CS 218, Spring 2017 Homework 4
Abhishek Kumar SRIVASTAVA May 3, 2017
Student ID: 861307778

I certify that this submission represents my own original work

---

Solution 1. Part a: When input elements are divided in group by 7.

Using the same example as given in the class we can observe that for the *median* of *medians* at least $n/7$ groups contribute four elements that are bigger than *median*, except for the last group with less than 7 elements and the group with *median* itself.

From the above observation we conclude that there are at least this amount of elements are greater than the *median*:

$$4([1/2[n/7]] - 2) \geq (2n/7) - 8$$

So the worst case split can has $5n/7 + 8$.

$$\max\{|L|, |R|\} < (5n/7) + 8$$

Therefore recurrence relation is:

$$\begin{aligned}
T(n) &= T([n/7]) + T(5n/7 + 8) + O(n) \\
&\leq c((n/7) + 1) + 5cn/7 + 8c + O(n) \\
&= cn - [c((n/7) - 9) - dn] \\
&\leq cn
\end{aligned}$$

The last step hold since $n \geq 70 \implies n/7 - 9$ is positive. Choosing c big enough will result $c((n/7) - 9) - dn$ positive and last line holds. Therefore,

$$\boxed{T(n) = O(n)}$$

Part b: Using group of 3 we can see that at least $n/3$ groups contribute two elements that are bigger than *median*. Therefore,

$$2([1/2[n/3]] - 2) \geq (n/3) - 4$$

So the worst case split can has $2n/3 + 4$ and it will result into following recurrence relation.

$$\begin{aligned}
T(n) &= T([n/3]) + T(2n/3 + 4) + O(n) \\
&\leq c((n/3) + 1) + 2cn/3 + 4c + O(n) \\
&= cn - [-5c - dn] \\
&\geq cn
\end{aligned}$$

Last line is true because for $n \geq 0$ and $-5c - dn$ is negative. Therefore,

$$\boxed{T(n) \neq O(n)}$$

Solution 2. Given: For n distinct integer $x_1, x_2, \dots x_n$ with positive weights $w_1, w_2, \dots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the weighted median is the element x_k satisfying

$$\sum_{i:x_i<x_k} w_i < 1/2 \quad \text{and} \quad \sum_{i:x_i>x_k} w_i \leq 1/2$$

To find the weighted median first find the median of $x_1, x_2, \dots x_n$ by using linear selection algorithm and picking the element of rank $n/2$ $i.e$ median. Maintain two arrays named left and right subarray. Traverse the whole array comparing each element with median if it is small then insert in the left subarray and add the corresponding weight otherwise it is inserted in the right subarray. After whole array traversal we check that if the left subarray sum is greater or less than 1/2. If is greater we do recursive operation on the left subarray since weighted median lies in the left subarray, otherwise we will do recursive operation on the right subarray to find weighted median.

The recurrence relation for the above described algorithm is:

$$T(n) = T(n/2) + O(n)$$

$T(n/2)$ because we will only traverse either left or right subarray. $O(n)$ because we are traversing whole array for comparing all the elements with the median found and we are finding median of the array using linear selection. By using master theorem case:3 we can get the asymptotic time as $T(n) = O(n)$.

```
function WeightedMedian(array, sum)
    len = length(array)
    if len == 1 then                                        ▷ Base Condition
        return array[0]
    end if
    median = Linear_Selection(array,len/2)                  ▷ rank = len/2
    leftSubArray;
    rightSubArray;
    leftSum = 0;
    for i ← 0 to length − 1 do
        if array[i] < median then
            insert(leftSubArray,array[i]);                  ▷ insert element in array given
            leftSum += weight[i];                           ▷ adding corresponding weight
        else
            insert(rightSubArray,array[i]);                 ▷ insert element in array given
        end if
    end for
    if sum+leftSum > 1/2 then
        WeightedMedian(leftSubArray, sum);
    else
        WeightedMedian(rightSubArray,leftSum)
    end if
end function
```

Solution 3. Given: t and p strings over an alphabet Σ of size σ. We have to find For each $\gamma \in \Sigma$, calculate $C_\gamma(i) = \sum_{k=0}^{n-1} equal_\gamma(a_{i+k}, b_k)$ for $i \in \{0, 1, \ldots, n\}$, where $equal_\gamma(\alpha, \beta)$ is 1 if $\alpha = \beta = \gamma$, zero otherwise.

Doing this in a naive way will give us $O(n^2)$. Using FFT we can get $O(nlogn)$ but for using convolution we have to first transform our input string into something different for both input array and pattern array.

For a $\gamma \in \Sigma$ we will transform the string array t and p into t_γ and p_γ. Transformation will be done by traversing t and p and setting 1 if $t[i]$ and $p[i]$ matches with γ otherwise 0. It will be transform into the array of 0's and 1's. And just like polynomial multiplication we can apply FFT on t_γ and p_γ which will give back C_γ. If the length of t and p is different we can add 0 in the end to do padding and making them of equal length and ignoring the coefficients of the padded elements.The length of the convolution will be $len(t) + len(p) - 1$. I am treating FFT as blackbox for which i will give input and i will get the corresponding result.

Given length of array is n and σ we will get the following complexity for the above described algorithm: For a single character,

$$T(n) = O(n) + O(nlogn)$$

$O(n)$ because we are transforming the string array by traversing it and setting 1's and 0's. $O(nlogn)$ because we are using convolution to computer C_γ. For σ number of characters.

$$T(n) = \sigma \cdot (O(n) + O(nlogn)) \tag{1}$$

$$\boxed{\therefore T(n) = O(\sigma nlogn)}$$

```
function COMPUTE_C_GAMMA(t, p, γ)
    t_gamma = Transform(t,γ)
    p_gamma = Transform(p,γ)
    C_gamma = Convolution(t_gamma,p_gamma)
    return C_gamma
end function

function TRANSFORM(array, γ)
    temp_array(0 ... length(array) − 1)
    for i ← 0 to length(array) − 1 do
        if array[i] == γ then
            temp_array[i] = 1
        else
            temp_array[i] = 0
        end if
    end for
    return temp_array
end function
```