Data Structures and Algorithms

Session 25. April 27th, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 5 solutions on courseworks
- Homework 6 due before last class: May 4th
- Final Review May 4th
- Exam Wednesday May 13th 1:10-4:00 PM, 633

Review

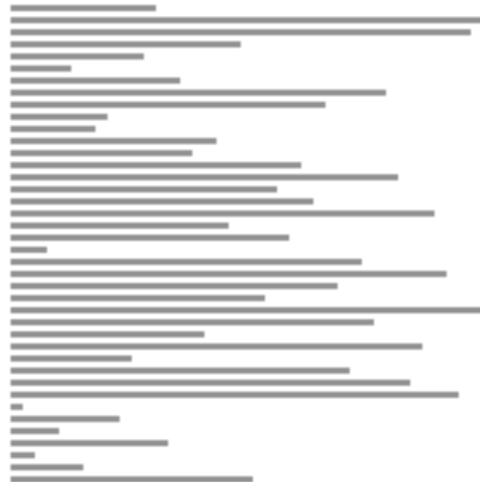
- Radix Sort specifics
- Comparison sorting algorithm characteristics
- Algorithms: Selection Sort, Insertion Sort, Shellsort, Heapsort, Mergesort, Quicksort

Today’s Plan

- Finish Quicksort discussion,
  - worst case, average case
- Quickselect
  - worst case, average case
- External Sorting

Quicksort

- Choose an element as the **pivot**
- Partition the array into elements greater than pivot and elements less than pivot
- Quicksort each partition



Choosing a Pivot

- The worst case for Quicksort is when the partitions are of size zero and **N-1**
- Ideally, the pivot is the median, so each partition is about half
- If your input is random, you can choose the first element, but this is very bad for presorted input!
- Choosing randomly works, but a better method is…

Median-of-Three

* Choose three entries, use the median as pivot
* If we choose randomly, **2/N** probability of worst case pivots
* Median-of-three gives **0** probability of worst case, tiny probability of 2nd-worst case. (Approx. $2/N^3$)
* Randomness less important, so choosing (first, middle, last) works reasonably well

Partitioning the Array

- Once pivot is chosen, swap pivot to end of array. Start counters **i**=1 and **j**=**N-1**
- Intuition: **i** will look at less-than partition, **j** will look at greater-than partition
- Increment **i** and decrement **j** until we find elements that don't belong (**A[i]** > **pivot** or **A[j]** < **pivot**)
- Swap (**A[i]**, **A[j]**), continue increment/decrements
- When **i** and **j** touch, swap pivot with **A[j]**

Quicksort Worst Case

- Running time recurrence includes the cost of partitioning, then the cost of 2 quicksorts
- We don't know the size of the partitions, so let **i** be the size of the first partition
- $\mathbf{T(N) = T(i)+T(N-i-1) + N}$
- Worst case is $\mathbf{T(N) = T(N-1) + N}$

Quicksort Average Case

- We'll average over all partition sizes:

$$T(N) = \frac{2}{N} \sum_{i=1}^{N-1} T(i) + N$$

$$NT(N) = 2 \sum_{i=0}^{N-1} T(i) + N^2$$

$$(N-1)T(N-1) = 2 \sum_{i=0}^{N-2} T(i) + (N-1)^2$$

Quicksort Average Case

$$NT(N) = 2\sum_{i=0}^{N-1} T(i) + N^2$$

$$(N-1)T(N-1) = 2\sum_{i=0}^{N-2} T(i) + (N-1)^2$$

$$\begin{aligned} NT(N) - (N-1)T(N-1) &= 2\left[\sum_{i=0}^{N-1} T(i) - \sum_{i=0}^{N-2} T(i)\right] \\ &\quad + N^2 - (N-1)^2 \end{aligned}$$

