GB21802 Programming Challenges

Week 2 - Sorting Algorithms

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Sciences

2015-04-20

Last updated April 20, 2015

Notes: Week 1 submissions results

- 3n+1:
- Chess:
- Erdos:
- Scoreboard:

Important note

The system just stores your First submitted code. If you want to send a new version after that, you have to e-mail me. Sorry about that.

The importance of Sorting

Sorting

Given a collection of items, change the position of those items inside the collection so that they obey some pre-defined order (alphabetical order, numerical order, etc)

Why is sorting important?

- Sorting the data is the first step of many well known algorithms;
- Sorting can also speed up some other algorithms;
- Sorting is a good way to start learning about Computer Science theoretical research;

The importance of Sorting (2)

Even Bill Gates has published a paper on Sorting!

Discrete Mathematics 27 (1979) 47–57.

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n+3$.

1. Introduction

We introduce our problem by the following quotation from [1]

Sorting Applications

Sorting can be used as a step in a huge number of algorithms. Let us check a few of them. Please give your own suggestions as well!

Sorting Applications (1)

Testing for Uniqueness or Duplicates

After sorting a data array, any duplicate data will be near each other. You can test for the existence of duplicates using only one pass through the array.

Sorting Applications (1)

Testing for Uniqueness or Duplicates

After sorting a data array, any duplicate data will be near each other. You can test for the existence of duplicates using only one pass through the array.

Unique subset

Using the same idea as above, you can find the largest subset of unique items, by keeping track of any unique items (or row of items) in a different array.

Frequency Counting, Finding the Mode

Using the same idea as above, you can count the frequency of a value (by counting duplicates). The value with the highest number of duplicates is the *Mode*.

Sorting Applications (2)

Finding the Median, or a quantile

You can find the n^{th} largest (smallest) element by sorting an array, and picking the n^{th} element in the sorted array.
The median is the element in the half of the array (different from the mean!).

Prioritizing Events

We can sort events by priority (or by the time they should happen). And now we have a priority queue.

Finding a target pair

We want to find a and b so that $a + b = z$. After finding the lowest value x in the array, we find y so that $z - x = y$, and we do the search from both ends.

Sorting Applications (3)

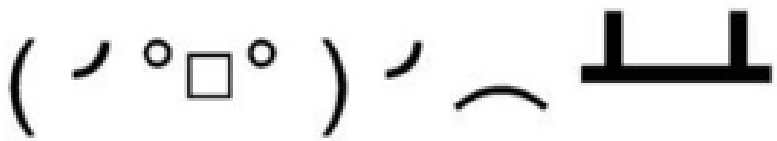
Can you give me suggestions of other uses for sorting?

How many sorting Algorithms are there?

- bubblesort
- insertion sort
- selection sort
- heapsort
- mergesort
- quicksort
- radix sort
- bin sort
- Shell sort
- gnome sort
- library sort
- comb sort
- tree transversal
- sorting networks
- cocktail shaking sort
- bucket sort
- bogo sort
- bitonick sort
- ...
- And many more!

How many sorting Algorithms are there?

- bubblesort
- insertion sort
- selection sort
- heapsort
- mergesort
- quicksort
- radix sort
- bin sort
- Shell sort
- gnome sort
- library sort
- comb sort
- tree transversal
- sorting networks
- cocktail shaking sort
- bucket sort
- bogo sort
- bitonick sort
- ...
- And many more!



Why are there so many sorting algorithms?

- Theoretical research in sorting can give insights about the theoretical basis of other algorithms;
- Some of these algorithms are very efficient in very specific situations;
- Sometimes you can exchange memory for time complexity, or vice-versa;

Why are there so many sorting algorithms?

For example, the Bill Gates Paper...

Discrete Mathematics 27 (1979) 47–57.

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n+3$.

1. Introduction

We introduce our problem by the following quotation from [1]

It focuses on sorting by "reverse radix" – an operation that reverses a sub-array. This is very important to calculate the genetic distance between two individuals. (How many reverse operations does it take to go from individual A to individual B?)

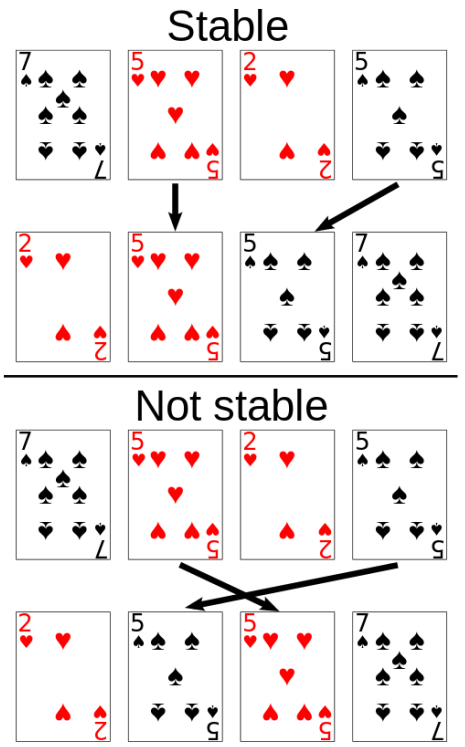
Characteristics of sorting algorithms

Computational Complexity

- How many comparisons? Comparisons require conditionals in the machine code, and sometimes can be, themselves, very expensive.
- How many swaps? Moving memory around can also be expensive, depending on how much memory we are using for the data (Main memory, cache, etc).
- Memory Requirements: Some algorithms require an extra array, or maybe even more memory, in exchange for a reduced number of comparisons/swaps.

