CS 61B — Spring 2021

More Sorting

Discussion 13: April 19, 2021

1 Quicksort

(a) Sort the following unordered list using stable Quicksort. Assume that we always choose first element as the pivot and that we use the 3-way merge partitioning process described in lecture. Show the steps taken at each partitioning step.

`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

(b) What is the best and worst case running time of Quicksort with Hoare Partitioning on N elements? Given the two lists [4, 4, 4, 4, 4] and [1, 2, 3, 4, 5], assuming we pick the first element as the pivot every time, which list would happen to result in better runtime?

Best: $\Theta(N\log N)$ [4, 4, 4, 4, 4]

Worst: $\Theta(N^2)$

(c) What are two techniques that can be used to reduce the probability of Quicksort taking the worst case running time?

随机选取锚点、

排序前洗混数组

2 Comparison Sorts Summary

(a) When choosing an appropriate algorithm, there are often several trade-offs that we need to consider. Complete the chart for the following sorting algorithms: give the expected time complexity in the worst case, in the best case, and whether or not each sort is stable.

	Time Complexity (Best)	Time Complexity (Worst)	Stability	In Place
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	✗	✓
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	✓	✓
Heapsort	$\Theta(n)$	$\Theta(n\log n)$	✗	✓
Mergesort	$\Theta(n\log n)$	$\Theta(n\log n)$	✓	✗(一般情况)
Quicksort (w/ Hoare Partitioning)	$\Theta(n\log n)$	$\Theta(n^2)$	✗	大部分是

(b) For selection sort, give an example of a list where the order of equivalent items is not preserved.

选择. 3 3 3 1

堆 1 1 1

快排. 3 5 2 5 1

(c) Notice that the worst-case runtime in the comparison sorts on an N element array listed above are lower bounded by $\Theta(N \log N)$. Can there be a sort that runs faster than $\Theta(N \log N)$ in the worst-case?

N个数. N!种排列.

使用比较式排序.需要至少 $\log_2(N!) \in \Omega(N\log N)$ 个比较.

使用非比较排序.可以.

3 Radix Sorts

(a) Sort the following list using LSD Radix Sort with counting sort. Show the steps taken after each round of counting sort. The first row is the original list and the last two rounds are already filled for you.

	30395	30326	43092	30315
1	43092	30395	30315	30326
2	30315	30326	30395.	43092.
3	43092	30315	30326	30395.
4	30315	30326	30395	43092
5	30315	30326	30395	43092

(b) Sort the following list using MSD Radix Sort with counting sort. Show the steps taken after each round of counting sort. The first row is the original list and the first three rounds are already filled for you.

	30395	30326	43092	30315
1	<u>3</u>0395	<u>3</u>0326	<u>3</u>0315	| <u>4</u>0392
2	<u>30</u>395	<u>30</u>326	<u>30</u>315	| <u>4</u>0392
3	<u>303</u>95	<u>303</u>26	<u>303</u>15	| <u>4</u>0392
4	| 30315	| 30326	| 30395	| 40392
5	| 30315	| 30326	| 30395	| 40392.

(c) Give the best case runtime, worst case runtime, and whether or not the sort is stable for both LSD and MSD radix sort. Assume we have N elements, a radix R, and a maximum number of digits in an element W.

	Time Complexity (Best)	Time Complexity (Worst)	Stability
LSD Radix Sort	$\Theta(W(N+R))$	$\Theta(W(N+R))$	✓
MSD Radix Sort	$\Theta(N+R)$	$\Theta(W(N+R))$	✓

(d) We just saw above that radix sort has great runtime with respect to the number of elements in the list. Given this fact, should we say that radix sort is the best sort to use?

只能用于可分离组的排序。W, R 可以很大。

4 Bounding Practice *Extra*

Given an array of n elements, the heapification operation permutes the elements of the array into a heap. There are many solutions to the heapification problem. One approach is bottom-up heapification, which treats the existing array as a heap and rearranges all nodes from the bottom up to satisfy the heap invariant. Another is top-down heapification, which starts with an empty heap and inserts all elements into it.

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

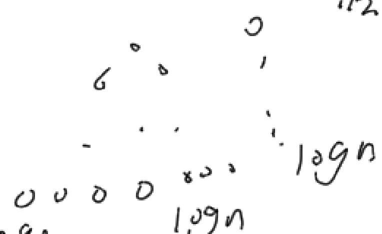
至少要check每一个元素

(b) Show that the worst-case runtime for top-down heapification is in $\Theta(n \log n)$. Why does this mean that the optimal solution for heapification takes $O(n \log n)$ time?

i层上升i次

最差每步 logN 共N个元素 $\Theta(N\log N)$

O描述上界



$\sum_{i=0}^{\log n} 2^i \cdot i \le \log_2 n \sum_{i=0}^{\log n} 2^i$ $\quad \sum_{i=0}^{\log n} 2^i \in \Theta(n)$

$\to \Theta(n\log n)$

(c) In contrast, bottom-up heapification is an $O(n)$ algorithm. Is bottom-up heapfication asymptotically-optimal?

✓

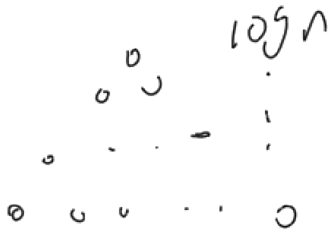
(d) Show that the running time of bottom-up heapify is $\Theta(n)$.

Some useful facts:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Taking the derivative:

$$\frac{d}{dx}\left(\sum_{i=0}^{\infty} x^i\right) = \frac{1}{(1-x)^2}$$



i层下降i次

$\sum_{i=0}^{\log n} i \cdot 2^{\log(n)-i-1}$

$= \sum_{i=0}^{\log n} i \frac{n}{2^{i+1}}$

$= \frac{n}{4} \sum_{i=0}^{\log n} i\left(\frac{1}{2}\right)^{i-1}$

$\frac{n}{4} \sum_{i=0}^{\log n} i x^{i-1} \le \frac{n}{4} \sum_{i=0}^{\infty} i x^{i-1}$

$= \frac{n}{4} \frac{d}{dx} \sum_{i=0}^{\infty} x^i$

$= \frac{n}{4} \frac{1}{(1-x)^2}$

$= \frac{n}{4} \frac{1}{(1-\frac{1}{2})^2}$

$= \Theta(n)$