CSC212

Data Structures

- Section FG

Computer Science
City College of New York

Lecture 1: Introduction

Instructor: Feng HU

fhu@gradcenter.cuny.edu

Department of Computer Science

City College of New York

Outline of this lecture

- Course Objectives and Schedule
  - WHAT (Topics)
  - WHY (Importance)
  - WHERE (Goals)
  - HOW (Information and Schedule)

- The Phase of Software Development
  - Basic design strategy
  - Pre-conditions and post-conditions
  - Running time analysis

Topics (WHAT)

- Data Structures
  - specification, design, implementation and use of
    - basic data types (arrays, lists, queues, stacks, trees…)
- OOP and C++
  - C++ classes, container classes , Big Three
- Standard Template Library (STL)
  - templates, iterators
  - ADTs in our DS course cut-down version of STL
- Recursion, Searching and Sorting Algorithms
  - important techniques in many applications

Importance (WHY)

- Data Structures (how to organize data) and Algorithms (how to manipulate data) are the cores of today's computer programming

- The behavior of Abstract Data Types (ADTs) in our Date Structures course is a cut-down version of Standard Template Library (STL) in C++

- Lay a foundation for other aspects of "real programming" – OOP, Recursion, Sorting, Searching

Goals (WHERE)

understand the data types inside out

- Implement these data structures as classes in C++
- Determine which structures are appropriate in various situations
- Confidently learn new structures beyond what are presented in this class
- also learn part of the OOP and software development methodology

Course Information (HOW)

- Objectives
  - Data Structures, with C++ and Software Engineering
- Textbook and References
  - Texbook: **Data Structures and Other Objects Using C++ , Fourth Edition by Michael Main and Walter Savitch**
  - Reference: *C++ How to Program* by Dietel & Dietel, 3rd Ed., Prentice Hall 2001
- Prerequisites
  - CSc103 C++ (Intro to Computing for CS and CpE)
  - CSc 104 (Discrete Math Structure I)
- Assignments and Grading
  - **6-7 programming assignments** roughly every 2 weeks (30%)
  - **3 in-class writing exams** (60%), several in-class quizzes (10%)
- Computing Facilities
  - PCs: Microsoft Visual C++ ; Unix / Linux : gc++?; MinGW
  - also publicly accessible at Computer Science labs

Tentative Schedule (HOW)

(28 classes = 23 lectures + 3 reviews + 3 exams, 6-7 assignments)

- Lecture 1. The Phase of Software Development (Ch 1)
- Lectures 2-3. ADT and C++ Classes (Ch 2)
- Lecture 4-5. Container Classes (Ch 3)
- Lectures 6-8. Pointers and Dynamic Arrays (Ch 4)
- Reviews and the 1st exam (Ch. 1-4, before Columbus Day)
- Lectures 9-10. Linked Lists (Ch. 5)
- Lectures 11. 11a. Template and STL (Ch 6)
- Lecture 12. Stacks (Ch 7) and Queues (Ch 8)
- Lectures 13-14. Recursion (Ch 9)
- Reviews and the 2nd exam (Ch. 5-9, before Thanksgiving)
- Lectures 15-18. Trees (Ch 10, Ch 11)
- Lectures 19-20. Searching and Hashing (Ch 12)
- Lectures 21- 22. Sorting (Ch 13)
- Lecture 23. Graphs (Ch 15)
- Reviews and the 3rd exam (mainly Ch. 10-13, Dec 14)

Course Web Page

You can find all the information at

http://ccvcl.org/~fhu/CSc212FG-Fall2016.html

-Come back frequently for the updating of lecture schedule, programming assignments and exam schedule

- Reading assignments & programming assignments

Outline

- Course Objectives and Schedule
  - Information
  - Topics
  - Schedule
- The Phase of Software Development
  - Basic design strategy
  - Pre-conditions and post-conditions
  - Running time analysis

Phase of Software Development

- Basic Design Strategy – four steps (Reading: Ch.1)
  - Specify the problem - Input/Output (I/O)
  - Design data structures and algorithms (**pseudo code**)
  - Implement in a language such as C++
  - Test and debug the program (Reading Ch 1.3)
- Design Technique
  - Decomposing the problem
- Two Important Issues (along with design and Implement)
  - **Pre-Conditions and Post-Conditions**
  - **Running Time Analysis**

Preconditions and Postconditions

- An important topic: **preconditions** and **postconditions**.
- They are a method of specifying what a function accomplishes.

