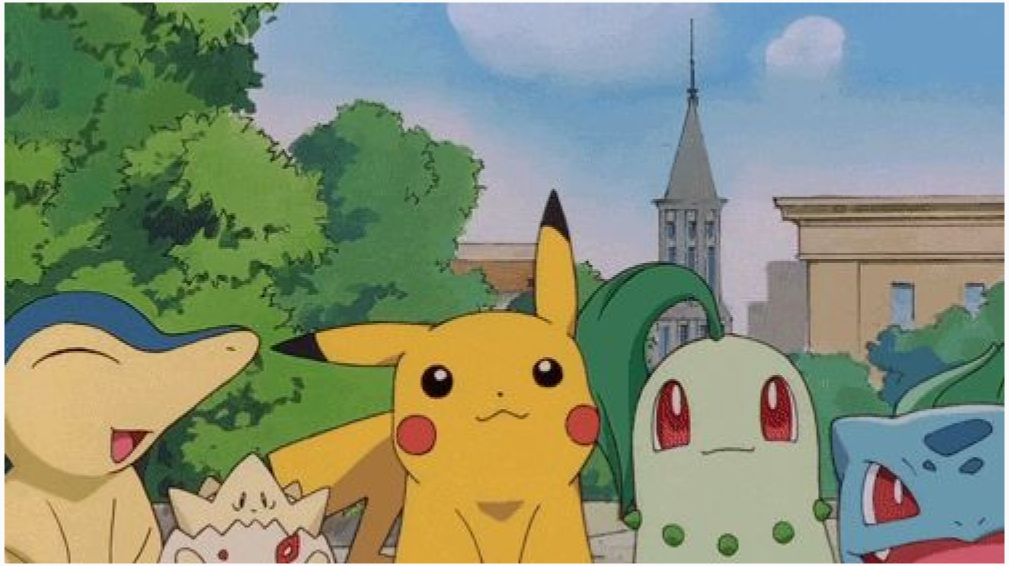
Sorting, Bounds, and Comparing Algorithms

CS 61B Spring 2016: Discussion 13

Announcements

This is our last real section covering material!



Announcements

Next section is “Goodbye, Fun”.

My favorite section of the semester.

We will say goodbye. It will be fun.

We will take it easy, maybe talk about 61B material, probably talk about life after 61B, maybe I will talk about an interesting topic outside of the scope of 61B...

I will probably bring some treats.

I will probably wear a funny hat.

You should come!

Announcements

Project 3 was due 14 hours ago!

But only 2% penalty if you turn it in by 11:59pm tonight.

Tangent: Informal feedback on Project 3? Thoughts? Complaints? Praise?

HW5 coming soon and due Mon, 4/25.

Announcements

Just two weeks left of material to cover (including this one)!

Remaining material:
Radix Sort, Tries, Compression, Reductions, Algorithmic Bounds, and NP-completeness

Sorting

Listed by mechanism:

- Selection sort: Find the smallest item and put it at the front.
- Insertion sort: Figure out where to insert the current item.
- Merge sort: Merge two sorted halves into one sorted whole.
- Partition (quick) sort: Partition items around a pivot.

Listed by memory and runtime:

	Memory	Time	Notes
Heapsort	Θ(1)	Θ(N log N)	Bad caching (61C)
Insertion	Θ(1)	$\Theta(N^2)$	Θ(N) if almost sorted
Mergesort	Θ(N)	Θ(N log N)	
Random Quicksort	Θ(log N) (call stack)	Θ(N log N) expected	Fastest sort

QuickSort Review

A sorting algorithm is **stable** if equal elements retain their relative ordering after the sort.

5a 2 **5b** 3 4 ---> 2 3 4 **5a** **5b**

A **stable** quicksort uses 3 partitions (less, equal, greater), as we saw in lab. An **unstable** quicksort uses an in-place partitioning scheme such as Hoare Partitioning.

QuickSort Review

Unstable quicksort is faster than **stable** quicksort by some constant factor.

Intuition: A **stable** quicksort requires more moving around of data. **Unstable** quicksort has pointers and scans around, doing a swap only when necessary. A **stable** one will need to move some nK copies around for every partition step.

Unstable quicksort is also faster than mergesort (**also stable**); similar intuition.

QuickSort Review

Unstable quicksort is faster than **stable** quicksort by some constant factor.

Intuition: A **stable** quicksort requires more moving around of data. **Unstable** quicksort has pointers and scans around, doing a swap only when necessary. A **stable** one will need to move some nK copies around for every partition step.

Unstable quicksort is also faster than mergesort (**also stable**); similar intuition.

Live (Stable) Quicksort!

Volunteers?

Quicksort

`Problem 1`

There is a typo on worksheet: it should be **stable** quicksort, not **in-place** quicksort.

