Biostatistics 615/815 - Statistical Computing
Lecture 1 : Introduction to Statistical Computing

Hyun Min Kang

September 4th, 2012

Welcome to BIOSTAT615/815

Basic Information

- Instructor : Hyun Min Kang
- Time : Tuesday and Thursday 8:30-10am
- Course Web Page : http://goo.gl/9DoFo

Today's outline

- Audience Polls
- Course Introduction
- Introductory Examples

Audience Polls : Enrollment

Which course did you register for?

1. BIOSTAT615
2. BIOSTAT815
3. Not registered

Audience Polls : Background

(Choose all) I have experience in (beyond novice level)

1. C/C++
2. R
3. Java
4. perl, python, php, or ruby
5. UNIX environment

Audience Polls : Operating Systems

(Choose all) I am used to the following operating systems

1. Windows
2. MacOS
3. UNIX

Audience Polls : Active Learning

During the class, I can connect to the Internet via laptop or smartphones

1. Yes
2. No

I am familiar with writing/sharing documents via Google Docs

1. Yes
2. No

Audience Polls : Current Status

Answer Yes/No to each of the questions

1. I can write "Hello, World" program with C++
2. I can explain the difference between value type, reference type, and pointer type in C++
3. I can describe what QuickSort is.
4. I can describe what Hidden Markov Model is.
5. I can describe what E-M algorithm is.
6. I can write a C++ program solving linear regression $\mathbf{y} = X\beta + e$

BIOSTAT615/815 Overview - Objectives

1. Equip the ability to **implement** computational and/or statistical **ideas** into working **software programs**.
   - ✓ Understand the concept of algorithm
   - ✓ Understand basic data structures and algorithms
   - ✓ Practice the implementation of algorithms into programming languages
   - ✓ Develop ability to make use of external libraries

BIOSTAT615/815 - Objectives

1. Equip the ability to **implement** computational and/or statistical **ideas** into working **software programs**.
2. Learn **computational cost** management in developing statistical methods.
   - ✓ Understand the practical importance of computation cost in many statistical inference applications.
   - ✓ Develop the ability to estimate computational time and memory required for an algorithm given data size.
   - ✓ Develop the ability to improve computation efficiency of existing algorithms and to optimize the cost/accuracy trade-off.

BIOSTAT615/815 - Objectives

1. Equip the ability to **implement** computational and/or statistical **ideas** into working **software programs**.
2. Learn **computational cost** management in developing statistical methods.
3. Understand **numerical methods** useful for statistical inference
   - ✓ Learn numerical optimization methods for solving analytically intractable problems computationally
   - ✓ Understand a variety of randomized algorithms for robust and efficient estimation of computationally intractable problems to obtain deterministic solution.

Why Is Statistical Computing Important?

1. Now is "Big Data" era
   - ✓ Next-generation sequencing studies often become >100TB in size
   - ✓ Storing MRI sessions of 1,500 subjects requires 5.4TB of data.
   - ✓ Google Maps has over 20 petabytes of data
   - Computation of simple statistics (e.g. mean) will take a long time.
2. Efficiency affects the feasibility of statistical methods
   - ✓ In statistical inference from genome-wide data, it is not common that more accurate methods takes much longer time than less accurate approximation (e.g. 40 years vs 1 day).
   - ✓ Implementation with low-level languages such as C++ has more room for speed improvements than higher level languages such as R or SAS.
   - ✓ Even with the same language, different algorithms can result in substantial gain in speed without losing accuracy.

What Will Be Covered?

1. C++ Basics and Introductory Algorithms
   - Computational Time Complexity
   - Sorting
   - Divide and Conquer Algorithms
   - Searching
   - Key Data Structure and Standard Template Libraries
   - Dynamic Programming
   - Hidden Markov Models
   - Interfacing between C++ and R

What Will Be Covered? (cont'd)

1. C++ Basics and Introductory Algorithms
2. Numerical Methods and Randomized Algorithms
   - Random Numbers
   - Matrix Operations and Least Square Methods
   - Importance Sampling
   - Expectation-Maximization
   - Markov-Chain Monte Carlo (MCMC) Methods
   - Simulated Annealing
   - Gibbs Sampling

