3137 Data Structures and Algorithms in C++

Lecture 1

July 5 2006

Shlomo Hershkop

It summer Session!

- Welcome
- Ask yourself, is it better to spend the summer outside or inside on this stuff ?!
- Hope to be very informal
  - small class size….which can be a good thing and bad!
- I hope to convince you this is more fun than sitting on the beach ☺

Overview

- Today:
  - Basic overview of the course and objectives
  - background c++
  - background algorithms
  - first assignment (gasp!)
- Goal:
  - Thing are much easier if everyone knows why they are here, and what we are trying to accomplish.
  - Interactive course
  - We will learn about programming ideas while trying to have fun.

What is 3137?

- CS3137: Fourth course for CS majors.
- Prerequisites:
  - Intermediate knowledge in general Programming
  - Basic discrete mathematic skills
  - Program Structure:
    - Not enough to know how to write a program, need to know how to analyze which structures work best for a specific task

quiz!

- why is 3137 after 3157 ?
- did someone mess up their sort implementation ??

So what are we going to be doing?

- Learn basic algorithm analysis
- bunch of basic data structures
- bunch of advanced DS
- advanced Algorithm analysis
- applied to practical problems

Basics

- Instructor: Professor Shlomo Hershkop (shlomo@cs.columbia.edu)
- About Me/my Research
- Office hours:
  M/W 4-5
  AIM: Prof Hershkop
- Class website:
  - cs.columbia.edu/~sh553/teaching/su06-3137/
  - Check it regularly (at least twice a week).
    - See announcement sections for update info.
- Meet twice a week: 825 Mudd
  - Please come on time

Resources

- TA:
  - Weijen Lee
- Since we are a small class:
  - Please take advantage of the web board
    - How do I check what version of gcc is running?
    - What does Error ?@?@!?@ mean ?
  - Bad:
    - What is wrong with the following code:
    - void foo()
  - These kind of questions email privately to TA or Instructor
  - Use your best judgement

Requirements

- Interest to learn about Computer Science
- Learn to use cool DS
- Learn to make your own program work better

Textbook

- Textbook can be acquired online or at the Columbia Bookstore.
  - Else: borrow, threaten, or ‘acquire’ a book
- Required:
  - Mark Allen Weiss
    Data Structures and Algorithm Analysis in C++
    3rd edition
    ISBN: 032144146X
- Recommended:
  - Any C++ background book.

Reading

- I will be posting reading on the website and in class notes
- Please try to keep up with the reading
  - I will try to make up examples for class, but there are random stuff which the book covers which is good to see in print
    - Feel free to ask questions from anything you read/see/imagine in the book

Course Structure

- 6 Homeworks – 120 points
  - Will have about 1 weeks per homework
- Midterm – 30 Points
  - thinking about take home
- Final (90 points)
  - open book, in class
- Homework is important:
  - Firm believer in hands on learning
  - Start early
  - Come to office hours, and ask questions
    - We are here for YOU!

Class participation and Attendance

- Attendance and participation is expected
  - Very interactive lectures & Labs
  - Small class, means more help
  - Class anonymous feedback system
- If you have to miss class, I expect you to catch up.
  - It's a short semester, so bear with me
  - There will be class notes posted to the website
  - There will be many examples in class only, so make sure to get someone's notes.

Homework & Projects

- Written:
  - Will be collected at first class after HW deadline.
- Programming:
  - Online submission
  - Must be able to run on cunix system (this is important).
- Late policy:
  - You have late days that can be used during the semester.
  - I can only review the homework and approaches if everyone submits on time, so try to ask for help earlier rather than later

Cheating Policy

- Plagiarism and cheating:
  - I'm all against it. It is unacceptable.
- You're expected to do homeworks by yourself
  - This is a learning experience.
  - You will only cheat yourself.
  - My job is to help you learn, not catch you cheating, but….
- Automated tools to catch plagiarizers
  - http://www.cs.berkeley.edu/~aiken/moss.html
  - Moving stuff around, renaming, etc. doesn't help
- Results: instant zero on assignment, referral to academic committee
  - Columbia takes dishonesty very seriously
  - I'd much rather you come to me or the TAs for help

