CSCI 3104 Spring 2018

Problem Set 4

Merola, Michael

06/04/1998

Problem Set 4

Problem 1

(10 pts) Suppose that we modify the Partition algorithm in QuickSort in such a way that on on every third level of the recursion tree it chooses the worst possible pivot, and on all other levels of the recursion tree Partition chooses the best possible pivot. Write down a recurrence relation for this version of QuickSort and give its asymptotic solution. Then, give a verbal explanation of how this Partition algorithm changes the running time of QuickSort.

QuickSort **Best-Case** Recurrence: $2T(\frac{N}{2}) + (N-1)$

QuickSort **Worst-Case** Recurrence: $T(N-1) + (N-1)$

Unrolling

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

$= 2T(\frac{N}{4}) + (N-1) + (N-2)$

$= 2T(\frac{N}{4}) + (N-1) + (N-2) + (N-3)$

$= 2T(\frac{((N-1))}{4}) + (N-1) + (N-2) + (N-3)$ ...plug in worst-case ((N-1)) at third level

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

$= \theta(nlog(n))$

The final complexity for this version of quicksort is the same as the best-case for the original algorithm [$\theta(nlog(n))$]. Due to this, the algorithm eventually nullifies the effect of third-level worst case, and it doen't have any effect on the runtime of the algorithm.

Problem 2

(10 pts) Mr. Ollivander, of Ollivanders wand shop, has hired you as his assistant, to find the most powerful wand in the store. You are given a magical scale which “weighs” wands by how powerful they are (the scale dips lower for the wand which is more powerful). You are given n wands W1, . . . , Wn, each having distinct levels of power (no two are exactly equal).

(a) Consider the following algorithm to find the most powerful wand:

```
i. Divide the n wands into n/2 pairs of wands.
ii. Compare each wand with its pair, and retain the more powerful of the tw
o.
iii. Repeat this process until just one wand remains.
```

Illustrate the comparisons that the algorithm will do for the following n = 8 input:

$W1: \frac{2}{3}, W2: \frac{5}{2}, W3: \frac{1}{2}, W4: 1, W5: 2, W6: \frac{5}{4}, W7: \frac{1}{4}, W8: \frac{9}{4}$

=> $W1: \frac{2}{3} ?? W2: \frac{5}{2} ||| W3: \frac{1}{2} ?? W4: 1 ||| W5: 2 ?? W6: \frac{5}{4} ||| W7: \frac{1}{4} ?? W8: \frac{9}{4}$

=> $W2: \frac{5}{2} ?? W4: 1 ||| W5: 2 ?? W8: \frac{9}{4}$

=> $W2: \frac{5}{2} ?? W8: \frac{9}{4}$

=> $W2: \frac{5}{2}$ is the most powerful wand

(b) Show that for n wands, the algorithm (2a) uses at most n comparisons.

$T(n) = T(\frac{n}{2}) + (\frac{n}{2})$

*divide array in half each time, perform n/2 operations

$= T(\frac{n}{4}) + \frac{n}{4} + \frac{n}{2}$

$= T(\frac{n}{6}) + \frac{n}{4} + \frac{n}{2} + \frac{n}{2}$

$= \ldots$

$\sum_{i=1}^{logn} \frac{n}{2^i} = n - 1 = O(n)$

(c) Describe an algorithm that uses the results of (2a) to find the second most powerful wand, using at most log(n) additional comparisons. There is no need for pseudocode; just write out the steps of the algorithm like we have written in (2a). Hint: if you follow sports, especially wrestling, read about the repechage.

```
i.   Divide the n wands into n/2 pairs of wands.
ii.  Compare each wand with its pair, and retain the more powerful of the tw
o.
     Track the winners/losers of each pair
iii. Repeat this process until just one wand remains.
iv.  Return an array of [wands that lost to the ultimate winner] to the func
tion
v.   Function returns the 2nd-most powerful wand out of original array.
```

(d) Show the additional comparisons that your algorithm in (2c) will perform for the input given in (2a).

...

=> $W2 : \frac{5}{2}$ is the most powerful wand

=> $W8, W4, and W1$ lost to $W2$ during the initial run-thru

=> $f([W8, W4, W1])$

=> $W8 : \frac{9}{4}??W4 : 1|||W1 : \frac{2}{3}$

=> $W8 : \frac{9}{4}??W1 : \frac{2}{3}$

=> $W8 : \frac{9}{4}$ is the 2nd most powerful wand

Problem 3

(20 pts) For obtuse historical reasons, Prof. Dumbledore asks his students to line up in ascending order by height in a very tight room with little extra space. Similar to Alex the African Grey parrot (look it up!), the students, being bored, decided to play a little trick on Prof. Dumbledore. They lined up in order by height—almost. They made sure that each person was no more than k positions away from where they were supposed to be (in ascending order), but this allowed them to significantly mess up the precise ordering. Here is an example of an array with this property when k = 2:

(a) Write down pseudocode for an algorithm that would sort such an array in place— so it fits in the tight room— in time O(n k log k). Your algorithm can use a function sort(A,l,r) that sorts the subarray A[l], . . . , A[r] in place in O((r −l) log(r −l)) steps (assuming r > l).

In [7]:
```
def sortMessy(A,k):
    for i in range(0, len(a)):
        sort(A,i,i+k)
```

(b) Suppose you are given to an auxiliary room which can fit k + 1 students. Modify your previous algorithm to sort the given array in time O(nk).

In [8]:
```
print("I don't know")
```
```
I don't know
```

(c) With the same extra room as in the previous part, modify heap sort using a binary min heap of size k + 1 to sort the given array in time O(n log k).

```
In [9]: print("I don't know")

        I don't know
```

(d) (5 pts extra credit) Include the correct story about Alex, with proper citation. If you wish, you may copy this story verbatim, but must indicate clearly that you have done so and, of course, still cite your source.