Textbooks

- *"Introduction to Algorithms"* (Recommended)
  - ✓ by Cormen, Leiserson, Rivest, and Stein (CLRS)
  - ✓ Third Edition, MIT Press, 2009
- *"Numerical Recipes"* (Recommended)
  - ✓ by Press, Teukolsky, Vetterling, and Flannery
  - ✓ Third Edition, Cambridge University Press, 2007
- *"C++ Primer Plus"* (Optional)
  - ✓ by Stephen Prata
  - ✓ Sixth Edition, Addison-Wesley, 2011

Assignments and Grading

BIOSTAT615

- Biweekly Assignments - 50%
- Midterm Exam - 25%
- Final Exam - 25%

BIOSTAT815

- Biweekly Assignments - 33%
  - Expected to solve extra problems on top of 615 assignments
- Midterm Exam - 17%
- Final Exam - 17%
- Projects, to be completed in pairs - 33%

Target Audience for BIOSTAT615

- Students may have little experience in C++ programming
  - But students must be strongly motivated to learn C++ programming
  - Students with no/little experience in C++ are expected to spend additional hours than other students to accomplish homework
- Students should be familiar with basic concept of probability distribution, hypothesis testing, and simple regression
  - But BIOSTAT601 is not strictly required
- Students are expected to know or willing to learn basics of R language.

Target Audience for BIOSTAT815

- Students are expected to have decent experience in C/C++/Java programming
  - And expected to be fluent in C++ enough to be able to accomplish term projects.
  - List of suggested projects will be announced in the next week.
  - Students must be strongly motivated to learn C++ programming
- Students should be familiar with basic concept of probability distribution, hypothesis testing, and simple regression
  - But BIOSTAT601 is not strictly required
- Students are expected to know or willing to learn basics of R language.

Class Schedule

Total of 22 lectures T/Th 8:30-10am except for

- Fall Study Break : Tuesday, October 16
- **Midterm** : Tuesday, October 23
- Instructor out of town (1000G meeting) : Tuesday, November 6
- Instructor out of town (ASHG meeting) : Thursday, November 8
- Thanksgiving Break : Thursday, November 22
- **Final exam** : Tuesday, December 11

Homework Assignments

Schedules

- Homework 0 (Announcement: 9/4, Due: 9/10)
- Homework 1 (Announcement: 9/11, Due: 9/22)
- Homework 2 (Announcement: 9/25, Due: 10/6)
- Homework 3 (Announcement: 10/9, Due: 10/20)
- Homework 4 (Announcement: 10/25, Due: 11/10)
- Homework 5 (Announcement: 11/13, Due: 11/24)
- Homework 6 (Announcement: 11/27, Due: 12/10)

Office hours

- When : Friday 9:00-10:30am
- Where : M4531 SPH II

Submission of Homework Assignments

- Programming language must be in C++
- Assignments must be submitted in both of two formats
  - Google Docs document shared only to instructor and grader
  - Compressed source codes ready to run in Linux environment.
- The grade of late submission of homework will be multiplied by a factor of $e^{-\Delta m/14400}$, where Δm is the difference between the due and submission time in minutes

Using Google Documents for Homework

Steps to share a google document

1. Use your umich google account than other google accounts.
2. Make sure to rename the document title into the following format : "[BIOSTAT615] (or [BIOSTAT815]) Homework 1 - John Doe"
3. Click "Share" and type hmkang@umich.edu in "Add people", and give "Can edit" permission and do NOT modify after submission due.
4. When grading is finished, you will be notified by email.
5. Grader edit documents with comments and corrections if necessary. You can see comments and revision history.

- Make sure not to modify the the submitted homework until the grading finishes. The late submission penalty will rely on the last modified time.

Assignments are Very Important

- The main objective of the course is to develop the ability to implement software on your own.
- You will meet the instructor's teaching goal for BIOSTAT615 if you can accomplish your homework on your own.
  - If you got a good grade from BIOSTAT615, it should suggest that you are likely capable of use C++ and numerical methods for your research.
- You will meet the instructor's teaching goal for BIOSTAT815 if you can implement sophisticated statistical methods that are useful and convenient for others' use.
  - If you got a good grade from BIOSTAT815, it should suggest that you are likely capable of develop and release a software for your research community.