Preconditions and Postconditions

Frequently a programmer must communicate precisely **what** a function accomplishes, without any indication of **how** the function does its work.

Can you think of a situation where this would occur ?

Example

- You are the head of a programming team and you want one of your programmers to write a function for part of a project.

What are Preconditions and Postconditions?

- One way to specify such requirements is with a pair of statements about the function.
- The **precondition** statement indicates what must be true before the function is called.
- The **postcondition** statement indicates what will be true when the function finishes its work.

Example

```
void write_sqrt( double x)

//   Precondition:  x  >=  0.
//   Postcondition:  The square root of x has
//   been written to the standard output.

   ...
```

Example

```
void write_sqrt( double x)

//   Precondition:  x  >=  0.
//   Postcondition:  The square root of x has
//   been written to the standard output.
```

- The precondition and postcondition appear as comments in your program.
- They are usually placed after the function’s parameter list.

Example

```
void write_sqrt( double x)

//   Precondition:  x  >=  0.
//   Postcondition:  The square root of x has
//   been written to the standard output.
```

- In this example, the precondition requires that

  x >= 0

  be true whenever the function is called.

Example

Which of these function calls meet the precondition ?

```
write_sqrt( -10 );
write_sqrt( 0 );
write_sqrt( 5.6 );
```

Example

Which of these function calls meet the precondition ?

```
write_sqrt( -10 );
write_sqrt( 0 );
write_sqrt( 5.6 );
```

The second and third calls are fine, since the argument is greater than or equal to zero.

Example

Which of these function calls meet the precondition ?

```
write_sqrt( -10 );
write_sqrt( 0 );
write_sqrt( 5.6 );
```

But the first call violates the precondition, since the argument is less than zero.

Example

```
void write_sqrt( double x)

//   Precondition:  x  >=  0.
//   Postcondition:  The square root of x has
//   been written to the standard output.
}
```

- The postcondition always indicates what work the function has accomplished. In this case, when the function returns the square root of **x** has been written.

Another Example

```
bool is_vowel( char letter )
//   Precondition:  letter is an uppercase or
//   lowercase letter (in the range 'A' ... 'Z' or 'a' ... 'z') .
//   Postcondition:  The value returned by the
//   function is true if letter is a vowel;
//   otherwise the value returned by the function is
//   false.

   ...
```

Another Example

What values will be returned by these function calls ?

```
is_vowel( 'A' );
is_vowel(' Z' );
is_vowel( '?' );
```

Another Example

What values will be returned by these function calls ?

```
is_vowel( 'A' );
is_vowel(' Z' );
is_vowel( '?' );
```

true

false

Nobody knows, because the precondition has been violated.

Consequence of Violation

Who are responsible for the crash ?

