Random Sort

M. Schmittmann

May 2, 2008

Abstract

This article provides a formal definition of the Random Sort sorting algorithm as well as its complexity properties and theoretical and practical performance. Also a reference is made to the effect quantum (co)processors will have on the complexity-related issues which are inherently linked to the Random Sort-algorithm.

1 Definition

The Random Sort sorting algorithm can be defined very elegantly with the following pseudo-code:

$$\texttt{while(!sorted) \{ shuffleList(}X\texttt{) \}}$$

where

$$\texttt{sorted} := \neg\exists x_n, x_{n+1} \in X \Rightarrow x_{n+1} < x_n$$

This is a relatively simple algoritm, which can be described in practically every programming language. Implementations of the `shuffleList` function are native for most applications, a few examples have been given in appendix A. For the shuffling of elements in X, a RNG[1] has to be available. Many of them and their underlying theories have been widely discussed about, see [1, 3] for valid examples. Of course they could also be implemented as entities flipping coins or – preferably – hardware implementations of based on nuclear decay of radioactive elements [4].

2 Complexity

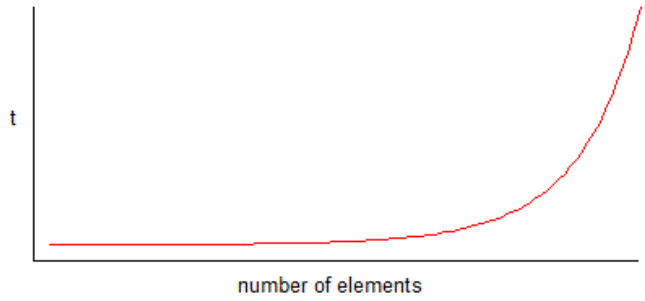
2.1 Worst Case

Very interesting things can happen when applying Random Sort to an unordered list. The hypothetical case where every iteration of the algorithm yields an unsorted list is theoretically possible, the worst case complexity is therefore $\mathcal{O}(\infty)$.

[1] Random Number Generator

2.2 Average Case

It is hard to define an average case complexity for this algoritm, since the sortedness of the list being sorted does not per-sé gradually increase or decrease with every iteration of the algorithm. Incrementing the length of the list however will exponentially decrease the chance of the algorithm being successful and therefore incrementing the time the sorting will probably take:



2.3 Best Case

As with most sorting algoritms, best case will be $\mathcal{O}(n)$, when the input list is already sorted.

3 Performance

Considering the nature of the Random Sort algorithm all states of the list being sorted come down to two essential states: sorted and unsorted. There is no degree of sortedness, nor is there any way of succesfully deriving a next state from the current state of the system or list. Any transition could result in the list being sorted. The next table demonstrates how the numbers of (unique) elements, and (unique) (sorted) states relate:

elements	1	2	2	3	3	3	4	4	4	4	...
unique elements	1	1	2	1	2	3	1	2	3	4	...
unsorted states	0	0	1	0	4	5	0	8	10	11	...
sorted states	1	2	1	6	2	1	12	4	2	1	...
unique states	1	2	2	6	6	6	12	12	12	12	...

In practice, for small lists this means there is a fairly good chance Random Sort will produce a sorted list in a considerate amount of time, since the number of unsorted states is lower and the probability of reaching the sorted state is higher. But as shown in 2.1, even for a list with 2 unique elements the time

required to sort the list could be infinite.

The performance of the algoritm is somewhat dependant on the quality of the RNG, and trivially also on the capacities of the hardware being used.

4 Quantum computing and Random Sort

Considering all all elements of the list are unique, there is 1 state which satisfies the `isSorted` condition as specified in 1. Theoretically, using quantumcomputing algorithms[2] that can simultaneously observe all possible states (since sorting is not NP-hard), the ordered state could be found by quantum interference.

Quantum interference is not specifically applicable on Random Sort – or even on sorting algorithms –, but will render the sorting algorithms mostly useless.

5 Conclusions

Random Sort is an interesting sorting technique, if it can even be called a sorting algorithm. Most algorithms are built for efficiëncy, so it is interesting to have a counterpart in the form of the ultimate inefficiënt algorithm with an unique complexity class for worst-case scenario's.

References

[1] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo random number generator. *SIAM Journal on Computing*, 15(2):364–383, 1986.

[2] A. Ekert and R. Jozsa. Quantum computation and Shor's factoring algorithm. *Reviews of Modern Physics*, 68(3):733–753, 1996.

[3] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Siam, 1992.

[4] J. Walker. HotBits: Genuine random numbers, generated by radioactive decay. *Online at http://www. fourmilab. com/hotbits*, 1999.

APPENDIX

A Example in a modern language

A.1 C++

```
#include <cstdlib>
#include <ctime>

void randomSort(int x[], int length) {
  srand(time(0));
  while(!isSorted(x,length)) do {
    shuffleList(x,length);
  }
}

bool isSorted(x[], int length) {
  for(int i=0; i < length - 1; length++) {
    if (x[i] > x[i+1]) {
      return false;
    }
  }
  return true;
}

void shuffleList(int x[], int length) {
  for (int i=0; i<(length-1); i++) {
    int r = i + (rand() % (length-i));
    int temp = x[i];
    x[i] = x[r];
    x[r] = temp;
  }
}
```