[Cited from ModernNotion.com, 'Alex, the Only Parrot to Ask an Existential Question']

[Copied Verbatim. URL to full story in 'Sources' section.]

Alex, an African Gray parrot, is so far the only non-human animal to ever ask an existential question about himself, which means he was self-aware enough to understand his own existence.

Alex was owned and used by researcher Irene Pepperberg, an animal psychologist, to better understand how parrots can learn from and mimic humans. However, as time went on, Pepperberg realized that Alex wasn’t just reciting her actions and words, he was learning and thinking for himself. For example, when Pepperberg would ask what color corn is, even though there was no corn visible, Alex would answer yellow.

Since free thought is often cited as the biggest difference between animals and humans, researchers have pursued an understanding of animal intelligence since at least the 1600s. According to The New York Times:

The question of animal intelligence goes back at least to Descartes and his famous aphorism, ”I think, therefore I am.” Animals cannot think, said Descartes, and therefore are inferior to humans. And for many theologians and philosophers, the ability to think gives man a unique closeness to God.

Unlike other animals such as dolphins and chimpanzees who can answer questions, Alex is so far the only one to ever ask one about himself. One day, while learning colors, Alex looked into the mirror and asked, “What color?” A research assistant then told Alex that he was a gray parrot. After repeating the question six times, Alex learned the color gray. Even though it seems like he was just trying to figure out what gray was, the question, if thought about in terms of Descartes’ quote, shows that Alex was also thinking about himself and his own existence as a living thing.

Besides being the first animal to fully show that he knew of his own existence, Alex also expressed opinions and asked many other questions such as what a carrot was and where Pepperberg was going.

Alex sadly died in 2007 at the age of 31. His last words were to Pepperberg the night before when he said, “You be good, see you tomorrow. I love you.” Pepperberg has since started training other parrots, none of which are as smart as Alex was.

Problem 4

(20 pts) Consider the following strategy for choosing a pivot element for the Partition subroutine of QuickSort, applied to an array A.

```
• Let n be the number of elements of the array A.
• If n ≤ 24, perform an Insertion Sort of A and return.
• Otherwise:

    - Choose 2bn
    (1/3)c elements at random from n; let S be the new list with the
    chosen elements.
    - Sort the list S using Insertion Sort and use the median m of S as a pi
vot
    element.
    - Partition using m as a pivot.
    - Carry out QuickSort recursively on the two parts.
```

(a) How much time does it take to sort S and find its median? Give a Θ bound.

insertion sort running-time: $O(n^2)$

$$O((2n^{\frac{1}{3}})^2) = O(4n^{\frac{2}{3}})$$

$$= \theta(n^{\frac{2}{3}})$$

(b) If the element m obtained as the median of S is used as the pivot, what can we say about the sizes of the two partitions of the array A?

The median pivot obtained from subarray S will never be an extreme min or max value from A because it chooses the median from a subarray of random numbers. Due to this, the partitions of the array will never be empty because at least one element will exist in each partition at all times.

(c) Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.

account for original runtime and insertion sort runtime:

$$\theta(n^{\frac{2}{3}}) + \theta(n)$$

account for recursive calls

$$T(n - 2n^{\frac{1}{3}}) + T(2n^{\frac{1}{3}})$$

so...

$$T(n - 2n^{\frac{1}{3}}) + T(2n^{\frac{1}{3}}) + \theta(n^{\frac{2}{3}}) + \theta(n)$$

Problem 5 (Extra-Credit)

(20 pts extra credit) Recall that the Insertion Sort algorithm (Chapter 2.1 of CLRS) is an in-place sorting algorithm that takes Θ(n 2) time and Θ(n) space. In this problem, you will learn how to instrument your code and how to perform a numerical experiment that verifies the asymptotic analysis of Insertion Sort's running time. There are two functions and one experiment to do this.

(i) InsertionSort(A,n) takes as input an unordered array A, of length n, and returns both an in-place sorted version of A and a count t of the number of atomic operations performed by InsertionSort. Recall: atomic operations include mathematical operations like −, +, $*$, and /, assignment operations like ← and =, comparison operations like <, >, and ==, and RAM indexing or referencing operations like [ ].

(ii) randomArray(n) takes as input an integer n and returns an array A such that for each 0 ≤ i < n, A[i] is a uniformly random integer between 1 and n. (It is okay if A is a random permutation of the first n positive integers; see the end of Chapter 5.3.)

(a) From scratch, implement the functions InsertionSort and randomArray. You may not use any library functions that make their implementation trivial. You may use a library function that implements a pseudorandom number generator in order to implement randomArray.

Submit a paragraph that explains how you instrumented InsertionSort, i.e., explain which operations you counted and why these are the correct ones to count.

Hint: your instrument code should only count the operations of the InsertionSort algorithm and not the operations of the instrument code you added to it.

In [10]:
```
import random

#insertion sort algorithm based on textbook

def insertionSort(A,n):
    oCount = 0
    for i in range(2, len(A)):
        key = A[i]
        oCount = oCount+1
        j = i-1
        oCount = oCount+1
        while (j > 0 and A[j] > key):
            A[j+1] = A[j]
            oCount = oCount+1
            j = j-1
            oCount = oCount+1

        A[j+1] = key
        oCount = oCount+1

def randomArray(n):
    A = []
    for i in range(0, n):
        r = random.randint(0, n)
        A.append(r)

    return A
```

Sources

People

Krish Dholakiya

Gustav Solis

Websites

http://modernnotion.com/alex-the-only-parrot-to-ask-an-existential-question/ (http://modernnotion.com/alex-the-only-parrot-to-ask-an-existential-question/)