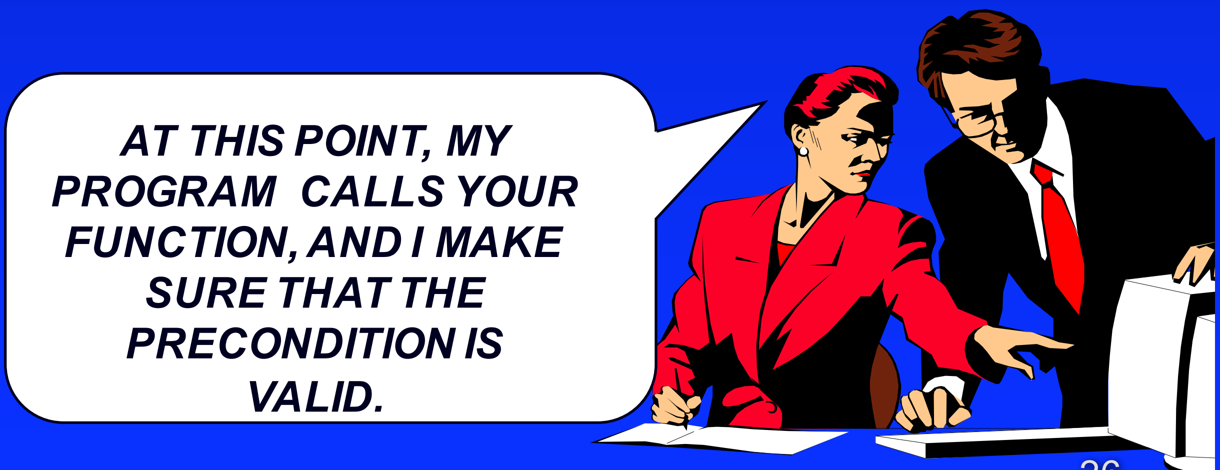
```
write_sqrt(-10.0);
is_vowel( '?' );
```

Violating the precondition might even crash the computer.

Bring up Notes!!!

Always make sure the precondition is valid . . .

- The programmer who calls the function is responsible for **ensuring that the precondition is valid** when the function is called.



. . . so the postcondition becomes true at the function's end.

- The programmer who writes the function counts on the precondition being valid, and **ensures that the postcondition becomes true** at the function's end.

A Quiz

The beginning of the fall semester at CUNY is in three days, and the website that controls everything at CUNY is wreaking havoc yet again. –Juan Monroy's Blog

Suppose that you call a function, and you neglect to make sure that the precondition is valid. Who is responsible if this causes a persist outage of the CUNYFirst ?*

* Primary issue seems to be the number of users accessing the system at the same time creating an overload situation. - Daniel Matos, CCNY Office of the Registrar

① You

② The programmer who wrote that CUNYFirst function

③ CUNY Chancellor

A Quiz

The beginning of the fall semester at CUNY is in three days, and the website that controls everything at CUNY is wreaking havoc yet again. –Juan Monroy's Blog

Suppose that you call a function, and you neglect to make sure that the precondition * *is valid. Who is responsible if this causes a persist outage of the CUNYFirst ?*

* Primary issue seems to be the number of users accessing the system at the same time creating an overload situation. - Daniel Matos, CCNY Office of the Registrar

① You

The programmer who calls a function is responsible for ensuring that the precondition is valid.

On the other hand, careful programmers also follow these rules:

- When you write a function, you should make every effort to detect when a precondition has been violated.
- If you detect that a precondition has been violated, then print an error message and halt the program.

On the other hand, careful programmers also follow these rules:

- When you write a function, you should make every effort to detect when a precondition has been violated.
- If you detect that a precondition has been violated, then print an error message and halt the program...
- ...rather than causing a chaos.

Example

```
void write_sqrt( double x)
//   Precondition:  x  >=  0.
//   Postcondition:  The square root of x has
//   been written to the standard output.
{
     assert(x >= 0);

   . . .
```

- The assert function (described in Section 1.1) is useful for detecting violations of a precondition.

Advantages of Using Pre- and Post-conditions

- Concisely describes the behavior of a function...
- ... without cluttering up your thinking with details of how the function works.
- At a later point, you may reimplement the function in a new way ...
- ... but programs (which only depend on the precondition/postcondition) will still work with no changes.

Break

Summary of pre- and post-conditions

Precondition

- The programmer who calls a function ensures that the precondition is valid.
- The programmer who writes a function can bank on the precondition being true when the function begins execution.

Postcondition

- The programmer who writes a function ensures that the postcondition is true when the function finishes executing.

