雜度是多少
非常方便

常見不會 TLE 的情況

[illegible]

時間複雜度與迴圈數量

```
void gnomeSort(int s[], int n) {  
    for (int i = 0; i < n; ++i) {  
        if (i && s[i] < s[i - 1]) {  
            swap(s[i], s[i - 1]);  
            i -= 2;  
        }  
    }  
}
```



$O(n^2)$

時間複雜度與遞迴

回溯演算法

- 八皇后等
- 通常難以估計
- 遇到時要靠經驗猜測是否會 TLE

分治演算法

- 列出遞迴關係式後用一些工具 (例如主定理) 求解

動態規劃演算法

- 未來會討論

主定理 (通常比賽時都亂猜沒人用)

設遞迴關係式 $T(n) = a \times T\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1$

情形一

- 則

情形二

- 則

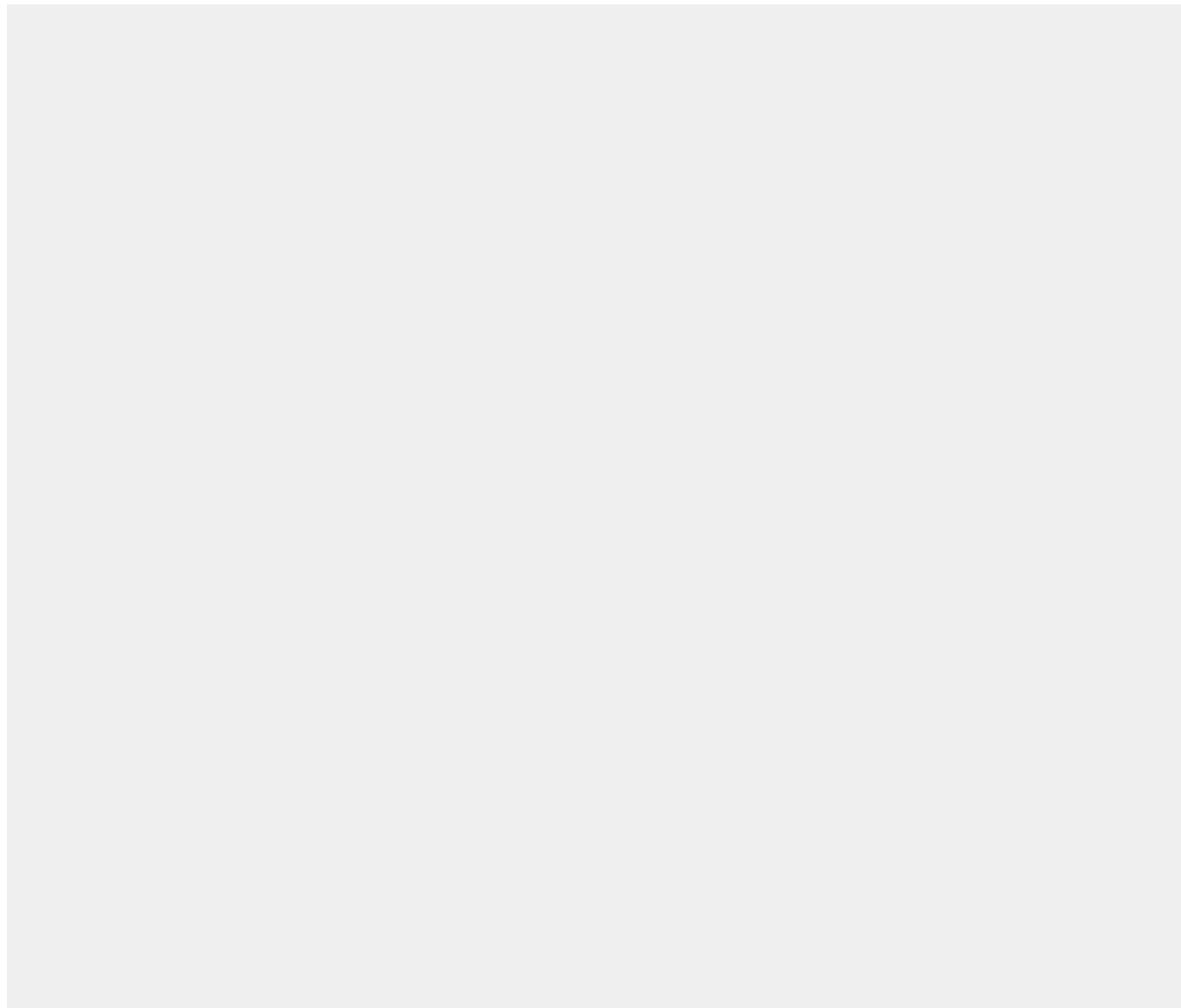
情形三

- , 且
- 則

Runtime error (RE)

