Problem Set 1

All parts are due Tuesday, September 27 at 11:59PM.

Name: Dimitris Koutentakis

Collaborators: Driss Hafdi

Part A

Problem 1-1.

(a) By logarithm rules, we get the following increasing order: $f_5 < f_2 < f_3 < f_1 < f_4$

(b) Again, by logarithm rules, we get the following order: $f_1 < f_6 < f_3 < f_5 < f_2 < f_4$

(c) By comparing the functions, we conclude that: $f_5 < f_1 < f_2 < f_4 < f_3$

Problem 1-2.

(a) 1. From the Master Theorem, we have the 3rd case, hence:

$$T(n) = \Theta(n^2) \tag{1}$$

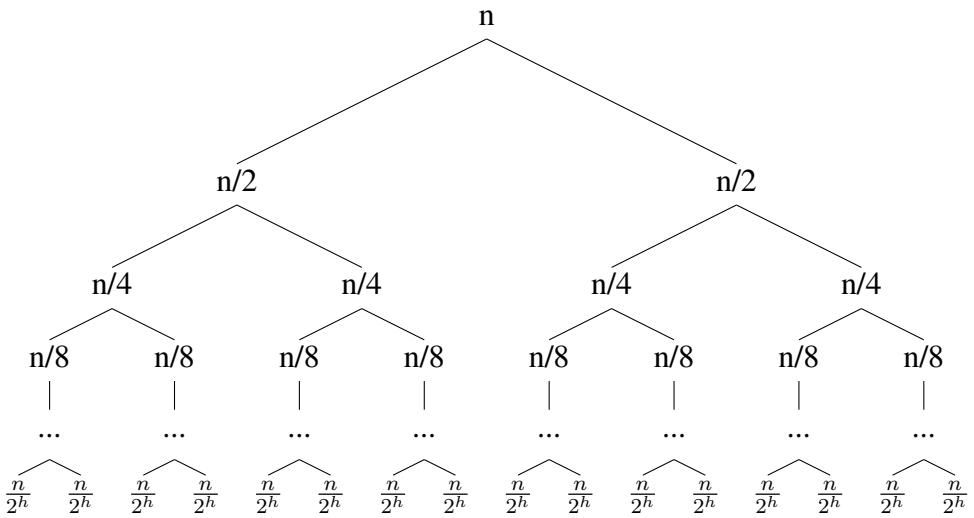
2. From first case of the Master Theorem, we have:

$$T(n) = \Theta(n^{\frac{lg(10)}{lg(3)}}) \tag{2}$$

3. • Master Theorem: From second case of the Master Theorem:

$$T(n) = \Theta(nlog(n)) \tag{3}$$

- Recursion Tree Method:



(4)

The height of the tree h is $lg(n)$. The number of leaves is n. Thus, the time it will take is:

$$T(n) = \Theta(nlog(n)) \tag{5}$$

4.

$$T(n) = T(n/2) + cn = T(n/4) + cn + cn/2 = ... = c(1 + 1/2 + 1/4 + ...) = O(log(n)) \tag{6}$$

(b) 1.

$$T(n) = T(n-1) + O(1) \tag{7}$$

2.

$$T(n) = T(n-1) + T(n-2) + O(1) \tag{8}$$

Problem 1-3.

(a) 1. Without loss of generality, we can take the case that the different color cell is on the top left. Then if A[0][k]==A[n-1][k] then the unbalanced sub-array will be on the left of the column k, since the only cell colored differently will still be the one on the top left.Then the unbalanced sub-array will have corners: (A[0][k],A[n-1][k], A[0][0],A[n-1][0]) Otherwise, the unbalanced sub-array will be on the right, since the only different corner cell will be one of the two in column k. Then the unbalanced sub-array will have corners: (A[0][k],A[n-1][k], A[0][n-1],A[n-1][n-1])

2. All we have to do in order to find a peak square is repeat the above method until n=m=2 with $k = \lfloor n = 2 \rfloor$. After that, we will repeat the test on the other dimension, with $k = \lfloor m = 2 \rfloor$.
3. Recurrence relation:

$$T(n) = O(1) + T(n/2) \tag{9}$$

By the master theorem, the running time of this relation will be T(n)=O(log(n)).

(b) 1. In order to find a circular sub array, I would split the original array into A[0:k] and A[k+1:n-1], and choose the one that includes the maximum of the four values.
2. After picking a sub-array in the way described above, I would recurse on that circular sub array, until there are only 4 values and then choose the maximum of the four as a peak. The time for that would be:

$$T(n) = T(n/2) + c = T(n/4) + c/2 = ... = c(1 + 1/2 + 1/4 + ...) = O(log(n) \tag{10}$$

Part B

Problem 1-4.

(a) *On alg.csail.mit.edu*

(b) *On alg.csail.mit.edu*

(c) *On alg.csail.mit.edu*

(d) For any part of the program, the initialization of the class would take

$$T(n) = O(nm) + O(n^2m) + O(m) + O(n) \tag{11}$$
$$\Rightarrow T(n) = O(n^2m) \tag{12}$$

My part (b) would have a run-time of:

$$T(n) = O(n) + O(k) + O(1) + O(m) \tag{13}$$
$$\Rightarrow T(n) = O(m + n + k) \tag{14}$$

Finally, part c would have a run-time of:

$$T(n) = O(qn) + O(k) \tag{15}$$
$$\Rightarrow T(n) = O(qn + k) \tag{16}$$

Hence, we can see that the slowest part would be the initialization with $T(n) = O(n^2m)$

(e)

1. We can see that in general, TF_ IDF will give a more intuitive result, since it gives importance on the rarity of the word, thus it gives a more specific result. However, there are some cases, such as with the terms *"Lime"* or *"Apple"*, where the first method is better, because it gives a more general result, instead of presenting only the "technical" side.
2. It is logical to assume that longer documents would have words appearing more times than shorter documents. That means that the TF_ IDF might end up promoting longer documents just because they have more words, instead of promoting documents that have a bigger ratio of the word appearing. In order to solve that, we could divide the TF with the length of the document.
3. The 3 first results for part (a) are: "Apple Inc" with a score of 0.4488, "Banana" with a score of 0.4518, and "Tomato" with a score of 0.4558. The 3 first results for part (b) are: "Apple Inc" with a score of 1.37, "Macintosh" with a score of 1.48, and "Pear" with a score of 1.48. We can see that the first method results in a more general answer, whereas the second one gives a more narrow result. In my opinion, the method that works better is the first one in this case, because it doesn't limit the results to Apple Inc related articles. However, that depends greatly on what one is looking for.
   In order to improve TF_ IDF in this case, we could make it such that it ignores the word in the title of the original document when looking for similarities between documents. That would place a bigger importance on other words. For example it is logical to expect that the word "Apple" appears many times in anything related to "Apple Inc", however, if we ignore the term "apple" because it was in the title of the first document, a bigger importance will be placed on other words such as "fruit", "red", "computer" etc.