Characteristics of sorting algorithms

- Stable Algorithms guarantee that two objects who are "equal" in the sorting order will maintain their relative position after sorting.
- Unstable Algorithms do not guarantee the final order of two objects who are "equal" in regard to the sorting comparison function.
- (for example, two objects containing the same data, but different memory addresses)



Characteristics of sorting algorithms

Other questions:

- Is the algorithm parallelizable?
- Is the algorithm recursive? How complex is the code?
- Is the algorithm based on comparisons, or on hashes?

Sorting Algorithm Review

Let us review quickly some sorting algorithms

Selection Sort

```
Divide data into sorted list and unsorted list;
For i = 1 to length of data:
    Select lowest element in unsorted list;
    Move selected element to the end of sorted list;
```

- Selection sort performs many comparisons, but few data swaps;
- Normally, it is very inneficient. But if you use the correct data structure for the unsorted data, it becomes the very efficient heapsort!

Merge Sort

```
If data size is < 3:
    sort data;
Else:
    Mergesort(first half)
    Mergesort(second half)
    Merge(first half, second half)
```

- The mergesort can be easily parallelized. You can sort HUGE amounts of data efficiently in a parallel context;
- In fact, it is implemented in many standard libraries;
- How would you implement it without recursion?

Quick Sort

```
Quicksort (dataarray):
  While dataarray > 1:
    p = Partition(dataarray);
    Quicksort(dataarray[:p-1]);
    Quicksort(dataarray[p+1:]);

Partition (d):
  Select a Pivot;
  PivPlace = 1;
  for i in d:
    if d[i] < Pivot:
       place d[i] in PivPlace;
       PivPlace++;
  place Pivot in PivPlace;
  return PivPlace;
```

- Quicksort is well known, because it is very fast in the average case.
- In the worst case, quicksort is actually slow (can you build a worst case?)
- Performance is heavily dependent of the smart selection of a pivot. Random pivots are usually an "okay" choice.

Bucket Sort

```
Create k Buckets;
For each element in data:
  Place element in correct bucket;
For i = 1 to k-1:
  Concatenate bucket i to bucket i+1;
```

- Can sort in O(n+k) – linear!
- However, the price to pay is the size of k;
- If you know the "absolute" positioning of the data, and there are many repetitions, k will be very
- On the other hand, for "infinite" data values, k can also get infinite!
- This method is not based on comparisons!

Sorting Methods Video

Movie time! Can you identify the sorting methods?

Why does the C++ sort leave "dirt" to be cleaned in the last pass?

Bogo Sort

The Worst Sorting Algorithm

1. Shuffle the Data
2. If the data is not sorted, return to 1

- Calculate the complexity of this Algorithm!
- Will this algorithm always stop?

The Quantum Bogo Sort

The Best Sorting Algorithm

1. Shuffle all the data;)
2. Check if the data is sorted;
3. If the data is not sorted, destroy the entire universe.

Sorting in Programming Challenges

Your language libraries have already implemented some sorting for you. You will often want to use that.

- Java: Collections.sort()
- C (stdlib.h): qsort();
- C++ (stl): sort(), stable_sort;

You usually have to implement the comparison function, except in trivial cases.

Implementing Sorting Functions

Sorting on multiple attributes of your data

Sort by height (ascending), in case of tie sort by last name (alphabetical), in case of tie sort by weight (descending), in case of tie sort by who has the birthday closest to June 30th.

How do you do that?

Implementing Sorting Functions (2)

Sorting on multiple attributes of your data

Sort by height (ascending), in case of tie sort by last name (alphabetical), in case of tie sort by weight (descending), in case of tie sort by who has the birthday closest to June 30th.

- Multiple Passes: Sort the items by the least important criteria, then by the next least important criteria, and so on. Simple, a bit slow, requires a stable sorting algorithm.

Implementing Sorting Functions (2)

Sorting on multiple attributes of your data

Sort by height (ascending), in case of tie sort by last name (alphabetical), in case of tie sort by weight (descending), in case of tie sort by who has the birthday closest to June 30th.

- Single Pass: Make a complex comparison function that takes all the sorting conditions into account. Faster than multiple passes, but be careful with code complexity!

Implementing Sorting Functions (2)

Sorting on multiple attributes of your data

Sort by height (ascending), in case of tie sort by last name (alphabetical), in case of tie sort by weight (descending), in case of tie sort by who has the birthday closest to June 30th.

- Pre-process the data: In some cases, you can modify parts of the data to make sorting simple. Example: "Sort closest to X" = sort "abs(X-value)";

- Crypt Kicker II
- File Fragmentation
- Vito's Family
- ~~Football (aka Soccer)~~ <- Don't do this one!
- Shoemaker's Problem