CSE521 Homework 1 Solution

Problem 1.

$$\left(1-\frac{1}{1000}\right)^n \ll 1 \equiv \left(1+\frac{1}{n}\right)^n \ll \log\log^* n \ll \log^* n \equiv \log^* \log n$$
$$\ll \log n \equiv \log(n \log n) \equiv \log_{1000} n \ll (\log n)^{1000} \ll 2^{\sqrt{\log n \log\log n}} \ll n^{1/100}$$
$$\ll n \ll n \log n \ll n^{\log\log n} \equiv (\log n)^{\log n} \ll n^{\log n}$$
$$\ll \left(1+\frac{1}{1000}\right)^n \ll 2^n \equiv n^{n/\log n} \ll (\log 1000)^n \ll (\log n)^n$$

Problem 2.

1. Step (a) and (c) can both be implemented in $O(n)$ time in the obvious ways. The running time $\sum_{i=1}^n O\left(B_i^2\right)$ of step (b) is achieved by sorting each bucket by any popular sorting algorithm (e.g. bubble sort or insertion sort). Thus, the algorithm can be implemented to achieve the running time $O(n) + \sum_{i=1}^n O(B_i^2)$ as required.

2. By linearity of expectation, we have:

$$\mathbf{E}\left(O(n) + \sum_{j=1}^{n} O(B_j^2)\right) = O(n) + \sum_{j=1}^{n} \mathbf{E}\left(O(B_j^2)\right) \tag{1}$$
$$= O(n) + \sum_{j=1}^{n} O\left(\mathbf{E}\left(B_j^2\right)\right) \tag{2}$$

Thus, the problem reduces to proving that $\mathbf{E}\left(B_j^2\right) = O(1)$. To this end, let X_{ij} be the indicator that the ith input falls into the jth bucket, i.e.

$$X_{ij} = \begin{cases} 1 & \text{if } x_i \in B_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Then, by linearity of expectation:

$$\begin{aligned}
\mathbf{E}\left(B_j^2\right) &= \mathbf{E}\left(\left(\sum_{i=1}^{n} X_{ij}\right)^2\right) && (4)\\
&= \sum_{i=1}^{n} \mathbf{E}\left(X_{ij}^2\right) + 2\sum_{i\neq k} \mathbf{E}\left(X_{ij}X_{kj}\right) && (5)\\
&= \sum_{i=1}^{n} \mathbf{Pr}\left(x_i \in B_j\right) + 2\sum_{i\neq k} \mathbf{Pr}\left(x_i \in B_j \text{ and } x_k \in B_j\right) && (6)\\
&= \sum_{i=1}^{n} \mathbf{Pr}\left(x_i \in B_j\right) + 2\sum_{i\neq k} \mathbf{Pr}\left(x_i \in B_j\right)\mathbf{Pr}\left(x_k \in B_j\right) && (7)\\
&= n \cdot \frac{1}{n} + n(n-1)\frac{1}{n^2} && (8)\\
&\leqslant 2. && (9)
\end{aligned}$$

where (7) follows from the fact that the two events "$x_i \in B_j$" and "$x_k \in B_j$" are independent; and (8) follows from the fact that x_i and x_k are both chosen uniformly at random in $[0,1]$. The last inequality completes the proof.

Problem 3. Given a sequence Z, let $Z[i]$ denote the ith character of Z and $Z[:i]$ denote the subsequence containing the first ith characters of Z. Given two sequences X and Y, let $V[i,j]$ denote the edit distance of $X[:i]$ and $Y[:j]$. Consider the matches of $X[i]$ and $Y[j]$. There are 3 cases:

1. $X[i]$ *is matched with* $Y[j]$. Then $V[i,j]$ can be decomposed into: (i) the cost to match $X[i]$ and $Y[j]$, and (ii) the cost to match $X[:i-1]$ and $Y[:j-1]$. Therefore, in this case, $V[i,j] = V[i-1,j-1] + c(X[i], Y[j])$ where $c(\alpha, \beta)$ is the cost of replacing α by β, which is 1 if $\alpha \neq \beta$ and 0 otherwise.