Runtime Error	bracketmatching	1ms	3MB
Runtime Error	bracketmatching	0ms	3MB
Runtime Error	bracketmatching	1ms	3MB
Runtime Error	bracketmatching	1ms	3MB
Runtime Error	bracketmatching	1ms	3MB

Stack 與 Heap



Stack Size 測試

```
size_t block_size, bound;
void stack_size_dfs(size_t depth = 1) {
    if (depth >= bound)
        return;
    int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
    memset(ptr, 'a', block_size);
    cout << depth << endl;
    stack_size_dfs(depth + 1);
}

void stack_size_and_runtime_error(size_t block_size, size_t bound = 1024) {
    ::block_size = block_size;
    ::bound = bound;
    stack_size_dfs();
}
```

Linux 可以直接求出 Stack size limit

```
#include <sys/resource.h>
void print_stack_limit() { // only work in Linux
    struct rlimit l;
    getrlimit(RLIMIT_STACK, &l);
    cout << "stack_size = " << l.rlim_cur << " byte" << endl;
}
```

Runtime error ☐

Time Limit
Exceeded?



製造 Segmentation fault

```
void runtime_error_1() {  
    // Segmentation fault  
    int *ptr = nullptr;  
    *(ptr + 7122) = 7122;  
}
```

```
void runtime_error_2() {  
    // Segmentation fault  
    int *ptr = (int *)memset;  
    *ptr = 7122;  
}
```


製造 invalid pointer

```
void runtime_error_3() {  
    // munmap_chunk(): invalid pointer  
    int *ptr = (int *)memset;  
    delete ptr;  
}
```

```
void runtime_error_4() {  
    // free(): invalid pointer  
    int *ptr = new int[7122];  
    ptr += 1;  
    delete[] ptr;  
}
```

製造 Floating point exception

```
void runtime_error_5() {  
    // maybe illegal instruction  
    int a = 7122, b = 0;  
    cout << (a / b) << endl;  
}
```

```
void runtime_error_6() {  
    // floating point exception  
    volatile int a = 7122, b = 0;  
    cout << (a / b) << endl;  
}
```

製造 Aborted

```
void runtime_error_7() {  
    // call to abort.  
    assert(false);  
}
```

測機時間

- 測機時可以用最簡單的題目檢查

1. Judge 執行速度
2. Stack size limit
3. RE 會不會被判斷成 TLE

- 打包整理過的測機模板

- <https://gist.github.com/jacky860226/cbd6d9d0d4e36a48aa3487d4bf578071>

STL 1

bitset, vector, deque 以及一些常用函數

STL 簡介

Container

- <array>
- <deque>
- <forward_list>
- <list>
- <map>
- <queue>
- <set>
- <stack>
- <unordered_map>
- <unordered_set>
- <vector>

Other

- <algorithm>
- <bitset>
- <complex>
- <limits>
- <string>
- <tuple>
- <utility>

STL Container

Array

- <array>
- <vector>
- <deque>
- <queue>
- <stack>

Linked List

- <list>
- <forward_list>

很慢
沒事不要用

Binary Search Tree

- <map>
- <set>

Hash Table

- <unordered_map>
- <unordered_set>

Template 寫給程式設計師的程式

```
template <class T> // template funtion
T func(T x) {
    return x * 2;
}
```

```
template <class T> // template class
struct ST {
    T x;
    ST(T x) : x(x) {}
};
```

```
int main() {
    double a = 0.7122;
    int b = 7122;
    cout << func(a) << ' ' << func(b) << endl;

    ST<string> s("ABC");
    cout << s.x << endl;
}
```


