CSM 61B Spring 2017

Sorting Algorithms

Week 13: April 18, 2017

1 Asymptotics Potpourri

Algorithm	Best-case	Worst-case	Stable
Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	Depends
Insertion Sort	$\Theta(N)$	$\Theta(N^2)$	Yes
Merge Sort	$\Theta(N \log N)$	$\Theta(N \log N)$	Yes
Quicksort	$\Theta(N)$	$\Theta(N^2)$	Depends
Heapsort	$\Theta(N \log N)$	$\Theta(N \log N)$	Hard

2 Vertigo

2.1 We have a list of N elements that should be sorted, but to our surprise we recently discovered that there are at most k pairs out of order, or k **inversions**, in the list. As a small example, the list `{ 0, 1, 2, 6, 4, 5, 3 }` contains 5 inversions: $(6,4), (6,5), (6,3), (4,3), (5,3)$.

For each value of k below, state the most efficient sorting algorithm and give a tight asymptotic runtime bound.

(a) $k \in O(\log N)$

Insertion sort is the most efficient in this case because its runtime is $O(N + k)$. The overall runtime bound for insertion sort in this scenario is $O(N)$.

(b) $k \in O(N)$

Insertion sort for the same reason above. The overall runtime bound for insertion sort in this scenario is $O(N)$.

(c) $k \in O(N^2)$

Merge sort, quicksort, or heap sort would be ideal here since the number of inversions causes insertion sort to run in $O(N^2)$ runtime. Using one of the three sorts listed earlier yields a runtime in $O(N \log N)$ in the normal case.

3 QuickCo

Malicious Mallory, a sinister hacker, has been hired by a competitor to break into QuickCo, the world leader in sorting algorithms, and tamper with its famous Quicksort algorithms by making them as slow as possible. Mallory succeeded in unlocking the mainframe, but now she needs your help in slowing QuickCo's algorithms down to a halt!

```
int[] data = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

3.1 First, Mallory decides to change the way QuickCo chooses a pivot for Quicksort.

Given the `int[] data`, what choice of pivot would cause the worst-case runtime for Quicksort?

First, last, or alternating first-last.

3.2 Mallory finds an algorithm which always selects the middle element but she is unable to gain write access to it. However, she discovers a way to intercept the incoming data and rearrange it before the algorithm runs.

Given the `int[] data`, rearrange the numbers such that the algorithm will run in its worst-case time.

```
{ 6, 7, 8, 9, 1, 2, 3, 4, 5 }
```

3.3 Does the worst-case runtime of Quicksort depend on the array order, pivot choice, or both? Why?

The worst-case runtime of Quicksort depends on both the array order and the choice of pivots. The worst-case always occurs when the pivot's final position is on an end of the array, which means it was either the smallest or the largest element.

4 Pivotal Choice

4.1 For each pivot selection strategy below, what is the best, average and worst case runtime?

The best case and average case runtime for quicksort in each scenario is still in $\Theta(N \log N)$.

(a) Always choose the first value in the list.

If we always select the first value as a pivot, the values in the list will determine the runtime. If we have a sorted or reverse sorted array, selecting the first value will only reduce the problem size by a single value during each call: we essentially have selection sort! The runtime in this worst-case scenario is $\Theta(N^2)$.

But if the values in the list are randomly shuffled, then choosing the first value not degrade into the worst-case runtime.

(b) Always find and choose the median value in the list. Assume finding the median takes $O(N)$ time where N is the length of the list.

The worst case runtime in this scenario will be in $\Theta(N \log N)$ including the time to find the median (which can be found in linear time using the median of medians algorithm). In reality, the additional cost associated with finding the median is usually not worth it over simply selecting a random value.

Even though median-finding takes linear time, quicksort normally needs to spend linear time bucketing values into less-than, equal-to, and greater-than buckets anyways so the asymptotic runtime is not affected.

(c) Always choose a random pivot.

It is possible for this to degrade to $\Theta(N^2)$ runtime in the worst case as well. If we pick a random pivot from an array we have a $\frac{1}{N}$ probability of selecting the smallest value and, after that, a $\frac{1}{N-1}$ of selecting the next smallest, and so forth. The total probability of selecting the smallest value every time is $\frac{1}{N!}$.

Although it is possible for for quicksort to run in $\Theta(N^2)$, it becomes less and less probable as N, or the size of the list, increases.

5 Showdown *Extra for Experts*

5.1 (a) What are the advantages and disadvantages of quicksort?

Advantages:

- Faster on average by a constant factor than merge sort
- In-place variant for $\Theta(1)$ memory complexity. Including the average-case call stack brings up space complexity to $\Theta(\log N)$
- Multi-pivot quicksort offers greater constant factor optimizations

Disadvantages:

- Most efficient implementations (in-place) are not stable
- Worst-case runtime is in $\Theta(N^2)$, thus making it a poor choice for adversarial datasets
- Not very good for external sorting

(b) What are the advantages and disadvantages of merge sort?

Advantages:

- Stable
- Good for external sorting
- Constant space usage for sorting linked lists

Disadvantages:

- Slower than quicksort on average
- Higher space complexity for array allocation