Biostatistics 615 - Statistical Computing

Lecture 4 Basical Algorithms I

Jian Kang

September 17, 2015

Summary of previous lectures

- Binary representation of integers
- Floating point representation
  - Single precision
  - Double precision
- Relative errors
  - 7 valid digits for `float`
  - 15 valid digits for `double`
  - Relative errors on arithmetic
- Basic syntax
  - Command line input arguments `argc` and `argv`
  - Convert string to integer `atoi` or floating-point `atom` with `#inlcude<cstdlib>`
  - Typecasting
  - Set precision in the standard output with `#inlcude<iomanip>`
- Three algorithms for compute sample mean and variance
- R storage

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) $A^0[1 \cdots n]$ of input sequence such that $A^0[1] \leq A^0[2] \leq \cdots \leq A^0[n]$

Sorting Algorithms

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

Recoures

- A Visual Overview of Sorting Algorithms http://www.sorting-algorithms.com
- 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.
- Recall how you sort a deck of playing cards.

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:

- $A[1 \cdots j-1]$ consists 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.

Notations for complexity

Big O notation

For two functions $f(x)$ and $g(x)$, $f(x) = O(g(x))$ if and only if there exists a positive real number M and a real number x_0 such that $f(x) \leq Mg(x)$ for all $x > x_0$.

Big Ω notation

For two functions $f(x)$ and $g(x)$, $f(x) = \Omega(g(x))$ if and only if there exists a positive real number M and a real number x_0 such that $f(x) \geq Mg(x)$ for all $x > x_0$.

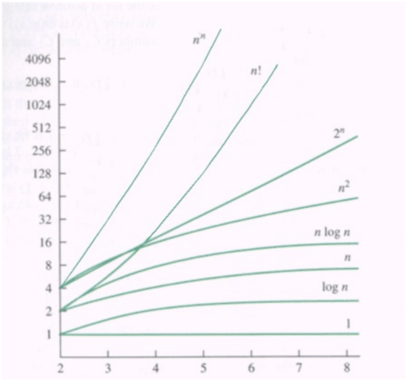
Big Θ notation

For two functions $f(x)$ and $g(x)$, $f(x) = \Theta(g(x))$ if and only if both $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

frequently used function classes

- $\Theta(1)$ is constant
- $\Theta(\log\log n)$ is double logarithmic
- $\Theta(\log n)$ is logarithmic
- $\Theta(n)$ is linear
- $\Theta(n\log n)$ is log-linear
- $\Theta(n^2)$ is quadratic
- $\Theta(n^3)$ is cubic
- $\Theta(n^p), p > 1$ is polynomial
- $\Theta(c^n), c > 1$ is exponential
- $\Theta(n!)$ is factorial

Growth of functions



http://www.cs.odu.edu/~toida/nerzic/content/function/growth.html

Examples

- $n = \Theta(2n)$
- $n = O(n^2)$
- $3n^3 + 2n^2 - 5n + 6 = \Theta(n^3)$
- $1 + 2 + \ldots + n = \frac{n(n+1)}{2} = \Theta(n^2)$
- $1^2 + 2^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$
- $1^3 + 2^3 + \ldots + n^3 = \frac{n^2(n+1)^2}{4} = \Theta(n^4)$
- $1^p + 2^p + \ldots + n^p = \Theta(n^{p+1})$ if $p > -1$
- $a + a^2 + \ldots + a^n = \frac{a^{n+1} - a}{a - 1} = \begin{cases} \Theta(a^n) & \text{if } a > 1; \\ \Theta(n) & \text{if } a = 1; \\ \Theta(1) & \text{if } 0 < a < 1. \end{cases}$
- $a + 2a^2 + \ldots + na^n = \frac{na^{n+2} - (n+1)a^{n+1} + a}{(a-1)^2} = \begin{cases} \Theta(na^n) & \text{if } a > 1; \\ \Theta(n^2) & \text{if } a = 1; \\ \Theta(1) & \text{if } a < 1. \end{cases}$

Examples (cont.)

