Homework 4

CS 677 - Analysis of Algorithm

Fall Semester - 2019

Dzung Bui

Due date: October 03, 2019

1. - *40 points*

Write an algorithm to rearrange elements of a given array of n real number.

2. Is Quicksort a stable sorting Algorithm? - *20 points*

Quicksort is not a stable sorting algorithm, because each value in the array will be switched with a value on the other side of the pivot. Selecting a value to switch is purely depending on the pivot value and the array, then the values in array can not be sorted in stable. Let consider the following example:

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
5	7	7	2	15	6	10	2	9	3

Assuming the value at p1 (5) is assigned as pivot. The value of array at p2 and p2 are 7. Starting from the left, it stops at p2 with value 7. From the right, the loop will stop at p10 with value 3. Value at p2 and p10 will be switched.

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
5	3	7	2	15	6	10	2	9	7

Starting a new loop, the left will stop at p3 with value 7. Similarly, the right will stop at p8 with value 2. Again, the value at p3 and p8 will be switched.

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
5	3	2	2	15	6	10	7	9	7

Now, the relative positions of two value 7 are changed, thus the Quicksort is not stable sorting algorithm.

3. - *20 points*

Consider the following problem: each of n visitors to a museum give their hand bags to the wardrobe attendant at the entrance. At the end of the day, the attendant gives the hand bags back to customers in random order. Use an indicator random variable to compute the expected number of visitors who get back their own bag.

Answer

Introduce event X - event that a visitor can get back their own bag after visiting museum. Defining the an indicator random variable I{X} associating with X.

$$I(X) = \begin{cases} 1, & \text{if } X \text{ occurs} \\ 0, & \text{if } X \text{ does not occurs} \end{cases} \tag{1}$$

The expected value of indicator random variable I{X} is A_X:

$$A_X = 1 * Pr(X) + 0 * Pr(X') = \frac{1}{n} \tag{2}$$

Let Y is the total number of customer can get back the right bag, so that:

$$Y = \sum_{i=1}^{n} A_{Xi} \tag{3}$$

The expected customer will get the correct bag:

$$E[Y] = E[\sum_{i=1}^{n} A_{Xi}] = \sum_{i=1}^{n} E[A_{Xi}] = \sum_{i=1}^{n} \frac{1}{n} = 1 \tag{4}$$

So, the number of customer is expected to get back the right bag is 1.

4. Exercise 9.3-5 - *20 Points*

Suppose that you have a "black-box" worst-case linear-time median subroutine. Give a simple, linear-time algorithm that solves the selection problem for an arbitrary order statistic.

Answer

Assuming we have a procedure MEDIAN that takes as parameters an array A, and subarray indices *left* and *right*, then returns the value x of the median element of A in *O(n)* in the worst case. The return value x will be input parameter of procedure PARTITION which divide the input array based on median value.

The procedure SELECT for finding the *ith* smallest element in A[left..right] as shown in the following:

SELECT(A,left,right,i)

if *left = right*

then return *A[left]*

$x \leftarrow$ *MEDIAN(A, left, right)*

pivot $\leftarrow$ *PARTITION(x)*

$k \leftarrow$ *pivot - left +1*

if $i = k$

then return *A[pivot]*

elseif $i < k$

then return *SELECT(A, left, pivot-1, i)*

else

then return *SELECT(A, pivot+1, right, i-k)*

Because x is the median of A, each of the subarrays *A[left .. pivot-1]* and *A[pivot+1 .. right]* has at most half the number of elements of array A. The recurrence for the worst-case running time of SELECT is $T(n) \leq T(n/2) + O(n) = O(n)$

5. Exercise 9.2-4 - *20 Points*

Suppose we use RANDOMIZED-SELECT to select the minimum element of the array A = (3; 2; 9; 0; 7; 5; 4; 8; 6; 1). Describe a sequence of partitions that results in a worst-case performance of RANDOMIZED-SELECT.

Answer

If we need to select the minimum element X = 0 in the array A:

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
3	2	9	0	7	5	4	8	6	1

The worst case is the pivot is selected the maximum value in the partition array. For the first loop, the pivot is selected item p3: 9 - the maximum value in the array. Thus the partition separate the initial array A into two sub-arra1, left and right one. The right one consists only the max value - 9. The left array contains the other items as following.

p1	p2	p3	p4	p5	p6	p7	p8	p9
3	2	1	0	7	5	4	8	6

After each iteration, only maximum item is removed from array. Thus, the running time is:

$$T(n) = \Theta(1)(computek) + \Theta(n)(partition) + T(n-1) = 1 + n + T(n-1) = \Theta(n^2) = \Theta(10^2) \quad (5)$$

6.Exercise 9.3-3 - *20 Points*

Show how quicksort can be made to run in O(nlgn) time in the worst case, assuming that all elements are distinct.

Answer

For a quicksort to run in *O(nlgn)* time in worst case, the pivot should be select in middle of the array, which is not the maximum or minimum value. To do that, a procedure SELECT is built with array A, bounds p and r, and rank i of an order statistic as the input parameters. This procedure will return the value with the rank *ith* of order statistic.

The second procedure PARTITION will get the output of SELECT as a pivot, and divide the input array into two relative equal sub-arrays, and return the address of the divided value. The variant algorithm of Quicksort as shown following:

Best_Quicksort(A,left,right)

if *left* < *right*

then:

i ← *[(right-left+1)/2*

middle ← *SELECT(A,left,right,i)*

pivot ← *PARTITION(middle)*

Best_Quicksort(A, left, pivot-1)

Best_Quicksort(A, pivot+1, right)

Because *Best_Quicksort* always recurses on subarrays that are at most half the size of the originial array, the recurrence for teh worst-case running time is $T(n) \leq 2T(n/2) + \Theta(n) = O(nlgn)$