
Introduction to Algorithms

Science Honors Program (SHP)

Session 1

Christian Lim
Saturday, February 17, 2024

Christian Yongwhan Lim



Education



Part-time Jobs



Full-time Job



Workshops



Coach/Judge



<https://www.yongwhan.io>

Christian Yongwhan Lim



- Currently:
 - **Adjunct**, Columbia CS;
 - **CEO** (Co-Founder), Stealth Mode Startup;
 - **Co-Founder**, Christian and Grace Consulting;
 - **Head Coach**, Columbia ICPC;
 - **Internship Manager**, ICPC Foundation;
 - **Leadership Team**, ICPC North America (NA);
 - **Trainer**, ICPC NA Programming Camp;
 - **Judge**, ICPC NA Qualifiers and Regionals;



<https://www.yongwhan.io>

Who are you?

- **School?** (State?)
- **Year?**
- **Programming Experience?**
- **USACO level**, if applicable?
- **Hobby?**

Slide deck in github

- You may find the slide deck from:
 - [https://github.com/yongwhan/yongwhan.github.io/blob/master/columbia/shp/Spring%202024%20Introduction%20to%20Algorithms %20Session%201.pdf](https://github.com/yongwhan/yongwhan.github.io/blob/master/columbia/shp/Spring%202024%20Introduction%20to%20Algorithms%20Session%201.pdf)
- You may get to the link by:
 - <https://github.com/yongwhan/>
 - => yongwhan.github.io
 - => columbia
 - => shp
 - => session 1 slide

Spring 2024 Overview - February

| | |
|--------------------------|--|
| February 17, 2024 | <ul style="list-style-type: none">• Logistics and Introduction• Complexity Analysis• Sorting Algorithm (bubble sort)• Practice Strategies• Primitive (Data) Types• How to program in C++? |
| February 24, 2024 | <ul style="list-style-type: none">• Built-in Data Structures (vector; stack; queue; priority_queue; set; map)• Custom Data Structures (disjoint set union; Fenwick/Segment tree; ordered set) |

Spring 2024 Overview - March

| | |
|-----------------------|--|
| March 2, 2024 | <ul style="list-style-type: none">• Complete Search• Divide and Conquer (merge sort, quicksort, etc.) |
| March 9, 2024 | <ul style="list-style-type: none">• Greedy• Dynamic Programming |
| March 16, 2024 | <ul style="list-style-type: none">• NO CLASSES (SPRING BREAK) |
| March 23, 2024 | <ul style="list-style-type: none">• Graphs: Shortest Paths and Minimum Spanning Trees• Graphs: Lowest Common Ancestor and Flows |
| March 30, 2024 | <ul style="list-style-type: none">• NO CLASSES (EASTER WEEKEND) |

Spring 2024 Overview - April

| | |
|-----------------------|--|
| April 6, 2024 | <ul style="list-style-type: none">● Ad Hoc● Combinatorics |
| April 13, 2024 | <ul style="list-style-type: none">● Number Theory● Games |
| April 20, 2024 | <ul style="list-style-type: none">● Strings: Fundamentals● Strings: Matchings |
| April 27, 2024 | <ul style="list-style-type: none">● Geometry: Fundamentals and Convex Hull● Advanced Topics |

Spring 2024 Overview - May

May 4, 2024

- Advanced Topics
- Next Steps

Complexity Analysis

- **Space Complexity**

- "the total amount of memory space used by an algorithm"
- typically, it includes the space for inputs too!

- **Time Complexity**

- "the total amount of time it takes to run an algorithm"

Analysis Framework: Big O, Big Omega, and Big Theta

- **O**: We write $f(x) = O(g(x))$ and read it "f(x) is big O of g(x)" if
there exists $M > 0$ and x_0 such that $|f(x)| \leq Mg(x)$ for all $x > x_0$
- **Ω** : We write $f(x) = \Omega(g(x))$ and read it "f(x) is big Omega of g(x)" if
there exists $M > 0$ and x_0 such that $|f(x)| \geq Mg(x)$ for all $x > x_0$
- **Θ** : We write $f(x) = \Theta(g(x))$ and read it "f(x) is big Theta of g(x)" if
 $f(x)$ is both $O(g(x))$ and $\Omega(g(x))$

Example 1: Big O

- is n^2 $O(n)$?
- is n^2 $O(n^3)$?
- is $n^2 + 1,000,000,000$ $O(n^3)$?
- is 2^n $O(n!)$?
- is $\log(n)$ $O(n)$?
- is $7n^2$ $O(n^2)$?

