COMPSCI390.01: Algorithmic Foundations of Data Science
Midterm I

NAME:

Prob #	Score	Max Score
1		20
2		25
3		30
4		25
Total		100

Instructions:

1. If you write pseudocode, make sure to describe your idea in words.
2. Analyze the time complexity of your algorithms if asked.
3. You may use any algorithm covered in the class without detailed description, but you should be explicit about the input and output.
4. For any change that you make to an algorithm covered in class, you should describe the changes precisely.

Problem 1: (**20 pts**) Construct a binary Huffman encoding tree for the string

`bubba ate a banana`

and show its corresponding encoding.

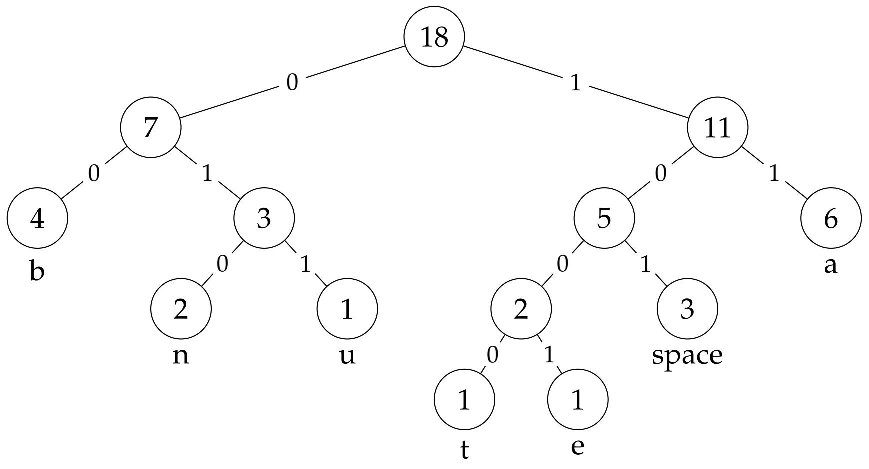
Solution. The frequency count for each character in this string is

Character	'a'	'b'	' '	'n'	'u'	't'	'e'
Count	6	4	3	2	1	1	1

The key to constructing the Huffman tree is we iteratively merge the least frequent nodes. One way (note the answer is not unique for when ties occur, you may break them in any way you wish) is as follows (where we use ϕ to represent the space character):

- Merge $\{t\}, \{e\}$. Result: $\{a\} = 6, \{b\} = 4, \{\phi\} = 3, \{n\} = 2, \{u\} = 1, \{t, e\} = 2$.
- Merge $\{n\}, \{u\}$. Result: $\{a\} = 6, \{b\} = 4, \{\phi\} = 3, \{n, u\} = 3, \{t, e\} = 2$.
- Merge $\{t, e\}, \{\phi\}$. Result: $\{a\} = 6, \{b\} = 4, \{n, u\} = 3, \{t, e, \phi\} = 5$.
- Merge $\{b\}, \{n, u\}$. Result: $\{a\} = 6, \{b, n, u\} = 7$, $\{t, e, \phi\} = 5$.
- Merge $\{a\}, \{t, e, \phi\}$. Result: $\{a, t, e, \phi\} = 11$, $\{b, n, u\} = 7$.
- One final merge and our tree is complete.

See the following tree for an illustration of this process.



Correspondingly, the encoding for each character is summarized in the table below. I will skip the step of actually converting the original string into a long binary string for you won't read it anyways. :)

Character	'a'	'b'	' '	'n'	'u'	't'	'e'
Encoding	11	00	101	010	011	1000	1001

Problem 2: (**25 pts**) Consider a special instance of *k-center* in $\mathbb{R}^1$ where we wish to partition a set $X = \{x_1, \ldots, x_n\}$ of n real numbers into 2 clusters so that the maximum distance between any point of X and the center of its assigned cluster is minimized. That is, compute a partition of X into two sets X_1 and $X_2 = X - X_1$, along with their centers c_1, c_2, respectively, such that the following objective function is *minimized*:

$$\max\{\max_{x \in X_1} |c_1 - x|, \max_{y \in X_2} |c_2 - y|\}.$$

Describe an $O(n \log n)$-time algorithm that achieves this goal, and briefly justify its time complexity and correctness. (**Hint:** *What property does a* 1*-dimensional optimal clustering have, and how do you use it to compute an optimal clustering efficiently?*)

Solution. It follows from the class discussion and the clustering problem in HW2 that the x-spans of the two clusters do not overlap, i.e., if $x_1 < x_2 \cdots < x_n$ then the two clusters are of the form $X_1 = \{x_1, \ldots, x_i\}$ and $X_2 = \{x_{i+1}, \ldots, x_n\}$ for some i.

