PFTWBV

- Sorting in theory and practice
  - Simple algorithms, more complex algorithms
  - Using the java.util library sorts
  - Which should you use in solving problems?
  - Personal sojourn related to Bubble Sort

- Using ideas from recitation to solve APTs
  - Recursion, Memoizing, Counting

How many search trees are there?

- How many four node trees? 14
  - Five trees with 3 nodes
  - Two for 2, one for 1
- What about six nodes?
  - Left subtree: 0,1,2,3,4,5
  - Right subtree: 5,4,3,2,1,0
- 8-node trees with 4 in left and 3 in right subtrees?
  - Why is this 14 x 5 = 60?
  - For each of 14, there are 5 different right subtrees

Spreading rumors and news

- Call B, takes 1 minute, done at T=6
  - Call A at T=1, done at T=6
  - Call B at T=2, done at T=7
- Greedy!
  - Biggest First

How high can you count?

- 1, 2, 3, 4, ...
  - How many values with 3 bits? 000, 001, 010, 011, 100, 101, 110, 111
  - In general, with N bits? $2^N$, so with 32 bits?
  - Half negative, half positive, what is $2^{31}-1$?
- The Java type long is 64 bits. $2^{63} - 1 = 9,223,372,036,854,775,807$
- What is 40!, it's $8.159 \times 10^{47}$
**Sorting in Theory and in Practice**

**Sorting: From Theory to Practice**

- **Will you use sorting in code you write?**
  - Yes, Maybe, No, It Depends?
  - You will in CompSci 201 for sure!!

- **Why do we study sorting?**
  - Elegant, practical, powerful, simple, complex
  - Everyone is doing it!
  - Example of algorithm analysis in a simple, useful setting

**Simple, O(n^2) sorts**

- **Selection sort** --- \( n^2 \) comparisons, \( n \) swaps
  - Find min, swap to front, bump front, repeat

- **Insertion sort** --- \( n^2 \) comparisons, no swap, shift
  - stable, fast on sorted data, slide into place

- **Bubble sort** --- \( n^2 \) everything, slow*
  - Catchy name, but slow and ugly*
  
  *this isn’t everyone’s opinion, but it should be

- **Shell sort**: quasi-insertion, fast in practice
  - Not quadratic with some tweaks

---

*Not Yogi*
More efficient O(n log n) sorts

- Divide and conquer sorts: O(n log n) for n elements
  - Quick sort: fast in practice, O(n²) worst case
  - Merge sort: good worst case, great for linked lists, uses extra storage for vectors/arrays

- Other sorts:
  - Heap sort, basically priority queue sorting
  - Radix sort: doesn't compare keys, uses digits/characters

- Cannot do better than O(n log n) when comparing!
  - Radix is O(n) ?!!!

Stable, Stability

- What does the search query 'stable sort' show us?
  - Image search explained
  - Why are numeric examples so popular?

Analyzing Sort Performance

- Let's look at overall structure, and algorithms
  - Try to see the trees, not the forest
  - Try to see the algorithm, not the syntax of Java

- Appreciate power of Java interfaces and generics
  - Syntax is not pretty, but language is powerful
  - YABE, yet another benchmarking example

Summary of simple sorts

- Selection sort has n swaps, good for “heavy” data
  - moving objects with lots of state, e.g., …
    - In C or C++ this is an issue, in Java everything is a pointer/reference, so swapping is fast since it's pointer assignment

- Insertion sort good on nearly sorted data, stable!
  - No swaps, shifts – good for cache performance
  - Used in Timsort current state-of-the-art sorting

- Bubble sort is overrated, but really easy to code!
  - Can be parallelized, but on one machine don't go near it (see quotes at end of slides)
Quicksort: fast in practice