Example 2: Big Omega

- is n^2 $\Omega(n)$?
- is n^2 $\Omega(n^3)$?
- is $n^2 + 1,000,000,000$ $\Omega(n^3)$?
- is 2^n $\Omega(n!)$?
- is $\log(n)$ $\Omega(n)$?
- is $7n^2$ $\Omega(n^2)$?

Example 3: Big Theta

- is n^2 $\Theta(n)$?
- is n^2 $\Theta(n^3)$?
- is $n^2 + 1,000,000,000$ $\Theta(n^3)$?
- is 2^n $\Theta(n!)$?
- is $\log(n)$ $\Theta(n)$?
- is $7n^2$ $\Theta(n^2)$?

Example 1

```
int ret=0;  
for (int i=0; i<n; i++)  
    ret++;
```

- Space Complexity: ?
- Time Complexity: ?

Example 1

```
int ret=0;  
for (int i=0; i<n; i++)  
    ret++;
```

- Space Complexity: $\theta(1)$
- Time Complexity: $\theta(n)$

Example 2

```
vector<int> a(n);  
for (int i=0; i<n; i++)  
    cin>>a[i];
```

```
int ret=0;  
for (int i=0; i<n; i++)  
    ret+=a[i];
```

- Space Complexity: ?
- Time Complexity: ?

Example 2

```
vector<int> a(n);  
for (int i=0; i<n; i++)  
    cin>>a[i];
```

```
int ret=0;  
for (int i=0; i<n; i++)  
    ret+=a[i];
```

- Space Complexity: $\theta(n)$
- Time Complexity: $\theta(n)$

Recurrence Relation

- **Fibonacci number:**
 - $f[0]=1;$
 - $f[1]=1;$
 - $f[n]=f[n-1]+f[n-2]$ for any $n \geq 2$.
- Other "named" linear recurrence can be found [here](#).
 - Lucas number, Padovan sequence, Pell number, Pell-Lucas number, Perrin sequence, ...
 - No need to memorize these, but knowing these exist is good enough!

- Space: $\theta(n)$
- Time: $\theta(n)$

Recurrence Relation: Iterative Implementation

- Writing a code for finding n^{th} Fibonacci number in iterative way is:

```
vector<int> f={1,1};  
for (int i=2; i<n; i++)  
    f.push_back(f[i-1]+f[i-2]);
```

- Space: $\theta(2^n)$
- Time: $\theta(2^n)$

Recurrence Relation: Recursive Implementation

- Writing it recursively is:

```
int fib(int n) {  
    if(n==0) return 1;  
    if(n==1) return 1;  
    return fib(n-1)+fib(n-2);  
}
```

- We will learn how to write it more efficiently using **memoization** later!

Sorting

- Given an array of integers (or any other data types that can be pairwise compared in terms of '<'), **sorting** would reorder the elements in the array from the **smallest** to the **largest** (or the largest to the smallest) using a comparing function ("comparer").
- Typically, in C++, you can use `sort(v.begin(), v.end())` to sort a container v (e.g., vector).

Sorting

- Given an array of integers (or any other data types that can be pairwise compared in terms of '<'), **sorting** would reorder the elements in the array from the **smallest** to the **largest** (or the largest to the smallest) using a comparing function ("comparer"). So, after sorting,

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 3 | 2 | 7 | 1 | 4 | 6 |
|---|---|---|---|---|---|---|

would look like:

Sorting

- Given an array of integers (or any other data types that can be pairwise compared in terms of '<'), **sorting** would reorder the elements in the array from the **smallest** to the **largest** (or the largest to the smallest) using a comparing function ("comparer"). So,

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 3 | 2 | 7 | 1 | 4 | 6 |
|---|---|---|---|---|---|---|

would look like:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|

Sorting

- There are many sorting algorithms. Some popular ones are:
 - **Bubble Sort;**
 - Insertion Sort;
 - Selection Sort;
 - Quick Sort;
 - Merge Sort;
- Today, we will cover:
 - Bubble Sort;

Bubble Sort

- Repeatedly swaps the adjacent elements if they are in the wrong order.

