University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020

February 18, 2020

Homework 6 problem 3

Problem 3 (10 points):

Let A and B be two sorted arrays of integers, each of length n. Show how you can find the median of the combined set of elements in $O(\log n)$ comparisons. (As in the Median algorithm discussed in lecture, you will need to solve the Select the k-th largest problem.) Justify your algorithm is correct.

Answer:

algorithm:

```
def findMedianSortedArrays(l, r, A, B):
# l: left bound of current interval of b-search
# r: right bound of current interval of b-search
n = len(A)
if l <= r:
    # set partition of A (pA) to be median of current interval, and
    # the partition of B (pB) to be the half size minus partition of A
    pA = l + (r - l) // 2
    pB = n - pA

    # check the corner case if the partition point is on left or
    # right of the array
    pA_l = A[pA - 1] if pA != 0 else -float('inf')
    pB_l = B[pB - 1] if pB != 0 else -float('inf')
    pA_r = A[pA] if pA != n else float('inf')
    pB_r = B[pB] if pB != n else float('inf')

    # if achieved the left bound of A's upper half (pA_r) is greater than the
    # right bound of B's lower half (pB_l) and right bound of A's lower half
    # (pA_l) is smaller than the left bound of B's upper half (pB_r), then
    # return the result based on whether the sum of size is odd or even:
    # odd: return bigger one of left bound of A & B's lower half
    # even: return the mean of (bigger one of left bound of A & B's
    # lower half) and (smaller one of right bound of their upper half)
    # if not the case, change the interval of consideration

    if (pA_l <= pB_r) & (pB_l <= pA_r):
        print((min(pA_r, pB_r) + max(pA_l, pB_l)) / 2)
```

```
        return (min(pA_r, pB_r) + max(pA_l, pB_l)) / 2
    elif pA_l > pB_r:
        return findMedianSortedArrays(l, pA - 1, A, B)
    elif pB_l > pA_r:
        return findMedianSortedArrays(pA + 1, r, A, B)
```

Proof: It is garantee to find the median :
(1) By maintaining the invariant $p_A+p_B = n$, change the partition point of A will affect the partition point of B. And this invariant garantee in every iteration, the two partition point separate two array in to two halves: (lower half of A and B) and (upper half of A and B).

(2) The return condition garantee the two final partition points will garantee to separate two array in two equal half
Since the array is sorted, we have:
$p_{A_l} \leq p_{A_r}$ and $p_{B_l} \leq p_{B_r}$
where
p_{X_l}: right bound of left half of array X
p_{X_r}: left bound of right half of array X
And if $p_{A_l} \leq p_{B_r}$ and $p_{B_l} \leq p_{A_r}$
then:
$p_{A_l} \leq min(p_{A_r}, p_{B_r})$ and $p_{B_l} \leq min(p_{A_r}, p_{B_r})$
$p_{A_r} \geq max(p_{A_l}, p_{B_l})$ and $p_{B_r} \geq max(p_{A_l}, p_{B_l})$
That means the left (lower) half is all smaller than the right (upper) half, therefore the partition satisfy the condition $p_{A_l} \leq p_{B_r}$ and $p_{B_l} \leq p_{A_r}$ will garantee to get median of two sorted array.

(3) The binary search of partition point on array A will halt: This is justify by a general binary search where each iteration the l or r pointer cut the problem size by half by inheriting the median pointer. And the binary search can work to find the optimal partition point because the array is sorted, and there is always a partition exist of A and B to find the median. The complexity:

$O(\log(n)))$
In each iteration, the problem size is cut by half, and there are $\log_2 n$ iteration to reach a single result, hence there are $T(n) = c \cdot \log_2 n = O(\log n)$