Let c_1, c_2 be the centers of X_1, X_2, respectively. Since the cost of X_i is defined as the distance between c_i and point of X_i farthest from c_i, we should choose c_i to be the midpoints of the leftmost and the rightmost points of X_i, i.e, $c_1 = (x_1 + x_i)/2$ and $c_2 = (x_{i+1} + x_n)/2$. In that case, the cost of X_1 is $(x_i - x_1)/2$ and the cost of X_2 is $(x_n - x_{i+1})/2$.

In view of these observations, here is the algorithm: First, sort the points of X, and let $x_1, \ldots, x_n$ be the sorted sequence. For every $1 \le i < n$, we do the following, we set

$$p_i = \max\{(x_i - x_1)/2, (x_n - x_{i+1})/2\}.$$

Let $i^* = \arg\min_{1 \le i < n} p_i$. We return $X_1 = \{x_1, \ldots, x_{i^*}\}$ and $X_2 = \{x_{i+1}, \ldots, x_n\}$.

The correctness of the algorithm follows from the above two observations. As for the run time, sorting takes $O(n \log n)$ time, computing p_i's takes $O(n)$ time, and computing i^* also takes $O(n)$ time. Hence, the overall run time is $O(n \log n)$.

Problem 3: (**30 pts**) We wish to construct the LSH functions with respect to the ℓ_1-metric in $\mathbb{R}^d$ for some fixed constant $d \geq 1$. (Recall $\|x-y\|_1 = \sum_{i=1}^{d} |x_i - y_i|$.)

Let $r > 0$ be a given integer, and let $c \geq 1$ be another integer. Consider building a randomly shifted d-dimensional grid with side length $\Delta = cr$, and define a hash function h that maps points lying within the same grid-cell to the same value. That is, we uniformly choose a random shift value $a = (a_1, \ldots, a_d) \in [\Delta]^d$ and for $x = (x_1, \ldots, x_d)$, set

$$h_a(x) = \left(\left\lfloor \frac{x_1 + a_1}{\Delta} \right\rfloor, \ldots, \left\lfloor \frac{x_d + a_d}{\Delta} \right\rfloor \right).$$

Consider two points $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$.

(i) (**7 points**) What is the probability that a grid line in the i^{th} dimension separates x, y? In other words, what is the probability that the i^{th} coordinate of $h_a(x)$ and $h_a(y)$ differ, assuming $a = (a_1, \ldots, a_d)$ is chosen randomly from $[\Delta]^d$?

(ii) (**8 points**) Provide an upper bound of $\Pr_{a \sim [\Delta]^d}[h_a(x) \neq h_a(y)]$ using $\|x-y\|_1$ and other defined parameters. (**Hint:** *Use (i) and the union bound for probability.*)

(iii) (**15 points**) Let $\epsilon > 0$ be a fixed parameter. Obtain a lower bound for $\Pr_{a \sim [\Delta]^d}[h_a(x) = h_a(y) \mid \|x-y\|_1 \leqslant r]$ and an upper bound for $\Pr_{a \sim [\Delta]^d}[h_a(x) = h_a(y) \mid \|x-y\|_1 > (1+\epsilon)r]$.

Solution.

(i) We assume $|x_i - y_i| \leqslant \Delta$, for otherwise they are far apart enough to be guaranteed separated, regardless of what the random shift offset is.

This part is now identical to the 1-dimensional modular hash covered in Lecture 11/12. We view the randomly shifted grids as objects that "cut" through the space. Consider any Δ consecutive integers containing both x_i and y_i. With equal probability ($1/\Delta$ each), the i^{th} component of the randomly shifted grid will cut through any integer among them. The desired outcome (i.e. the i^{th} coordinate of $h_a(x), h_a(y)$ differ) happens when the cut also lands between x_i and y_i, effectively sending x_i and y_i to different cells.

Out of the Δ equally likely events, $|x_i - y_i|$ are desired as there are $|x_i - y_i|$ integers between x_i, y_i. Thus, the probability we seek is $|x_i - y_i|/\Delta$.

(ii)

$$\begin{aligned}
\mathbb{P}(h_a(x) \neq h_a(y)) &= \mathbb{P}(h_a(x), h_a(y) \text{ differ along at least one coordinate}) \\
&= \mathbb{P}\Big(\bigcup_{i=1}^{d} \{h_a(x), h_a(y) \text{ differ along the } i^{\text{th}} \text{ coordinate}\}\Big) \\
[\text{union bound}] &\leqslant \sum_{i=1}^{d} \mathbb{P}(h_a(x), h_a(y) \text{ differ along the } i^{\text{th}} \text{ coordinate}) \\
[(i)] &= \sum_{i=1}^{d} \frac{|x_i - y_i|}{\Delta} = \frac{\|x-y\|_1}{\Delta}.
\end{aligned}$$