Feedback System

- Last minute of class will be set aside for feedback:
  - Please bring some sort of scrap paper to class to provide feedback.
  - Feel free to leave it anonymous.
  - Content: Questions, comments, ideas, random thoughts.
- I will address any relevant comments at the beginning of each class
- Summer is short, so provide feedback !
- Please feel free to show up to office hours or make an appointment at any time

Shopping List

- You need either a cunix or CS account
  - CS:
    - https://www.cs.columbia.edu/~crf/accounts
    - Try to log into the account asap
  - Cunix
    - log into cunix.cc.columbia.edu
- Check out the class page
- Make textbook plans
  - try keep up with the reading

- Any Questions ?

Survey 1

- Please introduce yourself
  - Programming background ?
  - what C++ environments you've worked with
  - Any cool technologies you would like to see covered ?

Definitions

- Algorithm:
  - Problem solving method to be used to solve a problem independent of particular computer or program
  - Central objects of study in computer science
- Heuristic
  - In CS it is an Algorithm which is not guaranteed to find a solution
  - we will be studying algorithms which guarantee a solution with some constraints

- most algorithms involve organizing data in a specific way and supporting a specific set of operations
- These are called Data Structures
  - will start with simple ones
  - study analysis techniques
  - combination of structures

solving a problem

- once you outline a problem to be solved by a computer
  - choice of language
  - choice of approach
  - for small problems exact solution might not make a big difference
  - for huge problems, sometimes a specific solution might take too long, and we are trying to get it solved faster

simple approach

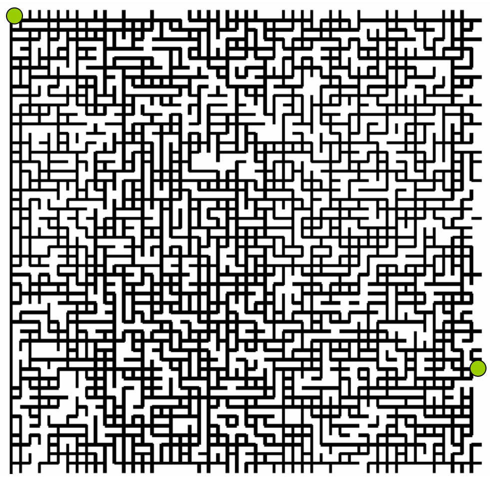
- Throw money and buy faster computer
  - might give you 10 – 100 times speedup
- Study the algorithms
  - might give you a million times speedup

Connectivity Example

- Given a pair of relationships between items, we want to know if a relationship can be inferred for a new pair a,b
- 3-4
- 4-9
- 8-0
- 2-3
- 5-6

? 2-9 ?

Graphical Example



Applications

- network communications
- circuitry
- mapping software
- variable name equivalence
- telephone network
- computer chip design

basic idea

- outline the problem
- understand clearly what kind of questions you are answering
  - don’t do all the work only to discover you can’t answer the question at the end
- understand the resource requirements

Sample Problem

- Have a collection of index cards with everyone’s names on it
- I want to organize it in alphabetical order
- Any ideas ??

Straightforward

- Find first name in list by going through it
- Find next
- etc

- Feels slow, how ?

Creative Approach

- Throw list in the air and make a new pile

- Will this ever find a solution ??

- any better ?