Phase of Software Development

- Basic Design Strategy – four steps (Reading: Ch.1)
  - Specify Input/Output (I/O)
  - Design data structures and algorithms
  - Implement in a language such as C++
  - Test and debug the program (Reading Ch 1.3)
- Design Technique
  - Decomposing the problem
- Two Important Issues (along with design and Implement)
  - **Pre-Conditions and Post-Conditions**
  - **Running Time Analysis**

Running Time Analysis – Big O

- Time Analysis
  - Fast enough?
  - How much longer if input gets larger?
  - Which among several is the fastest?

Example : Stair Counting Problem

- How many steps ?

1789 (Birnbaum)

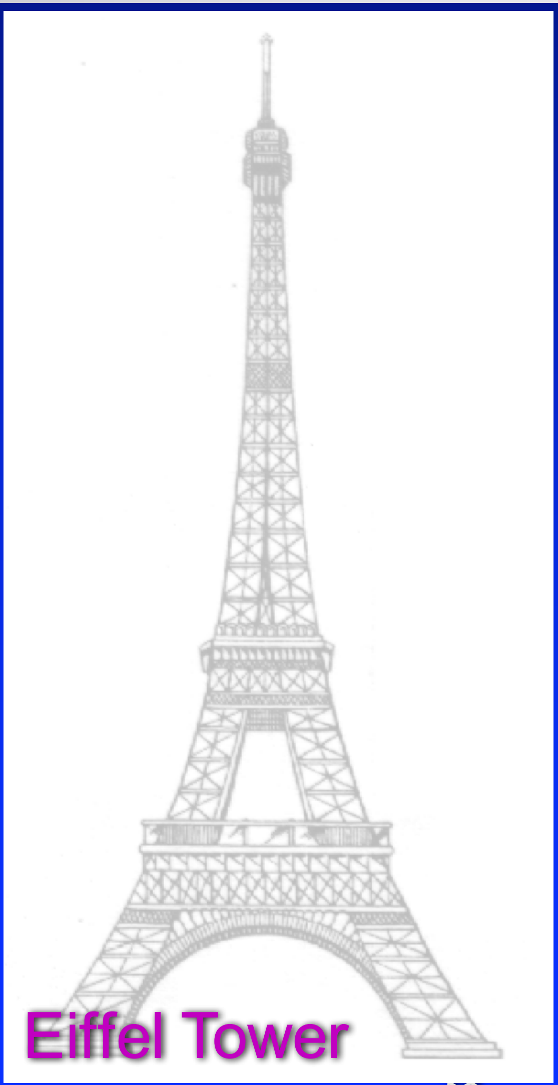
1671 (Joseph Harriss)

1652 (others)

1665 (Official Eiffel Tower Website)

-

- Find it out yourself !