std::bitset 位元運算的好工具

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

int main() {
    bitset<100> BIT(712271227122LL);
    cout << BIT << endl;
    for (size_t i = 0; i < BIT.size() / 2; ++i)
        BIT[i] = 1;
    cout << BIT << endl;
    cout << (~BIT ^ bitset<100>("1101")) << endl;
    return 0;
}
```

std::bitset 枚舉並印出子集合

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

int main() {
    const int n = 4;
    for (int i = 0; i < (1 << n); ++i) {
        cout << bitset<n>(i) << endl;
    }
    return 0;
}
```

```
jacky860226@
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

std::tuple 一口氣儲存一堆東西

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

int main() {
    tuple<int, double, char> mytuple;
    mytuple = make_tuple(10, 2.6, 'a');

    cout << get<0>(mytuple) << endl;
    cout << get<1>(mytuple) << endl;
    cout << get<2>(mytuple) << endl;
    return 0;
}
```

std::tuple 一口氣儲存一堆東西

```
#include <iostream>
#include <tuple>

int main() {
    std::tuple<int, double, char> mytuple;
    mytuple = std::make_tuple(10, 2.6, 'a');

    int myint;
    char mychar;
    std::tie(myint, std::ignore, mychar) = mytuple;

    std::cout << "myint contains: " << myint << '\n';
    std::cout << "mychar contains: " << mychar << '\n';
    return 0;
}
```

(utility) std::pair 儲存兩個東西

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

int main() {
    pair<int, double> P(1, 0.2);
    cout << P.first << ' ' << P.second << endl;

    P = make_pair('a', 2);

    int A;
    double B;
    tie(A, B) = P;
    cout << A << ' ' << B << endl;
    return 0;
}
```

C++17

Structured binding declaration

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

int main() {
    {
        int A[2] = {1, 2};
        auto [x, y] = A;
        auto &[xr, yr] = A;
        xr = 3, yr = 4;
        cout << x << ' ' << y << endl;
        cout << A[0] << ' ' << A[1] << endl;
    }
    {
        pair<int, double> P(1, 0.2);
        auto [x, y] = P;
    }
    {
        tuple<int, double, char> T(1, 0.2, 'a');
        auto [x, y, z] = T;
    }
    return 0;
}
```

(utility) std::swap 交換兩個東西

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

int main() {
    int x = 10, y = 20;
    swap(x, y);
    cout << x << ' ' << y << endl;

    int foo[4];
    int bar[] = {10, 20, 30, 40};
    swap(foo, bar);

    cout << "foo contains:";
    for (int i : foo) cout << ' ' << i;
    cout << '\n';
    return 0;
}
```

std::vector / std::deque 強化版陣列

```
#include<vector>
```

- 由**尾部**插入或是刪除資料, 可以隨機存取裡面的元素
- 內部陣列大小會自動增長(可能使用倍增法)
- 移除元素時**不**減少記憶體花費

