Dustin Cook

CSC 372 Analysis of Algorithms, Lisa Rebenitsch

Final Project

Fun-Sort, Sorting Unique SQL ID’s, C++

I implemented this algorithm in C++ however, I wrote a test script for it in python3. Usage for the program and correctness.py can be found near the end of this document.

You can binary search to find bad pairs in an array. The algorithm then performs insertions guided by the binary search on the unsorted array. It can be assumed that the array of integers given to Fun-Sort is an array of SQL ID’s.

# Fun-Sort runs in *n* phases. For phase *i*=1…*n*, binary search for the current content *ai* of cell *i* until the search is successful and “finds” the element in cell *i*. If the search fails, we find an index *h* such that *ah*−1<*ai*<*ah*. We then either swap the bad pair (*i*,*h*−1) if *i*<*h*−1, or the bad pair (*h*,*i*) if *h*<*i*. The phase continues with a search for the new content of cell *i*.

# Where It Is Used

# Toy algorithm for computer scientists with no practical industry use.

# Theoretical Correctness

# By induction, the first i – 1 cells are ordered correctly at beginning of phase i. If a search fails, we are not done with phase yet, and it returns an index *h* ≠ *i.* If h > i, the subsequent swap of (i, h-1) does not affect the sordidness of the first i cells. If h < i then we swap (h, i). Since binary search guarantees *ah*−1<*ai*<*ah,* this swap also maintains the sortedness of the first i – 1 cells. If the search for ai is successful (and phase i ends) then we have ai-1 < ai, and the first i cells are sorted as claimed.

# Run time

# Each binary search takes time O(logn). There are n successful searches, one per phase, and there are at most F = (n2) unsuccessful searches. Thus, the total time is O ((n + F) logn) = O(n2logn).

# Built-in Code Correctness Tests

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Description | Input | Actual output |
| Case 1 | Small distinct values | 1 0 2 3 | 0 1 2 3 |
| Case 2 | Large distinct values | 100000 200000 400000 -500000 | -500000 100000 200000 400000 |
| Case 3 | Small repeating values | 2 2 3 1 | 1 2 3 2 |

# References

Fun-Sort. (2004, June 08). Retrieved December 9, 2018, from https://www.sciencedirect.com/science/article/pii/S0166218X04001131

# Build / Usage

You start by compiling fun\_sort.cpp. No executable name is required “a.out” is fine and is what the python script is going to be looking for and “a.out” is what the gnu g++ compiler defaults to when given no name.

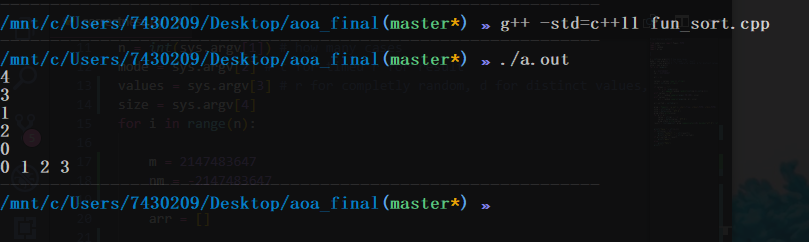
**Input:**

After you compile the program with a C++ compiler and run it, it will wait for input. It first expects an integer representing the problem size: N. It will then expect you to enter N integers separated by newlines.

**Output:**

The numbers given in sorted order.

**Example:**



**Run requirements:**

* Python3 to run correctness.py.
* A C++ compiler to compile fun\_sort.cpp.
* C++ 11.

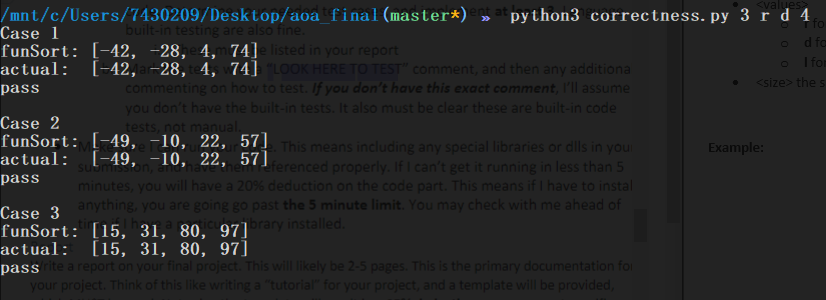
correctness.py

To use correctness.py, you have to have compiled fun\_sort.cpp to an executable named a.out from the previous example given. Correctness.py runs a given number of test cases for the fun-sort executable. It generates a random array, sends it to the fun-sort executable, and then receives the array back form the fun-sort executable. Correctness.py then sorts the array itself and compares the sorted array with the array returned from the fun-sort executable. Its output varies with the arguments given to correctness.py. If the two arrays are identical, it outputs pass, if the arrays are not identical (fun-sort failed), it will output FAIL.

**correctness.py usage**

python3 correctness.py <n> <mode> <values> <size>

* <n> Number of cases
* <mode> **r** for result “currently only option” possible values
* <values>
  + **r** for random (possible duplicates) [-50, 100]
  + **d** for small distinct [-50, 100]
  + **l** for large distinct [-2147483647, 2147483647]
* <size> the size of the array per test case, if size > 20, it will not print the two arrays to be viewed.

**Example: **

**Note: The script is case sensitive, and the order of arguments matters.**