Bubble Sort

- Repeatedly swaps the adjacent elements if they are in the wrong order.
- Specifically, in bubble sort:
 - elements are scanned from left to right,
 - each element is compared to its adjacent element and the higher one is placed at right side by swapping, as necessary.

Bubble Sort: Step 1: the largest element

| | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|
| 5 | 3 | 2 | 7 | 1 | 4 | 6 |
| 3 | 5 | 2 | 7 | 1 | 4 | 6 |
| 3 | 2 | 5 | 7 | 1 | 4 | 6 |
| 3 | 2 | 5 | 7 | 1 | 4 | 6 |
| 3 | 2 | 5 | 1 | 7 | 4 | 6 |
| 3 | 2 | 5 | 1 | 4 | 7 | 6 |
| 3 | 2 | 5 | 1 | 4 | 6 | 7 |

Bubble Sort: Step 2: the 2nd largest element

| | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|
| 3 | 2 | 5 | 1 | 4 | 6 | 7 |
| 2 | 3 | 5 | 1 | 4 | 6 | 7 |
| 2 | 3 | 5 | 1 | 4 | 6 | 7 |
| 2 | 3 | 1 | 5 | 4 | 6 | 7 |
| 2 | 3 | 1 | 4 | 5 | 6 | 7 |
| 2 | 3 | 1 | 4 | 5 | 6 | 7 |

Bubble Sort: the process will continue until possible!

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|

- Time: $O(n^2)$
- Space: $O(n)$

Bubble Sort: Code

```
void bubbleSort(vector<int> &a, int n) {  
    bool swapped;  
    for (int i=0; i<n-1; i++) {  
        swapped = false;  
        for (j=0; j<n-i-1; j++)  
            if (a[j] > a[j+1])  
                swap(arr[j], arr[j+1]), swapped = true;  
        if (!swapped) break;  
    }  
}
```

Remarks

- **Sorting** is guaranteed to be **$n \log n$** for **merge sort**.
- In practice, **quicksort** is faster ($n \log n$ still on average) but the worst case time complexity is quadratic.
 - Even when the pivot selection is randomized, we can get extremely unlucky and happen to hit the bad pivot each time.
- We will look at merge sort and quicksort closely in the future!

An aerial photograph of a breaking wave. The top of the image shows the dark blue, textured surface of the water. Below this, a thick, horizontal band of white, frothy foam stretches across the frame, representing the crest of the wave. The bottom half of the image shows the dark blue water again, but with a more complex, swirling pattern of white foam and dark patches, indicating the turbulent water beneath the surface.

BREAK

CodeForces Zealots Problem Set

- Please join the following group:
<https://codeforces.com/group/hosRkEuluH>
- Doing questions in Zealots problem set is **COMPLETELY** optional!

How to use the Zealots Problem Set?

- Those who are just starting should focus on the **first half** of problems in Zealots Problem Set. Your main focus should be gaining some experiences with an explicit goal to enjoy the process of solving new problems and potentially making it to **USACO Platinum!**
- Those who are more serious should focus on the **second half** of problems in Zealots Problem Set. Your goal would be making into **USACO Programming Camp** and/or **International Olympiad in Informatics!**

Practice Strategies

- If your goal is to get to a rating of **X**, you should practice on problems that are **X + 300** typically, with a spread of 100. So, picking problems within the range of:

$\{X + 200, X + 300, X + 400\}$

would be sensible!

- So, if you want to target becoming a **red**, which has a lower-bound of 2400, you should aim to solving $\{2600, 2700, 2800\}$.
- **(Eventual) Target:** You should focus on solving it for 30 minutes or less!

Practice Strategies

- You should focus on solving each problem for **30 minutes or less**; if you cannot solve any problem with this range, you should consider solving a problem with a lower rating.
- You should aim to solve **10 ~ 15 problems** each day within this range to expect a rank up within a quarter (3 months).

Practice Strategies

- If you **cannot** solve a problem, here is a sample recipe you can follow:
 - Look at editorial for **hints**, and try to solve the problem.
 - Look at editorial for **full solutions**, and try to solve the problem.
 - Look at **accepted solutions**, and try to solve the problem.
 - Make sure you look back **after two weeks** and see if you can solve it.

Live Contest Strategies

- [A Terse Guide to Live Contests](#)

C++ Tips and Tricks: best to learn those through practice!