```
#include<deque>
```

- 由**頭尾**插入或是刪除資料, 可以隨機存取裡面的元素
- 內部陣列大小會自動增長(可能使用倍增法)
- 移除元素時**會**減少記憶體花費

元素存取

	vector/deque
尾巴	back()
頭部	front()
隨機存取	[index]

數量資訊

	vector/deque
裡面是不是空的	empty()
裡面有多少東西	size()

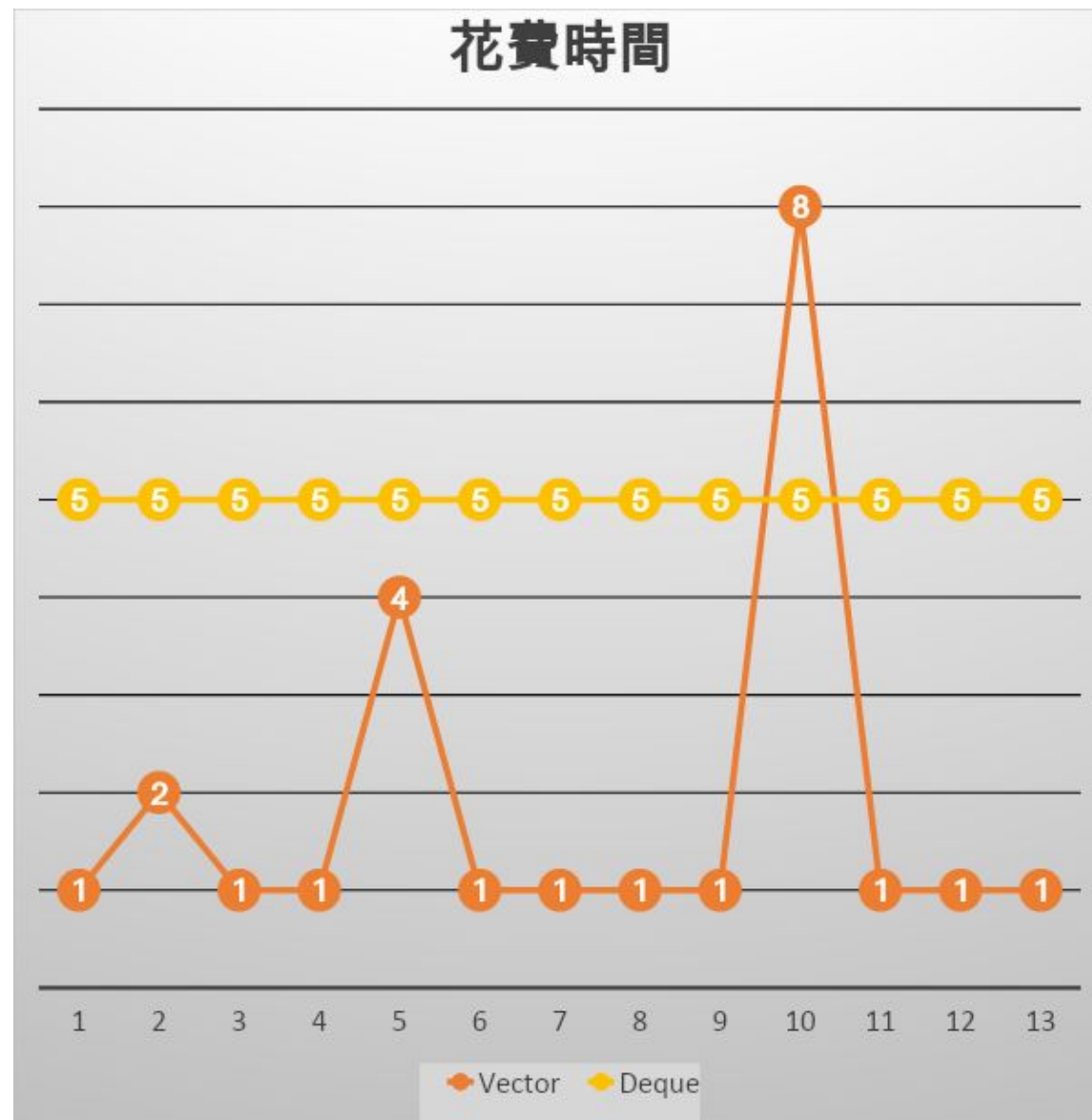
加入元素

- 在C++11以後盡可能使用emplace系列

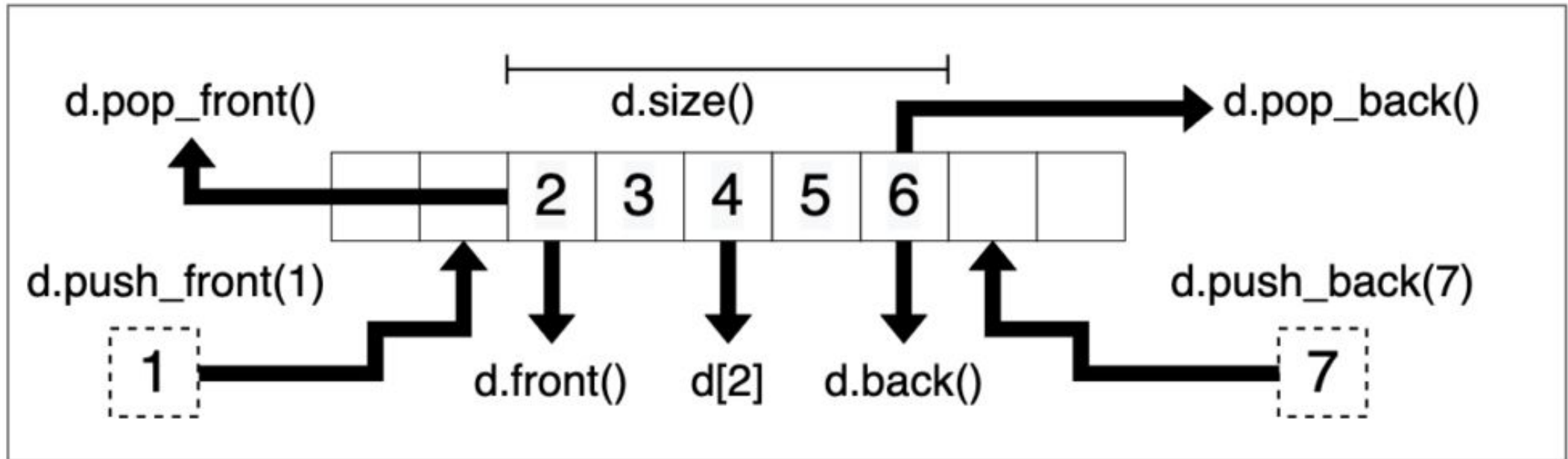
	Vector	Deque
在尾部加入	push_back(d)	push_back(d)
在尾部加入 (inplace)	emplace_back(d...)	emplace_back(d...)
在尾部刪除	pop_back()	pop_back()
在頭部加入		push_front()
在頭部加入 (inplace)		emplace_front(d...)
在頭部刪除	均攤 $O(1)$	pop_front() ⁽¹⁾

均攤複雜度

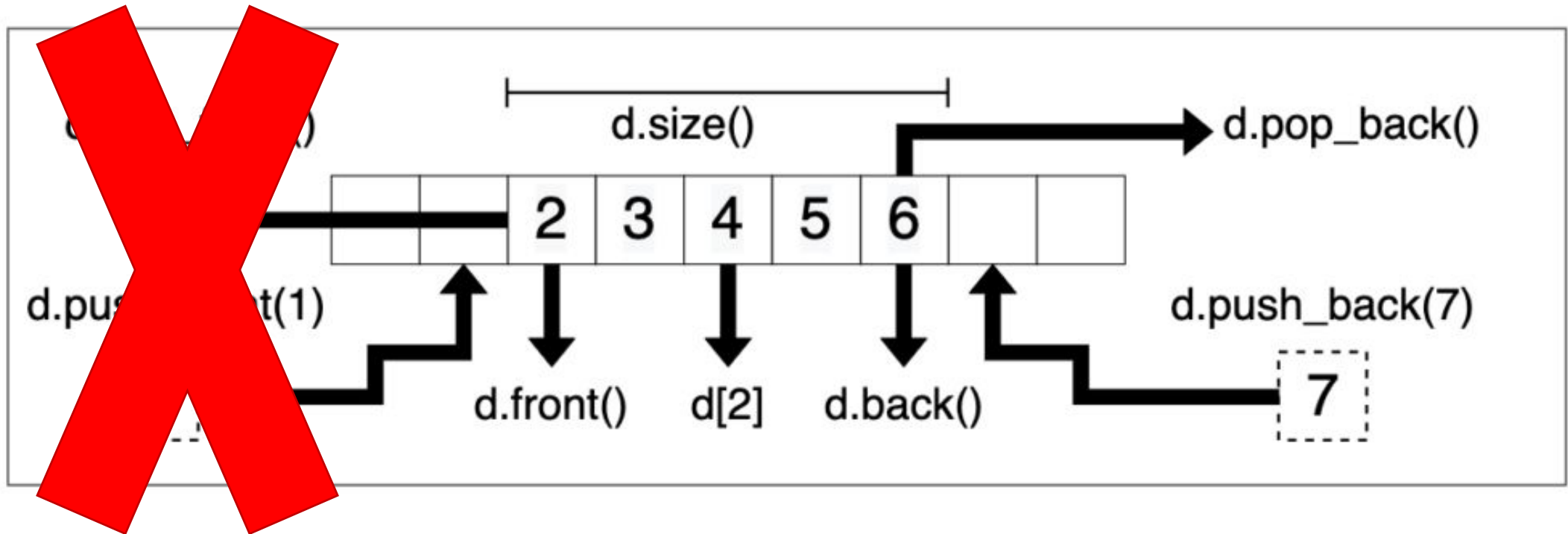
- 指經由多次操作後，整體的複雜度
 - Vector 一次操作：最快 $O(1)$ ，最慢 $O(n)$ ，但是平均來說是 $O(1)$
 - Deque 一次操作：都是 $O(1)$ ，但是有點慢的那種



deque



vector



Vector 使用範例

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

int main() {
    vector<int> V1{1, 2, 3}; // [1,2,3]
    vector<int> V2(5, 0);    // [0,0,0,0,0]
    vector<int> V3;          // []

    V1.pop_back();
    for (size_t i = 0; i < V1.size(); ++i)
        V3.emplace_back(V1[i]);
    for (size_t i = 0; i < V2.size(); ++i)
        V3.emplace_back(V2[i]);
    for (size_t i = 0; i < V3.size(); ++i)
        cout << V3[i] << '\n';
    return 0;
}
```