Problem 1a

```
 18, 7, 22, 34, 99, 18, 11, 4
-18-, 7, 22, 34, 99, 18, 11, 4
-7-, 11, 4 | 18, 18 | 22, 34, 99
4, 7, 11, 18, 18 | -22-, 34, 99
4, 7, 11, 18, 18, 22 | -34-, 99
4, 7, 11, 18, 18, 22, 34, 99
```

Problem 1b

$O(n^2)$.

Running quicksort on a sorted list will take $O(n^2)$ if the pivot chosen is always the first or last in the subarray.

In general, the worst case is such that the partitioning scheme repeatedly partitions an array into one element and the rest. At each level of recursion, you will need to do $O(n)$ work, and there will be $O(n)$ levels of recursion: $O(n) * O(n) = O(n^2)$.

Problem 1c

Theta(nlogn).

The optimal case for quicksort occurs if you can choose a pivot such that the left partition and right partition are of equal sizes. At each level of recursion, you will need to do O(n) work, and there will be Omega(logn) levels of recursion.

If you could, you'd violate the sorting lower bound from class on Monday (where we showed that sorting solves the "puppy, cat, dog" problem).

Problem 1d

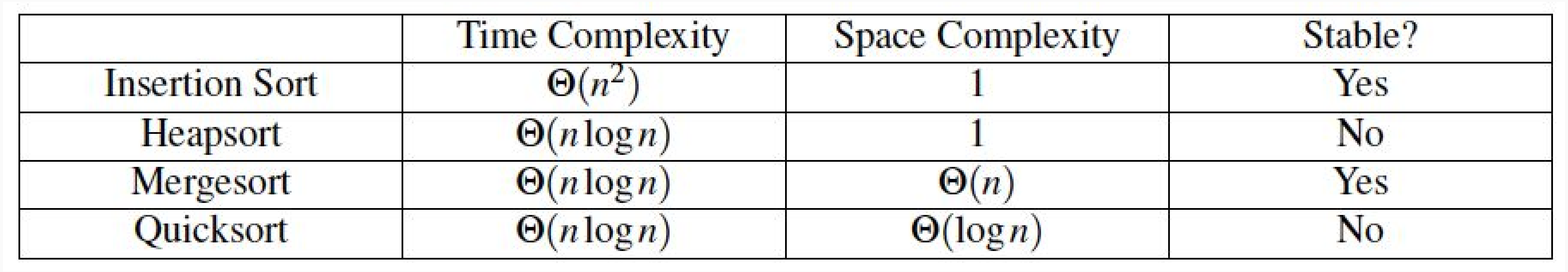
What are two techniques that can be used to reduce the probability of quicksort taking the worst case running time (N^2)?

1. Randomly choose pivots.
2. Shuffle list before running quicksort.

Comparing Sorting Algorithms

Problem 2

Problem 2



	Time Complexity	Space Complexity	Stable?
Insertion Sort	$\Theta(n^2)$	1	Yes
Heapsort	$\Theta(n \log n)$	1	No
Mergesort	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n \log n)$	$\Theta(\log n)$	No

Notes:

- Depends on implementation. These are expected worst case.
- Mergesort uses an auxiliary array for merging, which takes N space.
- The stack trace used for quicksort contributes logN space. But it's nearly constant.

Problem 2a

Unstable sorts we've covered:

Heapsort, Quicksort (Hoare/in-place partitioning or randomized)

Heapsort example: 1a, 1b, 1c

Quicksort example: 1, 5a, 2, 5b, 3

Side note: If we are using randomized quicksort (like we should be), any array could work as an example.

Problem 2b

Tradeoffs when designing an algorithm:

- Readability when other engineers use your algorithm
- Constant factors in runtime (these matter a lot, especially with small inputs!)

Bounding Practice

Problem 3

Problem 3a

Why can we say that any solution for heapification requires Omega(n) time?

In order to check that an array satisfies the heap invariant, we have to **at least** look at every element, which takes linear time.

Problem 3b

Give the worst-case runtime for top-down heapification in Theta notation. Why does this mean that the optimal solution for heapification takes O(nlogn) time?

Worst-case runtime for top-down heapification is Theta(NlogN). Therefore, the optimal solution for heapification is O(NlogN). In other words, we could do better than NlogN in some cases (e.g. bottom-up heapification), but we can always default to top-down for a NlogN solution.

Final Words

Next section is “Goodbye, Fun”.

My favorite section of the semester.

We will say goodbye. It will be fun.

We will take it easy, maybe talk about 61B material, probably talk about life after 61B, maybe I will talk about an interesting topic outside of the scope of 61B...

I will probably bring some treats.

I will probably wear a funny hat.

You should come!

Final Words

Vote for interesting topics:

- How does the Internet work? (Networking, routing)
- Computer graphics demos (and how data structures are used in CG)
    - Maybe show what progress I've made on my final project for my graphics class if I've started it by then...?
- Data structures we didn't cover in 61B
- More theory... CS170 stuff
- More lower level stuff... caches (61C)?
- How does the latest Virtual Reality technology work?
- How to play mid lane on League of Legends

Thanks for coming!

`See you all next week!`