Honor code

- Honor code is STRONGLY enforced throughout the course.
  - The key principle is that all the code you produce must be on your own.
  - See http://www.sph.umich.edu/academics/policies/conduct.html for details.
- You are NOT allowed to share any piece of your homework with your colleagues electronically (e.g. via E-mail or IM), or by a hard copy.
- Discussion between students are generally encouraged
  - You may help your colleague setting up the programming environment necessary for the homework.
  - You may help debugging your colleagues' homework by sharing your trial and errors only up to non-significant fraction of your homework
  - Significant fraction of help can be granted if notified to the instructor, so that the contribution can be reflected in the assessment.
- If a break of honor code is identified, your entire homework (or exam) will be graded as zero, while incomplete submission of homework assignment will be considered for partial credit.

Algorithms

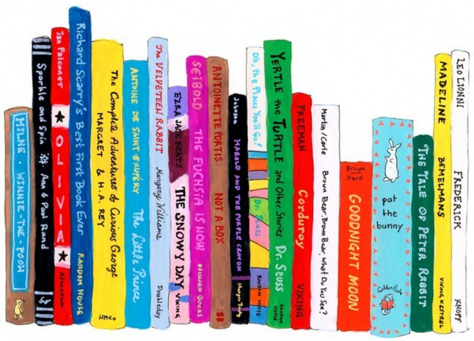
An Informal Definition

- An **algorithm** is a sequence of well-defined computational steps
- that takes a set of values as **input**
- and produces a set of values as **output**

Key Features of Good Algorithms

- Correctness
  - ✓ Algorithms must produce correct outputs across all legitimate inputs
- Efficiency
  - ✓ Time efficiency : Consume as small computational time as possible.
  - ✓ Space efficiency : Consume as small memory / storage as possible
- Simplicity
  - ✓ Concise to write down & Easy to interpret.

Sorting



How Would You Sort?

- 11
- 28
- 15
- 10

Note that..

- Computers are not as smart as you.
- Use only pairwise comparison and swap operations to sort them
- How many comparisons were made?

Sorting - A Classical Algorithmic Problem

The Sorting Problem

Input A sequence of n numbers. $A[1 \cdots n]$

Output A permutation (reordering) $A'[1 \cdots n]$ of input sequence such that $A'[1] \leq A'[2] \leq \cdots \leq A'[n]$

Sorting Algorithms

- Insertion Sort
- Selection Sort
- Bubble Sort
- Shell Sort
- Merge Sort
- Heapsort
- Quicksort
- Counting Sort
- Radix Sort
- Bucket Sort
- And much more..

A Visual Overview of Sorting Algorithms

http://www.sorting-algorithms.com

Today - Insertion Sort

http://www.sorting-algorithms.com/insertion-sort

Key Idea of Insertion Sort

- For k-th step, assume that elements $a[1], \cdots, a[k-1]$ are already sorted in order.
- Locate $a[k]$ between index $1, \cdots, k$ so that $a[1], \cdots, a[k]$ are in order
- Move the focus to $k+1$-th element and repeat the same step

Algorithm InsertionSort

Data: An unsorted list $A[1 \cdots n]$
Result: The list $A[1 \cdots n]$ is sorted
for $j = 2$ **to** n **do**
  $key = A[j]$;
  $i = j - 1$;
  while $i > 0$ *and* $A[i] > key$ **do**
    $A[i+1] = A[i]$;
    $i = i - 1$;
  end
  $A[i+1] = key$;
end

Correctness of InsertionSort

Loop Invariant

At the start of each iteration, $A[1 \cdots j-1]$ is loop invariant iff:

- $A[1 \cdots j-1]$ consist of elements originally in $A[1 \cdots j-1]$.
- $A[1 \cdots j-1]$ is in sorted order.