Example : Stair Counting Problem

- Find it out yourself !
  - Method 1: Walk down and keep a tally

Each time a step down, make a mark

  - Method 2 : Walk down, but let Judy keep the tally

Down+1, hat, back, Judy make a mark

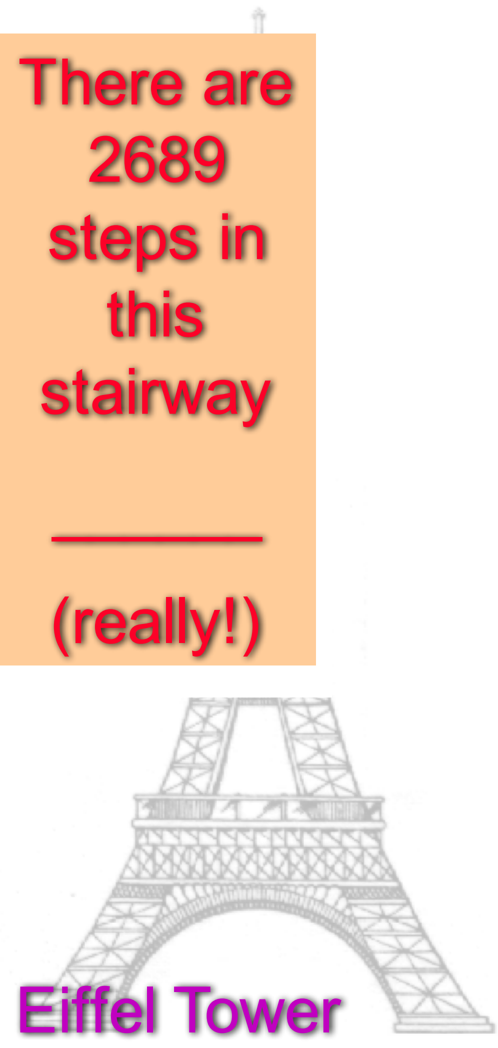
  - Method 3: Jervis to the rescue

One mark per digit

There are 2689 steps in this stairway

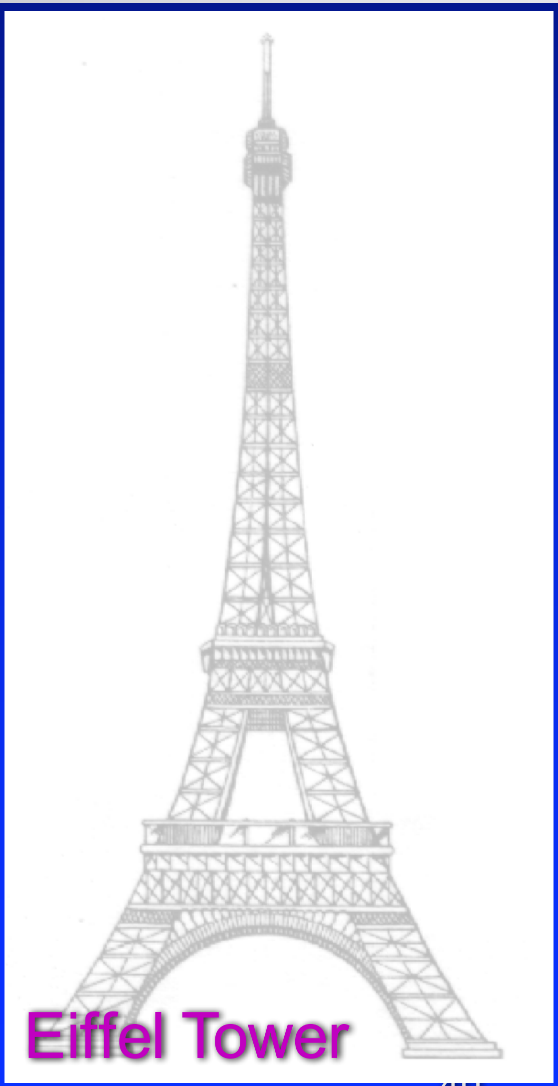
______

(really!)



Example : Stair Counting Problem

- How to measure the time?
  - Just measure the actual time
    - vary from person to person
    - depending on many factors
  - Count certain operations
    - each time walk up/down, 1 operation
    - each time mark a symbol, 1 operation



Example : Stair Counting Problem

- Find it out yourself !
  - Method 1: Walk down and keep a tally

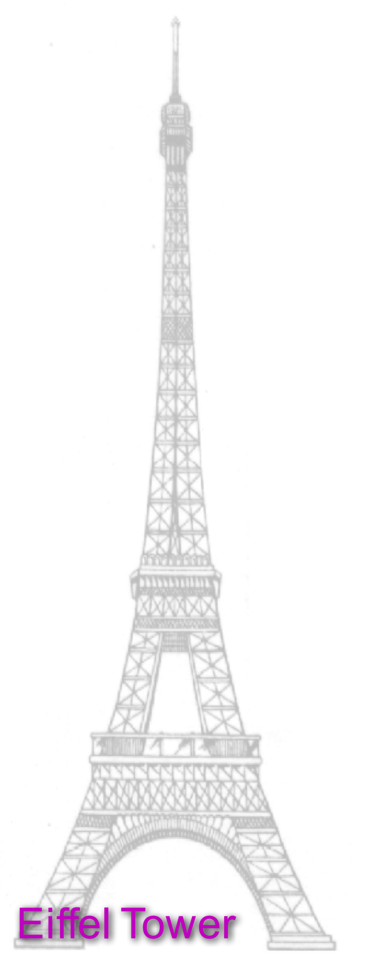
2689 (down) + 2689 (up) + 2689 (marks) = 8067