- [C++ Tricks](#) (HosseinYousefi)
- [C++ tips and tricks](#) (Golovanov399)
- [Some Tips for Coding in C++ in Competitive Programming](#) (Nea1)
- Use `"#include <bits/stdc++.h>"` header to include **almost everything**.

Standard Input/Output (stdio)

- [Yet again on C++ input/output](#) (andreyv)
- **scanf/printf vs cin/cout**
 - Often, use `"ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);"`

Primitive Type

- `int, long long, double, long double, char, float, ...`

int

- "int" is short for "integer"
- Used to store whole numbers
- Internally, they are stored using binary numbers: ones and zeros
- Number of bytes used for an int varies by system
- 1 byte = 8 bits

Examples

- 6
- 13
- 1993
- -777
- 10
- 2
- 2021

float

- "float" is short for "floating-point"
- Floats can store numbers with a fractional part (real numbers)

Examples

- -777.77
- 6.131993
- 9.301989
- 10.22021
- -123.765
- 0.0

char

- "char" is short for "character"
 - Used to store individual letters, digits, symbols, etc
 - These are the keys you have on your keyboard
-
- Typically stored using a single byte
 - But, with the rise of Unicode, many systems use two bytes now
 - In C, chars are delimited by apostrophes (single quotes)

Examples

- 'A'
- '7'
- ' '
- '\$'
- '&'
- '^'
- '-'

Numeric Limits (Machine Dependent)

- SIGNED INTEGERS (short, int, long long)
 - short minimum: $-32768 = -2^{15}$
 - short maximum: $32767 = 2^{15} - 1$
 - int minimum: $-2147483648 = -2^{31}$
 - int maximum: $2147483647 = 2^{31} - 1$
 - long minimum: $-9223372036854775808 = -2^{63}$
 - long maximum: $9223372036854775807 = 2^{63} - 1$
- UNSIGNED INTEGERS (unsigned short, unsigned int, unsigned long long)
 - minimum is all zero.
 - unsigned short maximum: $65535 = 2^{16} - 1$
 - unsigned int maximum: $4294967295 = 2^{32} - 1$
 - unsigned long maximum: $18446744073709551615 = 2^{64} - 1$

Numeric Limits (Machine Dependent)

- FLOAT PRECISION:
 - float precision digits: 6
 - float maximum exponent: 38
 - float maximum: 3.402823e+038
 - double precision digits: 15
 - double maximum exponent: 308
 - double maximum: 1.797693e+308
 - long double precision: 18
 - long double maximum exponent: 4932
 - long double maximum: 1.189731e+4932

Declaring Variables

- All variables **MUST** be declared.
- Examples:
 - `int day;`
 - `int cents;`
 - `float x;`
 - `float y1, y2;`
 - `double degrees;`
 - `double a,b,c;`

So, how to program in C++? First, setup your compiler!

- <https://www.onlinegdb.com/>
- If you are serious, you may want to have a local setup.
 - For Windows, I recommend using Visual Studio
(<https://code.visualstudio.com/docs/cpp/config-mingw>)
 - For Mac, you should already have a built-in compiler. If you like gcc, you may try: "brew install gcc" in terminal
(<https://osxdaily.com/2023/05/02/how-install-gcc-mac>)

So, how to program in C++? First, setup your compiler!

- You may also want to set up "bits/stdc++.h"
 - If you run into issues, please take a look at:
<https://apple.stackexchange.com/questions/148401/file-not-found-error-while-including-bits-stdc-h>

So, how to program in C++? Second, try "Hello World!"

- Let's try the following **hello world** program!

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

int main() {
    cout<<"Hello World!"<<endl;
    return 0;
}
```

So, how to program in C++? Third, learn template code

- Typically, you'd like to have the following template as `template.cpp`:

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

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    return 0;
}
```

(Optional) Try CodeForces Zealots Problem Set

- Next week, we will cover **Data Structures!**
 - **Built-in Data Structures:** vector; stack; queue; priority_queue; set; map;
 - **Custom Data Structures:** Disjoint Set Union; Fenwick/Segment Tree; Ordered Set;

Next Week!

- Next week, we will cover **Data Structures!**
 - **Built-in Data Structures:** vector; stack; queue; priority_queue; set; map;
 - **Custom Data Structures:** Disjoint Set Union; Fenwick/Segment Tree; Ordered Set;

Slide Deck

- You may find the slide decks from:
 - <https://github.com/yongwhan/yongwhan.github.io/blob/master/columbia/shp>

THANK YOU

