1 Hashing with open addressing

a)

We define as X_i the random variable of how many probes required for the i^{th} insertion. The total entries of the hash table are $2 \cdot n$ and $i-1$ entries have already been filled so the probability of finding an entry for i^{th} insertion is $Pr(X_i) = 1 - \frac{i-1}{2 \cdot n} \geq \frac{1}{2}$.Hence, we do not find an empty cell at least half of the times. Based on that and the fact that each probe is independent compared to all others we calculate the probability that i^{th} insertion requires more than m probes as following:

$$Pr(X_i > m) \leq \prod_{i=1}^{m} (\frac{1}{2})^i = (\frac{1}{2})^m \tag{1}$$

b)

Based on the subquestion (a) we probe the hash table $m = 2 \cdot logn$ times so the equation (1) takes the following form:

$$Pr(X_i > m) = Pr(X_i > 2 \cdot logn) \leq \prod_{i=1}^{2 \cdot logn} (\frac{1}{2})^i = (\frac{1}{2})^{2 \cdot logn} = \frac{1}{n^2}$$

c)

We define $X = max_{1 \leq i \leq n} X_i$, as the maximum number of probes required by any of the n insertions. So easily we can rewrite the required probability as:

$$Pr(X > 2 \cdot logn) = Pr(\bigcup_{i=1}^{n} (X_i > 2 \cdot logn))$$

Using the union bound to the previous equation we have:

$$\begin{aligned} Pr(X > 2 \cdot logn) = Pr(\bigcup_{i=1}^{n} (X_i > 2 \cdot logn)) &\leq \sum_{i=1}^{n} Pr(X_i > 2 \cdot logn) \xrightarrow{subquestion(b)} \\ &\leq \sum_{i=1}^{n} \frac{1}{n^2} = \frac{1}{n} \end{aligned}$$

d)

Using of conditional expectation for expectation probability $E[X]$ on the condition that $X > 2 \cdot logn$ we have:

$$\begin{aligned} E[X] &= Pr(X|X > 2 \cdot logn) \cdot E[X|X > 2 \cdot logn] + Pr(X|X \leq 2 \cdot logn) \cdot E[X|X \leq 2 \cdot logn] \Longrightarrow \\ &\leq \frac{1}{n} \cdot E[X|X > 2 \cdot logn] + Pr(X|X \leq 2 \cdot logn) \cdot E[X|X \leq 2 \cdot logn] \end{aligned}$$

If we bound the following terms with their maximum possible value as $Pr(X|X \leq 2 \cdot logn) = 1 - \frac{1}{n} \leq 1$ and $E[X|X \leq 2 \cdot logn] \leq 2 \cdot logn$ the previous equation takes the following form:

$$\begin{aligned} E[X] &\leq \frac{1}{n} \cdot E[X|X > 2 \cdot logn] + Pr(X|X \leq 2 \cdot logn) \cdot E[X|X \leq 2 \cdot logn] \Longrightarrow \\ &\leq \frac{1}{n} \cdot E[X|X > 2 \cdot logn] + 2 \cdot logn \end{aligned}$$

We know that the random variable X_i follows a geometric distribution since it probes until the first hit of empty bucket,so $X_i \sim Geo(p = 1 - \frac{i-1}{2n})$. Based on subquestion (a) the expected value of X_i is bounded by 2 since $Pr(X_i) = p \geq \frac{1}{2}$ so $E[X_i] = \frac{1}{p} \leq 2$. Hence based on the memoryless identity for geometric distributions we can imply that:

$$\begin{aligned} E[X|X > 2 \cdot logn] &\leq \sum_{i=1}^{n} E[X_i|X > 2 \cdot logn] \xrightarrow{memoryless} \\ &\leq \sum_{i=1}^{n} 2 \cdot logn + E[X_i] \Longrightarrow \\ &\leq \sum_{i=1}^{n} 2 \cdot logn + 2 \Longrightarrow \\ &\leq n \cdot (2 \cdot logn + 2) = O(n \cdot logn) \end{aligned}$$

Therefore the previous equation takes the following form:

$$\begin{aligned} E[X] &\leq \frac{1}{n} \cdot E[X|X > 2 \cdot logn] + 2 \cdot logn \Longrightarrow \\ &\leq \frac{1}{n} \cdot O(n \cdot logn) + 2 \cdot logn = O(logn) \end{aligned}$$