Fastest Solution

- Take a random name
  - can throw in air if you wish
- Sort into two piles
- redo from start
- known as quicksort, will cover when we cover sorting routines

Measurements

- Time
  - When designing an algorithm, think how fast it will run….then prove it
- Space
  - how much memory will it take up ?
  - important since we tend to treat memory as infinite
- Complexity
  - how easy it is to understand
    - given two algorithms, one complicated and one clear, tend to prefer the clear one

C++ Review

- Would like to review relevant C++
- make sure you can do the home works
- make sure you can do the work

Programming Environment

- online:
  - cunix
- Laptop/Desktop
  - cygwin
  - emacs
  - eclipse + (c++ plugin)

Basics

- You should be familiar with creating basic c++ programs
- Basic logical structure
- Basic types
- Basic function programming
- Basic memory manipulations
  - pointers
  - refrences

- We will now review some basics relating to dealing with classes and instances

CPP classes

- A class if a collection of functions and member variables
- instances of a class is called an object

- special functions called constructors and destructors can be automatically invoked

Question

- Anyone remember how to define a constructor ?
- destructor ?
- When are they invoked ?
- How to prevent them from being invoked?

Types of Functions

- Accessor
  - get some state information from the object
- Mutator
  - change information
- Helper
  - internal functions to accomplish tasks cleanly
- Predicate
  - help answer simple yes/no questions

Example

- want to represent a memory cell which can hold an integer value
- Call the class IntCell

Example

```
class IntCell {
private:
  int storedValue;
public:
  IntCell() { storedValue =0; }
  };
```

accessing variables

- `IntCell mycell;`
- how do you access the value ?
- how would you set the value ?

- `IntCell *cellPTR;`
- `cellPTR->read();`

abstraction

- important when defining a class to separate how to use the class and how we are representing the information

```
/**
 * A class for simulating an integer memory cell.
 */
class IntCell
{
  public:
    /**
     * Construct the IntCell.
     * Initial value is 0.
     */
    IntCell( )
      { storedValue = 0; }

    /**
     * Construct the IntCell.
     * Initial value is initialValue.
     */
    IntCell( int initialValue )
      { storedValue = initialValue; }

    /**
     * Return the stored value.
     */
    int read( )
      { return storedValue; }

    /**
     * Change the stored value to x.
     */
    void write( int x )
      { storedValue = x; }

  private:
    int storedValue;
};
```

Code Practice

- Any ideas of how to add a unique counter to each instance ?

Hands on Coding

- code the counter class
- add a static member ID (you need myid)

misc stuff

- review of misc things to do with basic class programing in C++

const class members

- const class members are assigned at construction time using the : notation

```
class Worker {
public:
   Worker(int id,int job);
   int getID () const;
private:
   const int _ID;
   int _job;
}
```

constructor

```
Worker(int id, int job) : _ID(id) {
   _job = job;
}
```

issues

- you should be careful about not returning private references
- can use const on functions when dealing with const arguments or member variables

const

- Allows the compiler to know which values shouldn’t be modified
- Very useful in your functions to either return const reference or make sure a pointer doesn’t alter the original object
- Example:

```
const int a = 5;

void foo(const int x) {   }
```

Const pointer to non-const

- This is a pointer which always points to same location, but the value can be modified

- int * const ptr = &x;

```
*ptr = ??
can't say
ptr = & ??
```

Const pointer to const data

- Int x = 200;
- const int * const ptr = &x;

- Some confusion
  - int const * X
  - const int * X //variable pointer to const
  - int * const Y //const pointer to int
  - int const * const Z //const point to const

Pointers to functions

- You can also pass around a pointer to a function
- `void foo (int , `**`int (*) (int , int)`**` );`
- `int example1(int x, int y) { return x+y; }`
- `foo(5, example1);`

Usage

- ```
  void foo(int a, int (*A)(int,int)){

  if((*A)(5,10) > 0){

  }
  else {

  }

  }
  ```