A Strategy to Prove Correctness

- Initialization: Loop invariant is true prior to the first iteration
- Maintenance: If the loop invariant is true at the start of an iteration, it remains true at the start of next iteration
- Termination: When the loop terminates, the loop invariant gives us a useful property to show the correctness of the algorithm

Correctness Proof (Informal) of InsertionSort

Initialization

- When $j = 2$, $A[1 \cdots j-1] = A[1]$ is trivially loop invariant.

Maintenance

If $A[1 \cdots j-1]$ maintains loop invariant at iteration j, at iteration $j+1$:

- $A[j+1 \cdots n]$ is unmodified, so $A[1 \cdots j]$ consists of original elements.
- $A[1 \cdots i]$ remains sorted because it has not modified.
- $A[i+2 \cdots j]$ remains sorted because it shifted from $A[i+1 \cdots j-1]$
- $A[i] \leq A[i+1] \leq A[i+2]$, thus $A[1 \cdots j]$ is sorted and loop invariant

Termination

- When the loop terminates $(j = n+1)$, $A[1 \cdots j-1] = A[1 \cdots n]$ maintains loop invariant, thus sorted.

Time Complexity of InsertionSort

Worst Case Analysis

for $j = 2$ **to** n	$c_1 n$
do	
$\quad key = A[j]$;	$c_2(n-1)$
$\quad i = j-1$;	$c_3(n-1)$
$\quad$ **while** $i > 0$ *and* $A[i] > key$	$c_4 \sum_{j=2}^{n} j$
$\quad$ **do**	
$\quad\quad A[i+1] = A[i]$;	$c_5 \sum_{j=2}^{n} (j-1)$
$\quad\quad i = i-1$;	$c_6 \sum_{j=2}^{n} (j-1)$
$\quad$ **end**	
$\quad A[i+1] = key$;	$c_7(n-1)$
end	

$$T(n) = \frac{c_4 + c_5 + c_6}{2} n^2 + \frac{2(c_1 + c_2 + c_3 + c_7) + c_4 - c_5 - c_6}{2} n - (c_2 + c_3 + c_4 + c_7)$$
$$= \Theta(n^2)$$

Summary - Insertion Sort

- One of the most intuitive sorting algorithm
- Correctness can be proved by induction using loop invariant property
- Time complexity is $O(n^2)$.

Environment for homework : Connect to scs.itd.umich.edu

See http://www.itcs.umich.edu/scs/ and http://www.itcs.umich.edu/ssh/ for details

1. Windows environment : Use PuTTY for command line and WinSCP for file transfer
2. MacOS X or UNIX
   - Command line ssh uniqname@scs.itd.umich.edu
   - File transfer scp [your-file] uniqname@scs.itd.umich.edu:[path]/[to]/[destination]

Tip : Add the following line in `~/.cshrc` file for more convenient command line (which shows current working directory).

```
set prompt="`whoami`@`hostname -s`:%~$ "
```

Steps for Homework 0

1. Create a directory `Private/biostat615/hw0/` under your home directory
   - Create a directory `Private/biostat615/hw0/` using WinSCP
   - Or type `mkdir --p ~/Private/biostat/hw0/` in the command line
   - Make sure your homework is in private space. If it is accessible by someone else, your homework will be discarded.
2. Create (or copy) a file (e.g. Private/biostat/hw0/helloWorld.cpp)
   - Directly type `vi Private/biostat615/hw0/helloWorld.cpp`
   - Or copy a file remotely using WinSCP or scp
3. Use basic Unix commands (e.g. `cd ~/my/path/`, `pwd`) to navigate between directories.
4. Compile and run the program (see the next slide)
5. Before submission, remove all the executable and object (.o) files, and compress your code (under Private/biostat615)

```
cd ~/Private/biostat615/
tar czvf uniqname_hw0.tar.gz hw0/
```


Getting Started with C++

Writing helloWorld.cpp

```
#include <iostream> // import input/output handling library
int main(int argc, char** argv) {
  std::cout << "Hello, World" << std::endl;
  return 0; // program exits normally
}
```

Compiling helloWorld.cpp

```
user@host:~$ cd ~/Private/biostat615/hw0/
user@host:~/Private/biostat615/hw0$ g++ -O -o helloWorld helloWorld.cpp
```

-O option will increase the computational speed. Use -g when you need debugging (e.g. with gdb)