2 Constructions of hash functions

a)

We consider $n = 4$ and $p = 7$ so the $(7X6)$ table takes the following form, in which each value depicts the $h_a(x)$:

X\A	1	2	3	4	5	6
0	0	0	0	0	0	0
1	1	2	3	0	1	2
2	2	0	2	1	3	1
3	3	2	2	1	1	0
4	0	1	1	2	2	3
5	1	3	1	2	0	2
6	2	1	0	3	2	1

b)

Based on definition, in order to characterize a set H of hash functions is universal, every pair $w_1.w_2 \in U$ (in our case $U = \{0\} \cup [p-1] = 0, 1, 2, 3, 4, 5, 6$) and for h chosen uniformly from H it should be true that $Pr[h(w_1) = h(w_2)] \leq \frac{1}{n}$. However, in our case focusing on $h_3(x)$ and $h_4(x)$ hash map functions, we notice that for the pairs $(2, 3)$ and $(4, 5)$ the hash functions produce the same output. Based on that we can imply that out of the 6 different values of hash function 2 of them collide. Hence,

$$Pr[h_3(x) = h_4(x)] = \frac{2}{6} = \frac{1}{3} > \frac{1}{n} = \frac{1}{4}$$

Also, we need to point out that this is just two counter examples for those 6 hash functions. There are more, but in our case proving that h_3 and h_4 collide, is enough to prove that H is not universal.

c)

In order to answer the question we fix the parameters p, n such as $p \geq n$ and the values to hash as x and y. In particular,we care about x and y values that are different $x \neq y$. If there is a collision for x and y numbers for a specific hash function $h_a(\cdot)$ then it is true that:

$$\begin{aligned} Pr[h_a(x) = h_a(y)] &\Longrightarrow \\ (a \cdot x \mod p) \mod n = (a \cdot y \mod p) \mod n &\Longrightarrow \\ (a \cdot x \mod p) - (a \cdot y \mod p) \mod n = 0 \mod n &\Longrightarrow \\ v_1 - v_2 \mod n = 0 \mod n & \end{aligned}$$

Based on the previous equation we can see that $v_1 - v_2 = (a \cdot x \mod p) - (a \cdot y \mod p)$ is divisible by k. Each term of this subtraction is an integer taking values from $[0, p-1]$ due to modulo p. The subtraction of the two terms is an integer between the range $[-(p-1), p-1]$, so the total possible integers who are divisible from n in that range are at most $\frac{p-1-(-(p-1))}{n} = \frac{2 \cdot (p-1)}{n}$. However, we show that for a specific hash function. In particular, there are $p-1$ different hash functions. We need to show that each pair (v_1, v_2) corresponds to a unique hash function $h_a(\cdot)$. In order to do so, let's assume that there are two unique hash functions $h_{a_1}(\cdot)$ and $h_{a_2}(\cdot)$ and try to prove it wrong. If there are two hash functions for a unique pair of numbers (v_1, v_2) which collide to the same value then the following is true:

$$\begin{aligned} a_1 \cdot (x-y) \mod p = a_2 \cdot (x-y) \mod p &\Longrightarrow \\ (a_1 - a_2) \cdot (x-y) \mod p = 0 \mod p & \end{aligned}$$

Since $x \neq y$ and p is prime then in order to be true the previous equation $a_1 = a_2$ must hold. So we prove wrong that there are two distinct hash functions $h_{a_1}(\cdot)$ and $h_{a_2}(\cdot)$ but there is only one. Based on that there are p-1 choices for choosing a hash function, which at most $\frac{2 \cdot (p-1)}{n}$ of them collide x and y to the same value. So, $Pr[h_a(x) = h_a(y)] \leq \frac{2 \cdot (p-1)}{n} \cdot \frac{1}{p-1} = \frac{2}{n}$.