- Invented in 1962 by Tony Hoare, didn’t understand recursion
  - Worst case is $O(n^2)$, but avoidable in nearly all cases, shuffle data, smart pivot, etc.

```
void quick(String[] a, int left, int right)
{
if (left < right) {
  int pivot = partition(a,left,right);
  quick(a,left,pivot-1);
  quick(a,pivot+1, right);
}
}
```

- Recurrence
  - $T(n) = 2T(n/2) + O(n)$
  - If partition "good"

Mergesort: stable, fast, aux storage

- Part of computing history vernacular
  - Using tapes to sort using external "memory"

```
void mergeSort(String[] a)
{
if (a.length > 1) {
  int half = a.length/2;
  String[] a1 = Arrays.copyOfRange(a,0,half);
  String[] a2 = Arrays.copyOfRange(a,half,a.length);
  mergeSort(a1);
  mergeSort(a2);
  mergeInto(a,a1,a2); // one from a1, one from a2,
}
```

- Recurrence always: $T(n) = 2T(n/2) + O(n)$
  - Uses extra storage (not for linked lists!)

Timsort: hybrid of merge and insertion?

- Stable, $O(n \log n)$ in average and worst, $O(n)$ best!
  - In practice lots of data is "close" to sorted
- Invented by Tim Peters for Python, now in Java
  - Replaced merge sort which is also stable
- Engineered to be correct, fast, useful in practice
  - Theory and explanation not so simple

https://en.wikipedia.org/wiki/Timsort

- Summary of $O(n \log n)$ sorts
  - Quicksort straight-forward to code, very fast
    - Worst case is very unlikely, but possible, therefore ...
    - But, if lots of elements are equal, performance will be bad
      - One million integers from range 0 to 10,000
      - How can we change partition to handle this?
  - Merge sort is stable, it’s fast, good for linked lists, harder to code?
    - Worst case performance is $O(n \log n)$, compare quicksort
    - Extra storage for array/vector
  - Timsort: hybrid of merge and insertion?
    - Really very fast in the wild/real world: Python, Java 7, Android
Running times @ $10^6$ instructions/sec

<table>
<thead>
<tr>
<th>N</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.000003</td>
<td>0.0001</td>
<td>0.000033</td>
<td>0.0001</td>
</tr>
<tr>
<td>100</td>
<td>0.000007</td>
<td>0.0010</td>
<td>0.000664</td>
<td>0.1000</td>
</tr>
<tr>
<td>1,000</td>
<td>0.000010</td>
<td>0.0100</td>
<td>0.010000</td>
<td>1.0</td>
</tr>
<tr>
<td>10,000</td>
<td>0.000013</td>
<td>0.0100</td>
<td>0.013900</td>
<td>1.7 min</td>
</tr>
<tr>
<td>100,000</td>
<td>0.000017</td>
<td>0.1000</td>
<td>1.661000</td>
<td>2.78 hr</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.000020</td>
<td>1.0</td>
<td>19.9</td>
<td>11.6 day</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.000030</td>
<td>16.7 min</td>
<td>18.3 hr</td>
<td>317 centuries</td>
</tr>
</tbody>
</table>

WOTO Questions


- What sorting algorithm will you use in writing code to solve problems?
  - Does it matter if you use Python or Java?

Brian Fox

- GNU Bash Shell (developer)
  - First employee at Free Software Foundation
- Developed first online banking system at Wells Fargo

Sorting in practice

- Rarely will you need to roll your own sort, but when you do...
  - What are key issues?
- If you use a library sort, you need to understand the interface
  - In C sort is complex to use because arrays are not easy to use, just pointers to memory
  - In Java guarantees and worst-case are important
    - Why won't quicksort be used?
- Comparators allow sorting criteria to change
How do we sort in Practice?