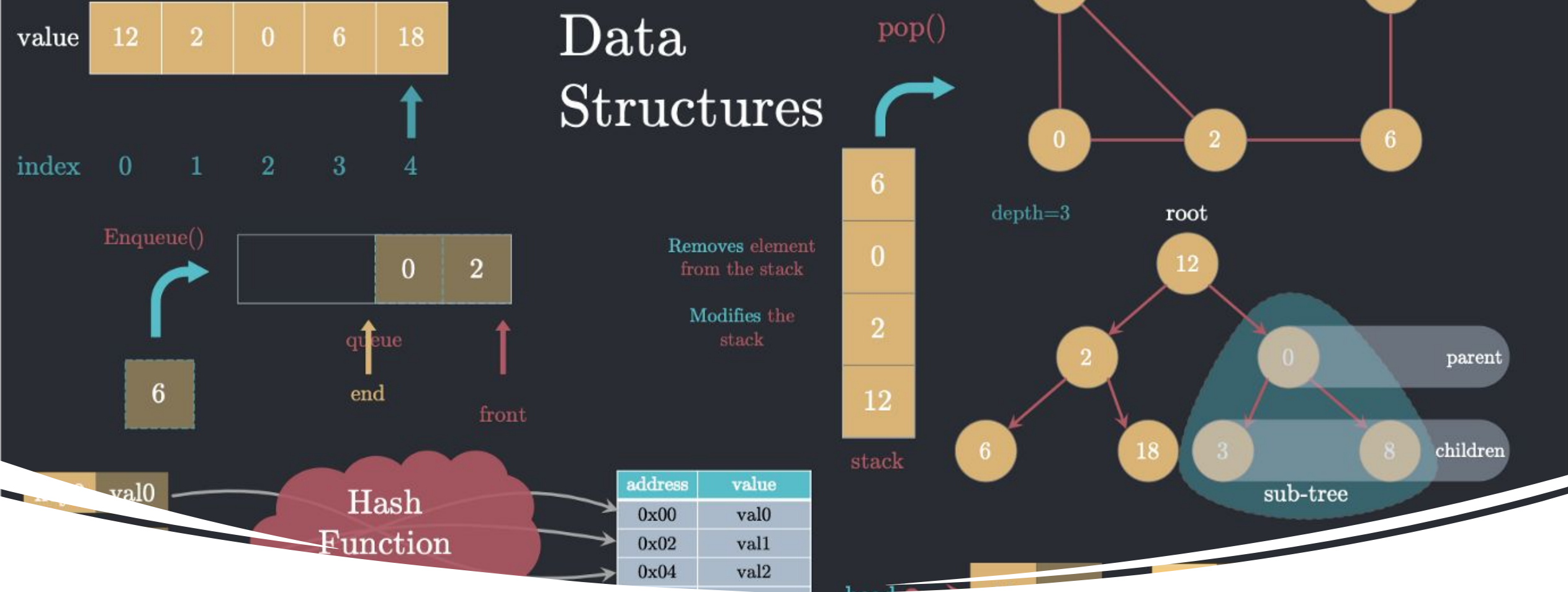
emplace vs push

```
vector<pair<int, int>> V;  
V.emplace_back(1, 2);  
V.emplace_back(3, 4);  
for (size_t i = 0; i < V.size(); ++i)  
    cout << V[i].first << ' ' << V[i].second << '\n';
```

```
vector<pair<int, int>> V;  
pair<int, int> P(1, 2);  
V.push_back(P);  
V.push_back(make_pair(3, 4));  
for (size_t i = 0; i < V.size(); ++i)  
    cout << V[i].first << ' ' << V[i].second << '\n';
```