  - Method 2 : Walk down, let Judy keep tally

Down:	3,616,705 = 1+2+…+2689	
Up:	3,616,705 = 1+2+…+2689	7,236,099 !
Marks:	2,689 = 1+1+…+1	

  - Method 3: Jervis to the rescue

only 4 marks !


Eiffel Tower

Example : Stair Counting Problem

- Size of the Input : n
  - Method 1: Walk down and keep a tally

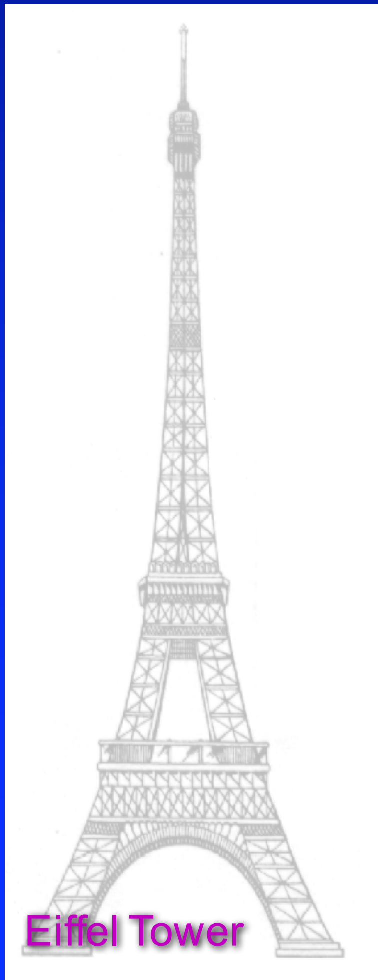
3n

  - Method 2 : Walk down, let Judy keep tally

$n+2(1+2+\ldots+n) = n+(n+1)n = n^2+2n$

    - Trick: Compute twice the amount
      - and then divided by two
  - Method 3: Jervis to the rescue

The number of digits in n = $[\log_{10} n]+1$



Example : Stair Counting Problem

- Big-O Notation – the order of the algorithm
    - Use the largest term in a formula
    - Ignore the multiplicative constant
  - Method 1: Linear time

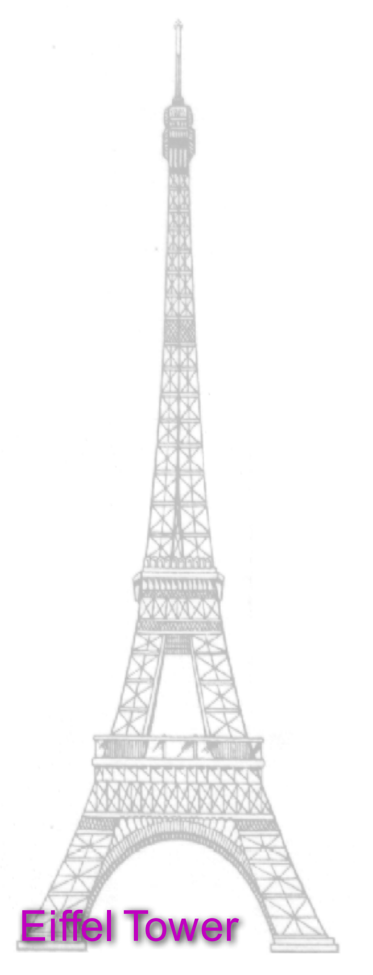
$3n \Rightarrow O(n)$

  - Method 2 : Quadratic time

$n^2+2n \Rightarrow O(n^2)$

  - Method 3: Logarithmic time

$[\log_{10} n]+1 \Rightarrow O(\log n)$



Eiffel Tower

A Quiz

Number of operations	Big-O notation
n^2+5n	$O(n^2)$
$100n+n^2$	$O(n^2)$
$(n+7)(n-2)$	$O(n^2)$
$n+100$	$O(n)$
number of digits in 2n	$O(\log n)$