- You call Collections.sort or Arrays.sort
  - Or you call sort in Python or stable_sort in C++

- You must be sorting Comparable objects or ...
  - You can specify how to compare objects using java.util.Comparator

- Let's look at .compareTo: builtin and custom
  - String and Person

OpenJDK String.compareTo

```java
public int compareTo(String anotherString) {
    int len1 = value.length;
    int len2 = anotherString.value.length;
    int lim = Math.min(len1, len2);
    char v1[] = value;
    char v2[] = anotherString.value;
    int k = 0;
    while (k < lim) {
        char c1 = v1[k];
        char c2 = v2[k];
        if (c1 != c2) {
            return c1 - c2;
        }
        k++;
    } return len1 - len2;
}
```

Person class

https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/PersonSorter.java

- What interface does Person implement?
  - What type is parameter?
  - What value is returned?

- Alternative to adding interface to class
  - We can't always do this, no access to source
  - May want to sort more than one way

Comparator

- Doing this in Java 8 is reasonably easy
  - Use Comparator.comparing(...)
  - Specify method, and chain together as needed
  - Other syntax possible, way to complicated
  - Specify method for comparing method in class
    https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/PersonSorter.java

Alternative is creating a named class, rather than the class created by calling Comparator.comparing(…)
Comparator before and after Java 8

Alternative is creating a named class, rather than the class created by calling Comparator.comparing(...)

- Comparator Interface, one method, two parameters
  - .compare(T a, T b) { ... return integer }
  - See how to do this with Person class

- Libraries can be complicated to use, but it's a syntactic issue that engenders complexity

Bubble Sort, A Personal Odyssey

Steve and Rachel, Duke 1997

Steve and Rachel, Duke 1997
Jim Gray (Turing 1998)

- Bubble sort is a good argument for analyzing algorithm performance. It is a perfectly correct algorithm. But it's performance is among the worst imaginable. So, it crisply shows the difference between correct algorithms and good algorithms.

Brian Reid (Hopper Award 1982)

Feah. I love bubble sort, and I grow weary of people who have nothing better to do than to preach about it. Universities are good places to keep such people, so that they don't scare the general public.

(continued)

Brian Reid (Hopper 1982)

I am quite capable of squaring N with or without a calculator, and I know how long my sorts will bubble. I can type every form of bubble sort into a text editor from memory. If I am writing some quick code and I need a sort quick, as opposed to a quick sort, I just type in the bubble sort as if it were a statement. I'm done with it before I could look up the data type of the third argument to the quicksort library.

I have a dual-processor 1.2 GHz Powermac and it sneers at your N squared for most interesting values of N. And my source code is smaller than yours.

Brian Reid who keeps all of his bubbles sorted anyhow.
Niklaus Wirth (Turing award 1984)

I have read your article and share your view that Bubble Sort has hardly any merits. I think that it is so often mentioned, because it illustrates quite well the principle of sorting by exchanging.

I think BS is popular, because it fits well into a systematic development of sorting algorithms. But it plays no role in actual applications.

Quite in contrast to C, also without merit (and its derivative Java), among programming codes.

Guy L. Steele, Jr. (Hopper ’88)

(Thank you for your fascinating paper and inquiry. Here are some off-the-cuff thoughts on the subject.)

I think that one reason for the popularity of Bubble Sort is that it is easy to see why it works, and the idea is simple enough that one can carry it around in one’s head …

continued

Guy L. Steele, Jr.

As for its status today, it may be an example of that phenomenon whereby the first widely popular version of something becomes frozen as a common term or cultural icon. Even in the 1990s, a comic-strip bathtub very likely sits off the floor on claw feet.

… it is the first thing that leaps to mind, the thing that is easy to recognize, the thing that is easy to doodle on a napkin, when one thinks generically or popularly about sort routines.

Sorting Conundrums

- You want to sort a million 32-bit integers
  - You’re an advisor to Obama
Debugging can be frustrating, but very rewarding when you find the cause of the problem. One of the nastiest bugs that I’ve found when I was at NASA and the Mars Explorer Rovers had just landed on Mars.

http://sortbenchmark.org/
Hadoop sets Terabyte sort record
- Java
- 34252 nodes
- Greedy!