2. $X[i]$ *is matched with* $Y[k]$ *for* $k < j$. Then $V[i,j]$ can be decomposed into: (i) the cost to insert $Y[j]$ into X after $X[i]$ or delete $Y[j]$ from Y – these two costs are the same in this question, but in general, we can take the smaller of the two, and (ii) the cost to match $X[:i]$ and $X[:j-1]$. Therefore, in this case, $V[i,j] = V[i,j-1] + 2$.

3. $Y[j]$ *is matched with* $X[k]$ *for* $k < i$. Similar to the above case, we have $V[i,j] = V[i-1,j] + 2$.

Since these cases cover all the situations of matching $X[:i]$ and $Y[:j]$, we have the following recurrent formula

$$V[i,j] = \min\left(V[i-1,j-1] + c(X[i], X[j]), V[i-1,j] + 2, V[j-1,i] + 2\right) \quad (10)$$

From (10), we can design a dynamic programing algorithm to compute the table $V[i,j]$ for $1 \leqslant i \leqslant |X|$ and $1 \leqslant j \leqslant |Y|$. The edit distance between X and Y is $V[|X|, |Y|]$. The algorithm's running time is $O(|X| \cdot |Y|)$, since the computation of each cell of the table using (10) takes $O(1)$ time.

Problem 4.

1. We will prove the correctness of the first algorithm by induction. The correctness of the second algorithm follows similarly.

   - For $n = 1$ and 2, the algorithm's correctness can be verified easily.
   - Assume that the algorithm works for $n/2$, we show that it works for n.

     Set the coordinate's origin at the center of the $n \times n$ pixel map. Let M and N be the pixel map before and after the rotation. We need to prove that for all $-n/2 <\leqslant x \leqslant n/2, -n/2 \leqslant y \leqslant n/2$, $M[x,y] = N[y,-x]$. Assume that $x \leqslant 0$ and $y \geqslant 0$ (the other cases are symmetric). Let M' be the resulted pixel map after the blit steps. Then $M(x,y) = M'[x+n/2, y]$. By the induction hypothesis, after the rotation step, we have $M'[x+n/2, y] = N[x', y']$, where (x', y') is the image of $(x+n/2, y)$ in the -90-degree rotation around $(n/4, n/4)$. Simple algebra shows that $x' = y$ and $y' = -x$, thus completes the proof.

2. Let $f(n)$ be the number of blits the first algorithm performs on a $n \times n$ pixel map. Then

$$f(1) = 0 \tag{11}$$

$$f(n) = 4f(n/2) + 5 \qquad \text{for } n > 1 \tag{12}$$

This recurrent formula yields $f(n) = 5(n^2 - 1)/3$.

The number of blits the second algorithm performs follows exactly the same recurrent formula. Thus the second algorithm performs the same number of blits with the first one.

3. Let $T(n)$ be the running time of the first algorithm on a $n \times n$ pixel map. We have:

$$T(1) = 0 \tag{13}$$

$$T(n) = 4T(n/2) + 5O\left(\frac{n^2}{4}\right) \qquad \text{for } n > 1 \tag{14}$$

By Master Theorem, we have $T(n) = O\left(n^2 \log n\right)$.

The running time of the second algorithm can be analyzed similarly.

4. Similar to above, the running time $T(n)$ of the first algorithm on a $n \times n$ pixel map satisfies the following recurrent formula:

$$T(1) = 0 \tag{15}$$

$$T(n) = 4T(n/2) + 5O\left(\frac{n}{2}\right) \qquad \text{for } n > 1 \tag{16}$$

Again, by Master Theorem, we have $T(n) = O\left(n^2\right)$.

The running time of the second algorithm can be analyzed similarly.

Problem 5. The problem can be solved by a breath-first traversal of (each connected component of) the graph. Every time we visit a node, we mark it as visited. If we ever re-visit a visited node, the graph contains a cycle. Otherwise, it's a forest. This ways, each node is visited at most once (except for possibly one node in case the graph does contain a cycle) and each edge is visited at most twice. Thus, the running time is $O(n + m)$.