Algorithms: Assignment #3

Due on March 20, 2016

Zhaoyang Li (2014013432)

Contents

Problem 1 **3**

Problem 2 **3**

Problem 3 **4**

Problem 1

CRLS Ex. 8.4-4.

This problem is all about defining the buckets, to make sure the expected numbers of points in each buckets n_i are the same.
Choose r_i such that $r_i = \sqrt{\frac{i}{n}}$, $i = 0, 1, 2, ...n$, meaning that

$$\forall i, \pi r_i^2 - \pi r_{i-1}^2 = \pi \frac{1}{n}$$

p_j falls into bucket i, if and only if $r_{i-1} \leq d_j < r_i$, $i, j \in \{1, 2, ..., n\}$.
It's obvious that the process of placing all the points into the buckets takes at most linear time.
Define indicator random variables $X_{ij} = I\{p_j \text{ falls in bucket } i\}$.
Therefore, $n_i = \sum_{j=1}^{n} X_{ij}$. Applying geometric probability model, we have $E(x_{ij}) = \frac{1}{n}, E(x_{ij}x_{ik}) = \frac{1}{n^2}$.

$$\begin{aligned} E\left(n_i^2\right) &= \sum_{j=1}^{n} E\left(X_{ij}^2\right) + \sum_{1\leq j\leq n} \sum_{1\leq k\leq n, k\neq j} E\left(X_{ij}X_{ik}\right) \\ &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{1\leq j\leq n} \sum_{1\leq k\leq n, k\neq j} \frac{1}{n^2} \\ &= 2 - \frac{1}{n} \end{aligned}$$

$$\begin{aligned} E\left(T(n)\right) &= \Theta(n) + \sum_{i=0}^{n-1} O\left(E\left(n_i^2\right)\right) \\ &= \Theta(n) + n \cdot O\left(2 - \frac{1}{n}\right) \\ &= \Theta(n) \end{aligned}$$

So the algorithm has a linear average running time.

Problem 2

Compare insertion sort, quick sort, merge sort and radix sort on randomly generated 32-bit unsigned integers.

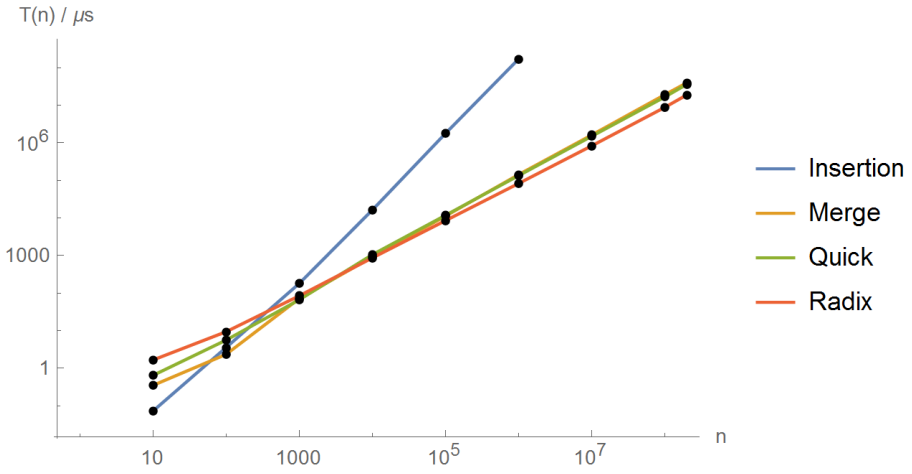
Problem 2 continued on next page...

Sorting algorithms are implemented in C++.
Correctness is verified by checking consistency of the results of the four algorithms.
Notes on generating large random integers: std::rand() gives 15-bit random integers, splicing three of which gives 32-bit random integers.
Execute `bin/SortAlgorithms.exe` to get a table like what is shown below.

CPU Time Consumed[a] (microseconds)

Size of problem	Insertion	Merge	Quick	Radix
10	0.0717057	0.347331	0.640546	1.64938
100	3.53864	2.32687	5.57795	9.32336
1000	179.396	71.1268	66.0131	84.4482
10000	16285.7	918.182	1052.08	863.248
100000	1.78e6	11222.2	11777.8	8500
1000000	1.71e8	143000	134000	83000
10000000		1.62e6	1.5e6	834000
100000000		1.92e7	1.67e7	8.85e6
200000000		3.99e7	3.64e7	1.9e7

A double-logarithmic chart visualizing the data above:



It's clear that insertion sort has the greatest time complexity $\Theta(n^2)$, while radix sort has the least $\Theta(n)$. Quick sort and merge sort are in between, and are basically the same. As size of the array grows, quick sort has the shorter running time.

[a]Compilation: MSVC++ 11.0 x64 Release; runs on an Intel Core i5 4120U Processor, 4GB DDR3 Memory, Windows 10 Pro.

Problem 3

Give an $O(n \lg n)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

Problem 3 continued on next page...

Combining dynamic programming and binary search techniques gives $O(n \lg n)$ time complexity.
The algorithm is implemented in C#. Execute `bin/LongestIncreasingSubsequence.exe` to bring up a GUI.
A run-time screenshot is shown below.