Big-O Notation

- The order of an algorithm generally is more important than the speed of the processor

Input size: n	O(log n)	O (n)	O (n^2)
# of stairs: n	[$\log_{10}n$]+1	3n	n^2+2n
10	2	30	120
100	3	300	10,200
1000	4	3000	1,002,000

Time Analysis of C++ Functions

- Example- Quiz (5 minutes)
  - Printout all item in an integer array of size N

```
for (i=0;  i< N; i++ )
{
        val = a[i];
        cout << val;
}
```

2 C++ operations or more?

- Frequent linear pattern
  - A loop that does a fixed amount of operations N times requires O(N) time

Time Analysis of C++ Functions

- Another example
  - Printout char one by one in a string of length N

```
for (i=0;  i< strlen(str); i++ )
{
        c = str[i];
        cout << c;
}
```

O(N^2)!

- What is a single operation?
  - If the function calls do complex things, then count the operation carried out there
  - Put a function call outside the loop if you can!

Time Analysis of C++ Functions

- Another example
  - Printout char one by one in a string of length N

```
N = strlen(str);
for (i=0;  i<N; i++ )
{
        c = str[i];
        cout << c;
}
```

O(N)!

- What is a single operation?
  - If the function calls do complex things, then count the operation carried out there
  - Put a function call outside the loop if you can!

Time Analysis of C++ Functions

- Worst case, average case and best case
  - search a number x in an integer array a of size N

```
for (i=0;  (i< N) && (a[i] != x); i++ );

if (i < N) cout << "Number " << x << "is at location " << i << endl;
else cout << "Not Found!" << endl;
```

- Can you provide an exact number of operations?
  - Best case: 1+2+1
  - Worst case: 1+3N+1
  - Average case: 1+3N/2+1

Testing and Debugging

- Test: run a program and observe its behavior
  - input -> expected output?
  - how long ?
  - software engineering issues
- Choosing Test Data : two techniques
  - boundary values
  - fully exercising code (tool: profiler)
- Debugging… find the bug after an error is found
  - rule: never change if you are not sure what's the error
  - tool: debugger

Summary

- Often ask yourselves FOUR questions
  - WHAT, WHY, WHERE & HOW
    - Topics – DSs, C++, STL, basic algorithms
    - Data Structure experts
    - Schedule – 23 lectures, 6 assignments, 3 exams
    - A lot of credits (>10/100) for attending the class
    - Information – website
- Remember and apply two things (Ch 1)
  - Basic design strategy
  - Pre-conditions and post-conditions
  - Running time analysis
  - Testing and Debugging (reading 1.3)

Reminder ....

Lecture 2: ADT and C++ Classes

Reading Assignment before the next lecture:

Chapter 1

Chapter 2, Sections 2.1-2.3

Office Hours:

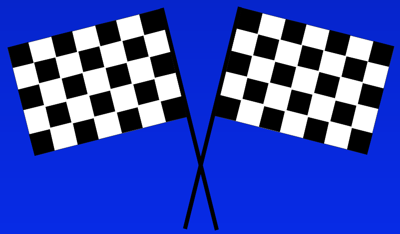
Mon/Wed 3:00 pm - 4:00 pm
(Location: NAC 8/210)

Update:

check website for details

Homework

- Send me an email (fhu@gradcenter.cuny.edu) listing your expectations/comments/suggestions of this course, as the first attendance.



THE END