Classes within classes

- class member variables can be other classes
- important: member constructors are actually called before main class constructors
  - does this make sense ?
  ```

this

- this is a keyword
- represents a pointer to the class itself
- this->x
- or (*this).x

static

- static members have instance wide scope and livability
- great for shared variable
- have to be careful how used

assert

- special macro runs a test
- if true continues
- if false
  - dies without calling destructors

friends

- can declare a function to be a friend
- allows access to private member of the class
- not scoped during definition

What can go wrong

- The good thing about cpp is that your program can now crash many times even before reaching main ☺
- secret: understanding scope

Ordering and where to look for problems

- Global variables
  - Assignments and constructors
  - What else ??
- Main
- Local variables
- End local variables
- End main
- Global destructors

Class friends

- allows two or more classes to share private members
- e.g., container and iterator classes
- friendship is not transitive

Operator overloading

- Most operators can be overloaded in cpp
- Treated as functions
- But its important to understand how they really work

- +
- ~
- -
- !
- =
- *
- /=
- +=
- <<
- >>
- &&
- ++
- []
- ()
- new
- delete
- new[]
- ->
- >>=

Look up list

- X = X + Y
- Need to overload
  +
  =
- But this doesn’t overload +=

- Functions can be member or non-member
- Non-member as friends
- If its member, can use this
- (), [], -> or any assignments must be class members

- When overloading need to follow set function signature

unary

- Y += Z
- Y.operator+=(Z)

- ++D
- member
  - D.operator++()
- Non member
  - operator++(D)

- Functions can be member or non-member, your choice!
  - Non-member as friends if need private data
- If its member, can use the *this* pointer

- Exception: operators (), [], -> or any assignments must be class members

- When overloading need to follow set function signature

cout

- cout << yourclass

- left operand is ostream &
- so non member functions (belongs to ostream)
- friend if you would like

- lets code something

String class

- lets define a simple string class
- put output in its const and dest so we can follow
- constructor should take `const char *`
- would like to have following defined:

```
int length();
int hash();
```

- any ideas on how to do it ?

overload printing

```
friend ostream & operator <<(ostream &, const String
  &);

ostream &operator<<(ostream &output, String &str) {
output << “’” << ???? << “’”;
return output;
}
```

note

- when you call:

```
cout << s1 << s2;
```

- it is first:

```
operator<<(cout,s1)
```

- and then

```
operator<<(cout,s2)
```

Next

- want to overload the unary operator !
- test if a string is blank
- int operator!() const;
- or
- friend int operator(const String &);
- !s1
- s.operator!() or operator!(s)

same idea

- const String operator+=(const String &)
- vs
- friend const String &operator+=(Stirng &, const String &)
- what will s1 += s2 produce ?

- so how can we tell the difference between ++s1 and s1++

signatures

- s1++
- s1.operator++(0)
- operator++(s1,0)

- ++s1;
- s1.operator++()
- operator++(s1)

reuse

- one of the powers to OOP is the idea of reuseability
- if I spend 5 billion hours working on my code, I probably want to get some use out of it outside of the specific task
  - design issues
  - extension issues

Separation

- .h files include your design
- .cpp files your implementation

preprocessor

- should be familiar with basic #define preprocessor directives
- anyone remember how to prevent an error if the same .h file is included twice in a project ??

- #ifndef __something__unique__
- #define __something__unique__
- #endif

inheritance

- idea: allow a new class to inherit data members and functions from a base class
- can add members and functions
- represents a more specific idea
- vehicle -> minivan

- you can access protected members of parent
- can not access private members of parent
  - can still use public accessors and modifiers

code

class IntArray: public Array {

- simplest type of inheritance
- private members not inherited
- public/protected inherited accordingly

code

- create a point class
  - setPoint
  - <<
- derive Square
  - getArea()
  - <<

overriding

- we can redefine a base class function in the derived class and have c++ call the correct one

Question

- can
- Point *pp1;
- Square *sp1;

- given
- Point p = Point(3,4);
- Square s = Square(..

- can we say:
- pp1 = s ?????
- sp1 = p ?????

private inheritance

- we have used public inheritance
- private inheritance makes everyone from the base class come in as private members of the derived class

base class constructors

- need to launch base class constructor in derived class if you don't want the default to be called
- destructors are reversed
- lets see this in action

is a vs has a

- one important design decision is to know when to derive and when to use member variable

issue

- one issue with overriding, is that if the derived class doesn't provide a function, we will use the base class definition
- this doesn't always make sense
- Example I want a function MPG for any type of vehicle, but doesn't make sense of base class

virtual functions

- solution :
- declare the function to be virtual
- virtual double MPG();
- allow you to use a base class pointer to call at runtime the correct function (polymorphism)

abstract class

- sometimes its even useful to have a base class which can't be instantiated
- if any virtual function is declared pure virtual:
- virtual int MPG() = 0;

note

- constructors can not be virtual
- need virtual destructors to make everything work if you are going to have destructors in any of your classes (do it anyway)

Class derivation

- encapsulation
  - derivation maintains encapsulation
  - i.e., it is better to expand IntArray and add sort() than to modify your own version of IntArray
- friendship
  - not the same as derivation!!
  - example:
- is a friend of
- B2 is a friend of B1
- D1 is derived from B1
- D2 is derived from B2
- B2 has special access to private members of B1 as a friend
- But D2 does not inherit this special access
- nor does B2 get special access to D1 (derived from friend B1)

Derivation and pointer conversion

- derived-class instance is treated like a base-class instance
- but you can't go the other way
- example:

```
main() {
IntArray ia, *pia;
// base-class object and pointer
StatsIntArray sia, *psia;
// derived-class object and pointer
pia = &sia; // okay: base pointer -> derived object
psia = pia; // no: derived pointer = base pointer
psia = (StatsIntArray *)pia; // sort of okay now since:
// 1. there's a cast
// 2. pia is really pointing to sia,
// but if it were pointing to ia, then
// this wouldn't work (as below)
psia = (StatsIntArray *)&ia; // no: because ia isn't a StatsIntArray
```

Compiler issues

- Back to our IntCell example:

```
IntCell icell;
icell = 37;
```

- will this compile ??

what happens

- IntCell temp(37);
- icell = temp;

explicit

- explicit keyword tells the compiler to not create constructors in the background for you

```
/**
 * A class for simulating an integer memory cell.
 */
class IntCell
{
  public:
    explicit IntCell( int initialValue = 0 )
      : storedValue( initialValue ) { }
    int read( ) const
      { return storedValue; }
    void write( int x )
      { storedValue = x; }

  private:
    int storedValue;
};
```

reminder

- pointer to objects has slight behavior differences

```
int main( )
{
    IntCell *m;

    m = new IntCell( 0 );
    m->write( 5 );
    cout << "Cell contents: " << m->read( ) << endl;

    delete m;
    return 0;
}
```

Templates

```
template<typename X>
void foo(X &first, X second){
  first += second;
  }
```

see book for complete review

```
/**
 * Return the maximum item in array a.
 * Assumes a.size( ) > 0.
 * Comparable objects must provide operator< and operator=
 */
template <typename Comparable>
const Comparable & findMax( const vector<Comparable> & a )
{
    int maxIndex = 0;

    for( int i = 1; i < a.size( ); i++ )
        if( a[ maxIndex ] < a[ i ] )
            maxIndex = i;
    return a[ maxIndex ];
}
```

```
int main( )
{
    vector<int>     v1( 37 );
    vector<double>  v2( 40 );
    vector<string>  v3( 80 );
    vector<IntCell> v4( 75 );

    // Additional code to fill in the vectors not shown

    cout << findMax( v1 ) << endl;  // OK: Comparable = int
    cout << findMax( v2 ) << endl;  // OK: Comparable = double
    cout << findMax( v3 ) << endl;  // OK: Comparable = string
    cout << findMax( v4 ) << endl;  // Illegal; operator< undefined

    return 0;
}
```

```
/**
 * A class for simulating a memory cell.
 */
template <typename Object>
class MemoryCell
{
  public:
    explicit MemoryCell( const Object & initialValue = Object( ) )
      : storedValue( initialValue ) { }
    const Object & read( ) const
      { return storedValue; }
    void write( const Object & x )
      { storedValue = x; }
  private:
    Object storedValue;
};
```

