September 30, 2015

CS 577 - Homework 3
Sejal Chauhan, Vinothkumar Siddharth, Mihir Shete

1 Graded written problem

Input: A sequence of n real numbers a_1, a_2,..., a_n and a corresponding sequence of weights $w_1, w_2, . . . , w_n$. The weights are nonnegative reals that add up to 1, i.e. $\sum_{n}^{i=1} w_i = 1$.

Output: The weighted median of the sequence is the number a_k such that $\sum_{a_i<a_k} w_i$<1/2 and $\sum_{a_i<=a_k} w_i \geq 1/2$

Algorithm 1 Algorithm to find weighted median

```
1: procedure Weighted-Median($A_w^a$)
2:     if $length(A_w^a) = 1$ then
3:         return A[0]
4:     $Median =\leftarrow$ Selection($A_w^a, \lceil length(A_w^a)/2 \rceil$)     ▷ The Selection will happen over A
5:     $Pivot \leftarrow Median$
6:     $L_w^a, R_w^r \leftarrow$ Partition($A_w^a, Pivot$)
7:     if $\sum w_L$ <1/2 then
8:         $w_{Pivot} \leftarrow \sum w_L + w_{Pivot}$
9:         if $w_{Pivot} \geq 1/2$ then
10:            return $Pivot$
11:        Weighted-Median($Pivot|R_w^a$)
12:    else
13:        Weighted-Median($L_w^a$)
```

Our Algorithm uses the Selection() procedure discussed in section 6 of the Lecture notes on Divide and Conquer. When *Weighted-Median* is called it will first find the $\lceil n/2 \rceil$ smallest element in A using the Selection() procedure in linear time, let's call this value *Median*(because it is actually the median of sorted elements in A). We will use this value to *Partition* the Array A into 2 arrays L and R such that the Value a_i^L of all elements in L is less than or equal to value of *Median* and Value a_i^R of all elements in R is greater than *Median*.

If the weight of all elements in L is less than 1/2 and on adding the weight of the *Pivot* it becomes greater than or equal to 1/2 then *Pivot* is our Weighted Median and we return the value of the *Pviot*. If the weight of all elements in L is greater than or equal to 1/2 then we can say that the Weighted Median is present in the array L and we recursively call *Weigred-Median* on L.

If the aggregate of weight of elements in L and the weight of *Pivot* is less than 1/2 then we know the Weighted Median is in R. So we change the weight of the *Pivot* as: $w_{Pivot} \leftarrow w_{Pivot} + \sum L_w$. So *Pivot's* weight now also includes the weight of elements to its Left(i.e all elements less than equal to pivot).

After changing the weight of *Pivot* we will recursively call *Weighted-Median* procedure on array $Pivot|R$. Prepending *Pivot* with added weight to R assures that we consider the weight of all elements which are lesser than all elements in R while calculating the Weighted Median from the new sub-array and the Median retured by the recursive call is actually the Median of original Array A.

1.1 Correctness

We will proceed to prove the correctness of our algorithm by proving partial correctness and termination.

1.1.1 Partial Correctness

The recursive calls to *Weighted-Median* will be made on subarray of the input. It is trivial to see that the length of the subarray will never be negative or 0. So the recursive calls to *Weighted-Median* will always have a valid array as input.

The algorithm returns from 2 places. The first case is trivially true because if the length of the array is 1 then the only element in the array is the Weighted-Median. In the second return we see that the sum of all elements to the Left of the *Pivot* in the partitioned array is less than 1/2 and when we add the weight of the *Pivot* the aggregated weight becomes $\geq 1/2$ then we say that the *Pivot* is the Weighted Median which is true as per definition because all the elements to the left of *Pivot* in the partitioned array are less than or equal to the *Pivot*.

1.1.2 Termination

Let $\mu(length(A))$ be the potential function then we can see that in the recursive calls

$$\mu(length(A_{recursive)}) = \mu(length(A[0...\lceil n/2 \rceil]))OR\mu(length(A_{recursive)}) = \mu(length(A[\lceil n/2 \rceil + 1...n-1]))$$

So the potential function will decrease each recursive call and our program is guaranteed to terminate.

1.2 Complexity Analysis

Its trivial to see that our recursive calls operate on an input of size $\lceil n/2 \rceil \pm 1$ and in worst case our algorithm will go through log(n) recursive calls.

All the steps in our procedure take constant time except *Selection* and *Partition* which take linear time. So total running time of our alogrithm in worst case will be:

$$c*(n/2^0) + c*(n/2^1) + c*(n/2^2)... + c*(n/2^{log(n)}) \leq c*n*(1 + 1/2 + 1/4....)$$

The above is a geometric series with sum limiting to 2. Hence, our algo can take c*2*n time in worst case and so it is linear in n i.e O(n)