resize 、 clear $O(n)$

```
vector<int> V;  
int n;  
cin >> n;  
V.resize(n);  
for (int i = 0; i < n; ++i)  
    cin >> V[i];  
V.clear();
```

Iterator 迭代器

- 資料結構總類繁多形狀各異
- 迭代器統一了各個資料結構**走訪**的介面

begin end $O(1)$

```
int main() {  
    vector<int> V{7, 1, 2, 2, 5, 3};  
    for (vector<int>::iterator It = V.begin(); It != V.end(); ++It)  
        cout << *It << ' '  
    cout << '\n';  
  
    for (auto It = V.begin(); It != V.end(); ++It)  
        cout << *It << ' '  
    cout << "\nsize = " << V.end() - V.begin() << endl;  
    return 0;  
}
```

```
jacky860226@DESKTOP-FCBV14M:/mnt,  
7 1 2 2 5 3  
7 1 2 2 5 3  
size = 6
```

Range-based for loop

```
for (auto &x : V) {  
    // ...  
}
```



```
{  
    for (auto It = V.begin(); It != V.end(); ++It) {  
        auto &x = *It;  
        // ...  
    }  
}
```

rbegin rend $O(1)$

```
int main() {  
    vector<int> V{7, 1, 2, 2, 5, 3};  
    for (vector<int>::reverse_iterator It = V.rbegin(); It != V.rend(); ++It)  
        cout << *It << ' ';  
    cout << '\n';  
  
    for (auto It = V.rbegin(); It != V.rend(); ++It)  
        cout << *It << ' ';  
    cout << '\n';  
    return 0;  
}
```

```
jacky860226@DESKTOP-FCBV14M:/mnt/  
3 5 2 2 1 7  
3 5 2 2 1 7
```


STL Algorithm functions

```
#include <algorithm>
```

std::reverse

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

int main() {
    vector<int> V{1, 2, 3, 4, 5};
    reverse(V.begin(), V.end());
    for (auto x : V)
        cout << x << ' ';
    cout << '\n';
    return 0;
}
```




jacky860226@DESK1
5 4 3 2 1

std::reverse

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

int main() {
    int V[5] = {1, 2, 3, 4, 5};
    reverse(V, V + 5);
    for (auto x : V)
        cout << x << ' ';
    cout << '\n';
    return 0;
}
```

A terminal window with a black background and a white border. It displays the output of the C++ program. The first line is the username 'jacky860226@DESK1' in green. The second line shows the reversed array '5 4 3 2 1' in white.

```
jacky860226@DESK1
5 4 3 2 1
```

std::reverse

```
template <class IterTy>
void reverse(IterTy first, IterTy last) {
    while ((first != last) && (first != --last)) {
        std::swap(*first, *last);
        ++first;
    }
}
```


std::fill

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

int main() {
    vector<int> V(5);
    fill(V.begin(), V.end(), 7122);
    for (auto x : V)
        cout << x << ' ';
    cout << '\n';
    return 0;
}
```

```
jacky860226@DESKTOP-FCBV14M: /mr
7122 7122 7122 7122 7122
```

std::fill

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

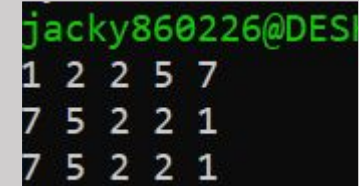
int main() {
    int V[5];
    fill(V, V + 5, 7122);
    for (auto x : V)
        cout << x << ' ';
    cout << '\n';
    return 0;
}
```

```
jacky860226@DESKTOP-FCBV14M: /mr
7122 7122 7122 7122 7122
```