(iii) We apply (ii). If $\|x-y\| \leqslant r$, then (ii) implies $\mathbb{P}(h_a(x) \neq h_a(y)) \leqslant \|x-y\|_1/\Delta \leqslant r/\Delta$, so taking complement gives

$$\mathbb{P}(h_a(x) = h_a(y) \mid \|x-y\| \leqslant r) \geqslant 1 - \frac{r}{\Delta}.$$

Likewise,

$$\mathbb{P}(h_a(x) = h_a(y) \mid \|x-y\| > (1+\epsilon)r) \leqslant 1 - \frac{(1+\epsilon)r}{\Delta}.$$

Note the second inequality is in fact not rigorous, but we will give full credit if you wrote so, since the lecture materials taught it this way. A more formal fix using (i): in order for $h_a(x) = h_a(y)$ we need coordinate-wise inequality, where the i^{th} coordinate values match with probability $1 - |x_i - y_i|/\Delta$. Therefore

$$\mathbb{P}(h_a(x) = h_a(y) \mid \|x-y\| = r') = \prod_{i=1}^{d} \left(1 - \frac{|x_i - y_i|}{\Delta}\right)$$

subject to $\sum_{i=1}^{d} |x_i - y_i| = r' = (1+\epsilon)r$. This implies the total sum $\sum_{i=1}^{d}(1 - |x_i - y_i|/\Delta)$ is fixed, and AM-GM implies that under this situation, the maximum product is attained when all terms equal. Using $(1 - r'/d)^d \leqslant e^{-r'} \leqslant 1 - r' + (r')^2/2$ [the second $\leqslant$ follow from Taylor expansion], a more rigorous bound would be

$$\mathbb{P}(h_a(x) = h_a(y) \mid \|x-y\| > (1+\epsilon)r) \leqslant 1 - \frac{(1+\epsilon)r}{\Delta} + \frac{(1+\epsilon)^2 r^2}{2\Delta^2}.$$

Problem 4: (**25pts**) Suppose we have two sets L and R, as well as two corresponding Bloom filters B_L, B_R storing L and R, respectively, that were constructed using the same hash function. Recall Bloom filters never have false negatives (if the query procedure for an item x returns no, then x is not in the set). Suppose the false positive rates (FPR) (i.e., the probability of the query procedure returning yes on an item x not in the set) of B_L, B_R are f_L, f_R, respectively.

(i) (**13 points**) Given B_L and B_R, how can one construct a Bloom filter $B_{L\cup R}$ that stores the union $L \cup R$ *without accessing individual items of the sets*? What is the runtime to construct $B_{L\cup R}$? Justify that $B_{L\cup R}$ never returns false negatives. Estimate its FPR.

(ii) (**12 points**) Suppose instead we want to construct a Bloom filter $B_{L\cap R}$ to store the intersection $L \cap R$. How will one construct it from B_L and B_R without having access to the set $L \cap R$? Does your construction return false negatives? Does your algorithm return the same Bloom filter as if one had constructed it directly for the set $L \cap R$ using the same hash function as for B_L and B_R (assuming we had access to the set $L \cap R$)? Justify your answers.

Solution.

(i) We let B be the bitwise OR between B_L and B_R (i.e., a bit in our data structure is set to 1 if either bits of B_L or B_R at the same location is 1). Doing so requires a linear scan of B_L and B_R so the total runtime is within a constant factor of the original Bloom filter sizes, or equivalently $= O(|B_L|) = O(|B_R|)$.

The query procedure remains the same: check all locations that the query element is hashed to (call these the relevant bits), and return NO if and only if all relevant bits are 0.

Our data structure B never returns false negatives. If $x \in L$, then every relevant bit in B_L is set to 1, and the OR operation ensures the relevant bits in B are also 1. Thus, performing query(x) on B will return YES. Likewise if $x \in R$.

A false positive of B occurs when an element $x \notin L \cup R$ is mistakenly reported as present by at least one of the individual filters B_L, B_R. The complement is when both B_L, B_R successfully avoids false positives, which happens with probability $(1 - f_L)(1 - f_R)$. Therefore, the FPR of B is $1 - (1 - f_L)(1 - f_R) = f_L + f_R - f_L f_R$.

(ii) Similar to (i), here we define B to be the bitwise AND between B_L and B_R. The runtime is clearly identical. There will still be no false negatives, for if $x \in L \cap R$ then every relevant bit in both B_L and B_R will be toggled, so all relevant bits in B will also be set to 1.

However, our Bloom filter (likely) differs from one directly built upon $L \cap R$. Consider some element x that serves as false positives for both B_L and B_R. All relevant bits in B_L, B_R are 1, and so are the relevant bits in B. Hence query(x) on B would return YES, whereas this *could have been* avoided if we build a Bloom filter directly on $L \cap R$.