```
int main( )
{
    MemoryCell<int>    m1;
    MemoryCell<string> m2;( "hello" );

    m1.write( 37 );
    m2.write( m2.read( ) + "world" );
    cout << m1.read( ) << endl << m2.read( ) << endl;

    return 0;
}
```

STL

- standard template library
- tons of useful stuff here
  - they've worked out all the bugs ☺
  - very efficient
  - make sure you understand what you are doing
- #include <vector>
- #include <string>

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

int main( )
{
    vector<int> squares( 100 );

    for( int i = 0; i < squares.size( ); i++ )
        squares[ i ] = i * i;

    for( int i = 0; i < squares.size( ); i++ )
        cout << i << " " << squares[ i ] << endl;

    return 0;
}
```

Reviewing

- make sure you are comfortable writing c++ code
- please speak to me ASAP if you need more help/reading etc
- Please ask if you need help
- Read Chapter 1 (ending) for more examples

Switch Gears

- Back to DS & A
- Lets assume we have some algorithm
- Lets discuss how to measure algorithms

Model of Computation

- In order to analyze algorithms
  - Will want to consider a model to study what it means to compute
  - would like to create classes of algorithms, so that we can talk about them in a uniform way
    - broad catagories
  - Will make some simplifications

Simplifications

- Computation
  - Assume every step of the algorithm takes one step
    - Different than real life
      - Generally Addition/subtraction < Multi <<< Division
      - CPU tasks << Memory access <<<<<< Disk access
      - Will come back to this when we discuss multi threaded environments
- Space
  - Assume infinite memory
    - Will adjust later
- Time
  - Will be counting time steps

Definition - Theta

- $T(N) = \Theta(g(N))$
  - set of functions $f(N)$ are in $\Theta(g(N))$

    if there exists positive constants c_1, c_2, n_0

    such that $0 < c_1\ g(N) < f(N) < c_2\ g(N)$

    for all $N \geq n_0$

- theta bound is strongest bounding
- real world sometimes hard to make such guarantees
- need to relax bound

Big-O

- $T(n) = O(g(N))$

  if there are positive constants c and n_o such that
  $T(N) \leq c\, g(N)$ when $N \geq n_o$

- Known as Big O notation

- Asymptotic in the upper limit

Omega

- lower bound only

little o

- little -o provides an upper bound but not a tight one
- doesn't say much
- should be aware of it

Functions

- We would like to use functions to describe the growth of some resource by an algorithm
- Want to compare different algorithms by growth rate
- Big O allows us to define an upper bound on a function
- So we can say:
  something is on the order of Big-O of something else

Careful

- On small input sizes, it is hard to analyze an algorithm
- Might be lucky
- Its been shown time and time again that something which just “works” but poorly designed can have some very expensive ramifications when scaling goes up.

Simplification

- Say an algorithm is said to run in $3n^2 + 2n + 5$
- Drop constants
- Drop low order polynomial terms
- We are interested in the function as it is taken to the limit

What to analyze

- Input size is strong consideration
- Generally an algorithm might have
  - Best case (ha!)
  - Worst case
  - Average case
- Which is most interesting?

Other considerations

- Remember it's a great tool, but very simplified
- Programming language
- Compiler
- Computer code

Example to analyze

```
int sum(int n) {
   int part_sum = 0;

   for(int i=0;i <= n; i++){
      part_sum += i * i * i;
   }
   return part_sum;
   }
```

- What is the runtime of this algorithm in terms of a function ?

General rules

- For simplification here are some general rules

For Loop

- Running time of a for loop is at most the running time of statements inside (plus tests) multiplied by number of iterations

Nested Loops

- Analyze inside out

```
for ( $i = 0; $i < $n; $i++ )
   {
   for( $j = 0; $j < $n; $j++ )
   {
      k++;
   }
}
```

More Rules

- Consecutive statements
  - Just add consecutive statements within a code block
- If/else
  - The runtime of if/else is the test plus the larger of the running time
  - Take worst behavior

Example

```
for( $i = foo_1(); $i < $n; $i++)
   {
      somesub($i);
      $total += foo2();
   }
```