std::sort

```
template <class T> void print(T &V) {  
    for (auto x : V)  
        cout << x << ' ';  
    cout << '\n';  
}
```

```
#include <algorithm>  
#include <iostream>  
#include <vector>  
using namespace std;  
  
int main() {  
    vector<int> V{7, 1, 2, 2, 5};  
    sort(V.begin(), V.end());  
    print(V);  
    sort(V.begin(), V.end(), [](auto a, auto b) { return a > b; });  
    print(V);  
    sort(V.rbegin(), V.rend());  
    print(V);  
    return 0;  
}
```



jacky860226@DESI
1 2 2 5 7
7 5 2 2 1
7 5 2 2 1

std::stable_sort

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

int main() {
    double V[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};
    int N = sizeof(V) / sizeof(double);
    stable_sort(V, V + N, [](int a, int b) { return a < b; });
    print(V);
    return 0;
}
```

```
jacky860226@DESKTOP-FCBV14M:/mnt/d/users/
1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67
```

std::unique

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

int main() {
    vector<int> V{7, 1, 2, 2, 5};
    sort(V.begin(), V.end());
    unique(V.begin(), V.end());
    print(V);
    return 0;
}
```

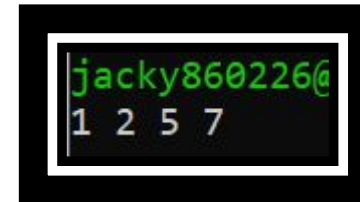


```
jacky860226@DESK
1 2 5 7 7
```

std::unique

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

int main() {
    vector<int> V{7, 1, 2, 2, 5};
    sort(V.begin(), V.end());
    auto It = unique(V.begin(), V.end());
    V.erase(It, V.end());
    print(V);
    return 0;
}
```



std::min / std::max

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

int main() {
    cout << min(1, 2) << endl;
    cout << min({7, 1, 2, 2}) << endl;
    cout << max(1, 2) << endl;
    cout << max({7, 1, 2, 2}) << endl;
    return 0;
}
```

std::min / std::max 形態要一樣

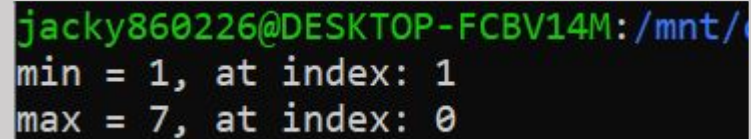
```
#include <algorithm>
#include <iostream>
using namespace std;

int main() {
    int a = 1;
    long long b = 2;
    cout << max(a, b); // error
    return 0;
}
```


std::min_element / std::max_element

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

int main() {
    vector<int> V{7, 1, 2, 2};
    auto It1 = min_element(V.begin(), V.end());
    cout << "min = " << *It1 << ", at index: " << It1 - V.begin() << endl;
    auto It2 = max_element(V.begin(), V.end());
    cout << "max = " << *It2 << ", at index: " << It2 - V.begin() << endl;
    return 0;
}
```



A terminal window showing the output of the C++ program. The prompt is 'jacky860226@DESKTOP-FCBV14M: /mnt/'. The output consists of two lines: 'min = 1, at index: 1' and 'max = 7, at index: 0'.

```
jacky860226@DESKTOP-FCBV14M: /mnt/
min = 1, at index: 1
max = 7, at index: 0
```

std::next_permutation

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

int main() {
    vector<int> V{7, 1, 2, 2};
    sort(V.begin(), V.end());
    do {
        print(V);
    } while (next_permutation(V.begin(), V.end()));
    return 0;
}
```

```
jacky860226@DESKTOP
1 2 2 7
1 2 7 2
1 7 2 2
2 1 2 7
2 1 7 2
2 2 1 7
2 2 7 1
2 7 1 2
2 7 2 1
7 1 2 2
7 2 1 2
7 2 2 1
```

std::prev_permutation

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

int main() {
    vector<int> V{7, 1, 2, 2};
    sort(V.rbegin(), V.rend());
    do {
        print(V);
    } while (prev_permutation(V.begin(), V.end()));
    return 0;
}
```



```
jacky860226
7 2 2 1
7 2 1 2
7 1 2 2
2 7 2 1
2 7 1 2
2 2 7 1
2 2 1 7
2 1 7 2
2 1 2 7
1 7 2 2
1 2 7 2
1 2 2 7
!
```