Running helloWorld

```
user@host:~/Private/biostat615/hw0$ ./helloWorld
Hello, World
```


Implementing InsertionSort Algorithm

insertionSort.cpp - main() function

```
#include <iostream>
#include <vector>
void printArray(std::vector<int>& A);    // declared here, defined later
void insertionSort(std::vector<int>& A); // declared here, defined later
int main(int argc, char** argv) {
  std::vector<int> v; // contains array of unsorted/sorted values
  int tok;            // temporary value to take integer input
  while ( std::cin >> tok ) // read an integer from standard input
    v.push_back(tok);       // and add to the array
  std::cout << "Before sorting:";
  printArray(v);    // print the unsorted values
  insertionSort(v); // perform insertion sort
  std::cout << "After sorting:";
  printArray(v);    // print the sorted values
  return 0;
}
```


Implementing InsertionSort Algorithm

insertionSort.cpp - printArray() function

```
// print each element of array to the standard output
void printArray(std::vector<int>& A) { // call-by-reference : will explain later
  for(int i=0; i < A.size(); ++i) {
    std::cout << " " << A[i];
  }
  std::cout << std::endl;
}
```

Implementing InsertionSort Algorithm

insertionSort.cpp - insertionSort() function

```
// perform insertion sort on A
void insertionSort(std::vector<int>& A) { // call-by-reference
  for(int j=1; j < A.size(); ++j) { // 0-based index
    int key = A[j];  // key element to relocate
    int i = j-1;     // index to be relocated
    while( (i >= 0) && (A[i] > key) ) { // find position to relocate
      A[i+1] = A[i]; // shift elements
      --i;           // update index to be relocated
    }
    A[i+1] = key;    // relocate the key element
  }
}
```


Running InsertionSort Implementation

Test with small-size data (in Linux)

```
user@host:~/Private/biostat615/hw0$ ./insertionSort
11
28
15
20
(Press Ctrl-D, indicating end of input)
Before sorting: 11 28 15 10
After sorting: 10 11 15 28
```

Test with automatically shuffled input

```
user@host:~/Private/biostat615/hw0$ seq 1 20 | ~hmkang/Public/bin/shuf | \
                                    | ./insertionSort
Before sorting: 4 3 2 14 5 6 10 13 19 15 9 18 11 12 17 16 7 1 20 8
After sorting: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```


How fast is InsertionSort?

```
user@host::~/Private/biostat615/hw0$ time sh -c 'seq 1 100000 | \
                  ~hmkang/Public/bin/shuf | ./insertionSort > /dev/null'
0:03.33 elapsed, 3.334 u, 0.012 s, cpu 100.3%, 0 swaps, 0 rds, 0 wrts, \
pgs: 0 avg., 0 max.
user@host::~/Private/biostat615/hw0$ time sh -c 'seq 1 100000 | \
                  ~hmkang/Public/bin/shuf | sort -n > /dev/null'
0:00.16 elapsed, 0.157 u, 0.010 s, cpu 100.0%, 0 swaps, 0 rds, 0 wrts, \
pgs: 0 avg., 0 max.
```

default sort application is orders of magnitude faster than insertionSort

Summary

- Algorithms are sequences of computational steps transforming inputs into outputs
- Insertion Sort
  - ✓ An intuitive sorting algorithm
  - ✓ Loop invariant property
  - ✓ $\Theta(n^2)$ time complexity
  - ✓ Slower than default `sort` application in Linux.
- A recursive algorithm for the Tower of Hanoi problem
  - ✓ Recursion makes the algorithm simple
  - ✓ Exponential time complexity
- C++ Implementation of the above algorithms.

Homework 0

- Implement the following two programs and send the output screenshots to the instructor (hmkang at umich dot edu) by E-mail
  1. HelloWorld.cpp
  2. TowerOfHanoi.cpp
- Briefly describe your operating system and C++ development environment with your submission
- This homework will not be graded, but mandatory to submit for everyone who wants to take the class for credit
- No due date, but homework 0 must be submitted prior to submitting any other homework.

Next Lecture

- C++ Programming 101
- Implementation of Fisher's exact test