3 Using hashing for comparing multisets

a)

Based on the described algorithm, the algorithm makes an error when the counters of an element i do not match for S_1 and S_2 multisets. This could happened if the algorithm incorrectly defines the element i as

one of the other $n-1$ elements. Let define $E_y^{(i)}$ as the event of incorrectly defining element i as element y, where $y \neq i$, and $y \in [n]$. Assuming that hash functions are 2-universal the probability to hash incorrectly the element i as another element y is $Pr(E_y^{(i)}) \leq \frac{1}{c \cdot n}$. The probability Pr(Error) that element i is defined incorrectly is the union of $Pr(E_y^{(i)})$ for all possible elements y , so $Pr(Error) = Pr(\bigcup_{y \in [n], y \neq i} E_y^{(i)})$. Using the union bound we have:

$$Pr(Error) = Pr(\bigcup_{y \in [n], y \neq i} E_y^{(i)}) \leq \sum_{y \in [n], y \neq i} Pr(E_y^{(i)}) \leq \frac{n-1}{cn} \leq \frac{1}{c}.$$

A Monte Carlo algorithm is a randomized algorithm whose output may be incorrect with a certain (typically small) probability. In order to turn the above algorithm to a Monte Carlo with an even small probability δ, we will run it k times and output that the multisets S_1 and S_2 are the same if at least one iteration give this result. The only way to fail is if all iterations fail. We choose constant $c = 10$. Moreover, we define $\delta = \frac{1}{c}$ and $k = log_{10}(\frac{1}{\delta}) + 1$, and this would happen with a smaller error probability equal to $\delta^k = \delta^{log_{10}(\frac{1}{\delta})+1} = \delta^2 < \delta = \frac{1}{c}$.

The calculation of running time consists of the time spent for constructing the hash tables, compare them and run the iterations of the Monte Carlo algorithm. Constructing the hash tables, need indexing each one (O(1)) of n elements and increasing their counters, so $O(n)$. Also, comparisons for $c \cdot n$ counters need $O(c \cdot n) \approx O(n)$ running time and the total number of k iterations need $O(k \cdot n) \approx O(n)$. So the total running time is linear $O(n)$.

b)

As in the subquestion (a) the expected running time is calculated as the time spent for creating the hash table, hashing the numbers and comparing the two hash tables. For comparison of the cn counters of those two hash tables, we will need $O(c \cdot n) \approx O(n)$ time. We define as U_i the random variable describing the number of unique elements in each entry i of the hash table. For constructing the hash tables, we need running time to sort and increase the counters for all U_i in $c \cdot n$ buckets, which is at most equal to $\sum_{i=1}^{c \cdot n} U_i^2$. Based on the lecture about perfect hashing, we define as $C_{i,j}$ the indicator variable of having a collision between two elements i and j. So:

$$C_{i,j} = \begin{cases} 1 & \textit{,collision between i and j element} \\ 0 & \textit{,no collision} \end{cases}$$

Hence, the total collisions are defined as $C = \sum_{i,j \in [n]} C_{i,j}$. The expected value of C is calculated due to linear of expectation, the fact that the hash function is 2-universal and the fact that we have cn bins in total, the following:

$$E[C] \leq \binom{n}{2} \cdot \frac{1}{c \cdot n} \leq \frac{n}{2 \cdot c} \tag{2}$$

Therefore, the expected running time is bounded as follows:

$$\begin{aligned}
\sum_{i=1}^{c\cdot n} U_i^2 &= \sum_{i=1}^{c\cdot n} U_i^2 - U_i + U_i \Longrightarrow \\
&= \sum_{i=1}^{c\cdot n} U_i \cdot (U_i - 1) + U_i \Longrightarrow \\
&= \sum_{i=1}^{c\cdot n} 2 \cdot \binom{U_i}{2} + U_i \Longrightarrow \\
&= \sum_{i=1}^{c\cdot n} 2 \cdot \binom{U_i}{2} + U_i \Longrightarrow \\
&= \sum_{i=1}^{c\cdot n} 2 \cdot \binom{U_i}{2} + \sum_{i=1}^{c\cdot n} U_i \xrightarrow{(2)} \\
&\leq 2 \cdot \frac{n}{2 \cdot c} + n = \frac{n}{c} + n
\end{aligned}$$

In conclusion, the total time needed consists from a linear time for constructing the hash table, a linear time for hashing and sorting the elements and a linear time for comparing the counters, so in general the total time is linear.

About the space needed, the hash table has $c\dot{n}$ buckets, each one of them contains a linked list with at most n elements. So, in the worst case the space needed is $O(cn + n) \approx O(n)$.