Practice

- Lets do some simple examples

Example 1

```
int findMax(int list[],int max){

int maxValue = list[0];
for(int i =0; i < max; i++) {
    if( maxValue < list[i]) {
        maxValue = $list[i];
    }

    }

return maxValue;
}
```

Example 2

```
int Example2 (int list[],int max) {
int k =0;

for(int i =0; i < max; i++) {
   for(int j =0; j < max; j++) {
      k = (i * j) + n;
   }
}
   return k;
}
```

Example 3

```
int Example3(int n){

int k =0;
for(int i =0 ; i < 1000; i++) {
      k = k + n;
   }
return k;
}
```

Example 4

```
sub Example4(int n) {

int k =0;
for( int i=0; i < n; i++) {
   for(int j =0; j < n *n; j++) {
      k = (i * j) + n;
   }
}
return k;
}
```

Example 5

```
int Example5(int n) {
int k =0;
for(int i =0; i < n; i++) {
   for(int j =0; j < n; j++) {
      k += Example4(n);
   }
}
return k;
}
```

Example 6

```
int Example6(int n) {
int k =0;
while(n > 1) {
   n -= 1;
   k++;
}
return k;
}
```

Example 7

```
int Example7( int n) {
int k =0;
while (n > 1) {
   n = n / 2;
   k++;
}
return k;
}
```

Example 8

```
int Example8 (int n) {
if(n == 0 ) {
  return 1;
} else {
  return Example8(n/2) + 1;
}
}
```

Example 9

```
int Example9( int n ) {
   if( n <= 1 ) {
      return 1;
   } else {
      return Example9(n -1) + Example9(n-2);
}
}
```

Question

- Given a sequence of numbers (possibly negative) $A_1, A_2, .., A_n$ what is the sequence for the maximum subsequence value (0 if all are negative)

  -2, 11, -4, 13, -2, -10

Quick attempt

- Try to write some pseudo code, and provide a rough analysis for the running time

```
1  /**
2   * Cubic maximum contiguous subsequence sum algorithm.
3   */
4  int maxSubSum1( const vector<int> & a )
5  {
6      int maxSum = 0;
7
8      for( int i = 0; i < a.size( ); i++ )
9          for( int j = i; j < a.size( ); j++ )
10         {
11             int thisSum = 0;
12
13             for( int k = i; k <= j; k++ )
14                 thisSum += a[ k ];
15
16             if( thisSum > maxSum )
17                 maxSum = thisSum;
18         }
19
20     return maxSum;
21 }
```

- Line 13,14 O(1)
- Loops are of N
- 3 loops inside each other

- What is the run time actually ?
- What is the big O of n?

How can we improve?

- Think about the triple loop

```
/**
 * Quadratic maximum contiguous subsequence sum algorithm.
 */
int maxSubSum2( const vector<int> & a )
{
    int maxSum = 0;

    for( int i = 0; i < a.size( ); i++ )
    {
        int thisSum = 0;
        for( int j = i; j < a.size( ); j++ )
        {
            thisSum += a[ j ];

            if( thisSum > maxSum )
                maxSum = thisSum;
        }
    }

    return maxSum;
}
```

We can do better

- Next algorithm builds on a popular principle of “divide and conquer”
- Divide the set of number into 2 halves
  - Might be on left
  - Might be on right
  - Might span both sets
- So how to do we analyze the running time?

```
/**
 * Linear-time maximum contiguous subsequence sum algorithm.
 */
int maxSubSum4( const vector<int> & a )
{
    int maxSum = 0, thisSum = 0;

    for( int j = 0; j < a.size( ); j++ )
    {
        thisSum += a[ j ];

        if( thisSum > maxSum )
            maxSum = thisSum;
        else if( thisSum < 0 )
            thisSum = 0;
    }

    return maxSum;
}
```

- So why does the linear solution work ?
- Any thoughts ?
- So running time is easy to calculate
- How correct is it ?

Next

- Get book
- set up working environment
- Read chapters 1,2
- Download homework, start working on it
- start skimming 3-3.2