- $\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{n} = 2^n = \Theta(2^n)$
- $\binom{n}{1} + 2\binom{n}{2} + \ldots + n\binom{n}{n} = n2^{n-1} = \Theta(n2^n)$
- $1 + \frac{1}{2} + \ldots + \frac{1}{n} = \Theta(\ln(n))$
- $\ln(2)^p + \ln(3)^p + \ldots + \ln(n)^p = \Theta(n \ln(n)^p)$ if $p > 0$
- $\ln(n!) = n\ln(n) - n + O(\ln(n))$ (Sterling's formula)
- $n! = O(n^n)$
- $n! = \Theta\left(\left(\frac{n}{e}\right)^n \sqrt{2\pi n}\right)$

Time Complexity of InsertionSort

Worst Case Analysis

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

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

Summary - Insertion Sort

- One of the most intuitive sorting algorithm
- Correctness can be proved by induction using loop invariant property
- It's a stable sorting algorithm
  - A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array.
  - Example: $a_1 = 3$, $a_2 = 3$ and $a_3 = 1$.

Stable sort: $(a_1, a_2, a_3) \to (a_3, a_1, a_2)$

- Time complexity is $O(n^2)$.

Implementing InsertionSort Algorithm

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

Useful standard class template: vector

```
//insertionSort.cpp
#include <iostream>
#include <vector>
using namespace std;
void printArray(vector<int>& A);  // declared here, defined later
void insertionSort(vector<int>& A);// declared here, defined later
//- main() function
int main(int argc, char** argv) {
  vector<int> v; // contains array of unsorted/sorted values
  int tok;            // temporary value to take integer input
  while ( cin >> tok ) // read an integer from standard input
    v.push_back(tok);       // and add to the array
    ... ...
}
```

- Sequence containers representing arrays that can change in size.
- Very efficient accessing its elements (`at()` or `[]`)
- Relatively efficient adding or removing elements from its end. (`push_back()`)

Implementing INSERTIONSORT Algorithm

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

- Declare the function parameters as references (`vector<int>& A`) rather than variables.
- `A.size()` returns the length of the vector.
- `A[i]` accesses its elements.
- What is the difference between `++i` and `i++`?

```
j=++i;          i=i+1;j=i;
k=i++;          k=i;i=i+1;
A[i++]= A[i],A[i+1]....
```

Implementing InsertionSort Algorithm

```
//insertionSort.cpp - insertionSort() function
// perform insertion sort on A
void insertionSort(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
  }
}
```

- `while`-loop simply repeats `statement` while `expression` is true.

  `while (expression) statement`

- Logical operator "`&&` ": AND. ("`||` " OR; "`!`" NOT)

Running INSERTIONSORT Implementation

Test with small-size data (in Linux)

```
$./insertationSort
4
5
1
3
2
(Press Ctrl-D, indicating end of input)
Before sorting: 4 5 1 3 2
After sorting: 1 2 3 4 5
```

Test with automatically shuffled input

```
$ seq 1 2 20 | shuf | ./insertionSort
Before sorting: 19 13 11 7 9 1 3 15 5 17
After sorting: 1 3 5 7 9 11 13 15 17 19
```

- `seq`: a utility for generating a sequence of numbers.
  Basic syntax: `seq first increment last`
- `a | b`: pipes the output of program `a` into program `b`.

How fast is INSERTIONSORT?

```
$ time sh -c 'seq 1 100000 | shuf | ./insertionSort > /dev/null'
0:02.44 elapsed, 2.433 u, 0.007 s, cpu 99.5%,
0 swaps, 0 rds, 0 wrts, pgs: 0 avg., 1420 max.

 $ time sh -c 'seq 1 100000 | shuf | sort > /dev/null'
0:00.22 elapsed, 0.217 u, 0.003 s, cpu 95.4%,
0 swaps, 0 rds, 0 wrts, pgs: 0 avg., 3854 max.
```

- `time`: can display the elapsed time during the execution of a command or script. Basic syntax: `time command`
- `sh`: a command language interpreter that executes commands read from a command line string (`-c` *command string*), the standard input, or a specified file.
- Linux/Unix default `sort` application is orders of magnitude faster than insertionSort
- `/dev/null` is a special file, not a directory. It is used to dispose unwanted output streams of a process, or as a convenient empty file for input streams.