Quicksort Average Case

$$NT(N) - (N-1)T(N-1) = 2\left[\sum_{i=0}^{N-1} T(i) - \sum_{i=0}^{N-2} T(i)\right] + N^2 - (N-1)^2$$

$$NT(N) - (N-1)T(N-1) = 2T(N-1) + 2N - 1$$

$$NT(N) = (N+1)T(N-1) + 2N$$

$$\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2}{N+1}$$

Quicksort Average Case

$$\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2}{N+1}$$

$$\frac{T(N-1)}{N} = \frac{T(N-2)}{N-1} + \frac{2}{N}$$

$$\frac{T(N-2)}{N-1} = \frac{T(N-3)}{N-2} + \frac{2}{N-1}$$

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2}{3}$$

$$\frac{T(N)}{N+1} = \frac{T(1)}{2} + 2\sum_{i=3}^{N+1} \frac{1}{i}$$

$$\frac{T(N)}{N+1} = O(\log N)$$

$$T(N) = O(N \log N)$$

Quicksort Properties

- Unstable
- Average time O(N log N)
- Worst case time $O(N^2)$
- Space O(log N)/$O(N^2)$ because we need to store the pivots

Sorting Algorithm Summary

	Worst Case Time	Average Time	Space	Stable?
Selection	$O(N^2)$	$O(N^2)$	$O(1)$	No
Insertion	$O(N^2)$	$O(N^2)$	$O(1)$	Yes
Shell	$O(N^{3/2})$	?	$O(1)$	No
Heap	$O(N \log N)$	$O(N \log N)$	$O(1)$	No
Merge	$O(N \log N)$	$O(N \log N)$	$O(N)/O(1)$	Yes/No
Quick	$O(N^2)$	$O(N \log N)$	$O(\log N)$	No

Selection

- Recall selection problem: best solution so far was Heapselect
    - Running time: **O(N+k log N)**
- We should expect a faster algorithm since selection should be easier than sorting

Quickselect

- Choose a pivot, partition array, recurse on the partition that contains **k**’th element
- e.g., select 3rd element

81	91	13	16	97	96	49	80	14	42
42	14	13	16	80	49	**81**	97	91	96
16	14	13	**42**	80	49	81	97	91	96
13	14	**16**	42	80	49	81	97	91	96

Quickselect Worst Case

- Quickselect only recurses one one of the subproblems
- However, in the worst case, pivot only eliminates one element:
  - **$T(N) = T(N-1) + N$**
- Same as Quicksort worst case

Quickselect Average Case

- Assume pivot is randomly selected; equal probability for each subproblem size

$$T(N) = \frac{1}{N}\sum_{i=0}^{N-1} T(i) + N$$

$$NT(N) = \sum_{i=0}^{N-1} T(i) + N^2$$

$$(N-1)T(N-1) = \sum_{i=0}^{N-2} T(i) + (N-1)^2$$

$$NT(N) - (N-1)T(N-1) = T(N-1) + N^2 - (N-1)^2$$

Quickselect Average Case

$$NT(N) - (N-1)T(N-1) = T(N-1) + N^2 - (N-1)^2$$

$$NT(N) - NT(N-1) + T(N-1) = T(N-1) + \ldots$$

$$NT(N) = NT(N-1) + N^2 - (N-1)^2$$

$$NT(N) = NT(N-1) + 2N - 1$$

$$T(N) \leq T(N-1) + 2$$

$$T(N) = O(N)$$

External Sorting

- So far, we have looked at sorting algorithms when the data is all available in RAM
- Often, the data we want to sort is so large, we can only fit a subset in RAM at any time
- We could run standard sorting algorithms, but then we would be swapping elements to and from disk
  - Instead, we want to minimize disk I/O, even if it means more CPU work

MergeSort

- We can speed up external sorting if we have two or more disks (with free space) via Mergesort
- One nice feature of Mergesort is the merging step can be done online with streaming data
- Read as much data as you can, sort, write to disk, repeat for all data, write output to alternating disks
  - merge outputs using 4 disks

Simplified Running Time Analysis

- Suppose random disk i/o cost 10,000 ns
  - Sequential disk i/o cost 100 ns
  - RAM swaps/comparisons cost 10 ns
- Naive sorting: 10000 N log N
- Assume **M** elements fit in RAM.
  External mergesort:
  10 N log M + 100 N (# of sweeps through data)

Counting Merges

- After initial sorting, **N/M** sorted subsets distributed between 2 disks
- After each run, each pair is merged into a sorted subset twice as large.
  - Full data set is sorted after **log(N/M)** runs
- External sorting:
  $10 N \log M + 100 N \log (N/M)$

Next Class

- Data structures for Machine Learning/Artificial Intelligence
  - Not on exam
- Start review

Reading

- http://www.sorting-algorithms.com/
- Weiss Chapter 7