



## [Solution] Quiz 2 Question 2

[Ragav Sachdeva](#)

[All sections](#)

Hi all,

I marked Q2, Quiz 2 of your submissions and I'd like to provide the solutions and some feedback.

**Q2.1:** A hash function,  $h(x)$ , hashes a name,  $x$  (the hash key), onto a hash value (the index into an array) as listed in the following table. The hash table has a length of 10 (indexed from 0 to 9).

The keys are inserted into a hash table in the order listed in the following table.

$y$	$h(y)$
Alice	3
Bob	6
Carol	2
Dave	3
Eve	7
Trent	6
Walter	0

Show the table contents after insertion with:

- chaining (2 marks)
- linear probing (2 marks)

Comments:

There are a few important things to point out:

- The question states that *the table* has length 10. It then asks you to show *the table* contents. Therefore, in your answers your table must have 10 slots (some of which may be empty). This is a bit pedantic and I have been lenient while marking as long as the items inserted were shown at the correct indices (even if some of you only showed 7 entries of the table as opposed to 10).
- The question also states that the keys are inserted in the order listed in the table provided in the question i.e. order of insertion is very specific which makes the outcome specific/deterministic.

## Chaining

## Linear probing

Index	Table value	Index	Table value
0	< Walter >	0	Walter
1	< >	1	⊥
2	< Carol >	2	Carol
3	< Dave, Alice >	3	Alice
4	< >	4	Dave
5	< >	5	⊥
6	< Trent, Bob >	6	Bob
7	< Eve >	7	Eve
8	< >	8	Trent
9	< >	9	⊥

Note that in case of collisions when using chaining, inserting the new element at both the head and the tail are acceptable.

Explanation of how chaining works: [link](https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/)  (https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/)

Explanation of how linear probing works: [link](https://www.geeksforgeeks.org/hashing-set-3-open-addressing/)  (https://www.geeksforgeeks.org/hashing-set-3-open-addressing/)

**Q2.2:** In lectures, on insertion into a skiplist we flipped a coin until a *head* occurred to determine the height of an element. So for example, if we flipped two tails before a head then we would insert an element of height 3. In all the examples in lectures the probability of flipping a head was exactly  $1/2$ .

Now assume that the coin is biased so the probability of flipping a head on a given throw is now  $9/10$ . Derive an expression for the probability of producing an element of height  $h$  with this biased coin. Describe in broad terms the consequence of this bias in terms of the average cost of insertion into the skiplist. (2 marks)

Comments:

For starters, the question asks for the *probability* of producing an element of height  $h$  and **not** the *expected height* of inserting an element into the list (which btw is different to *expected maximum height* of the entire skip list).

To insert an element at height  $h$ , you'd need to flip a tails on 1st, 2nd, 3rd ...  $(h-1)$ th tosses and then finally a heads on the  $h$ -th toss. The probability of doing that is:  $(0.1 \times 0.1 \times 0.1 \dots (h-1)\text{times}) \times 0.9 = 0.1^{(h-1)} \times 0.9$  (which is **not** the same as  $0.1 \times (h-1) \times 0.9$ ).


In fact, this is something called a [Geometric distribution](https://en.wikipedia.org/wiki/Geometric_distribution) and here is a snippet from Wikipedia:

The geometric distribution gives the probability that the first occurrence of success requires  $k$  independent trials, each with success probability  $p$ . If the probability of success on each trial is  $p$ , then the probability that the  $k$ th trial (out of  $k$  trials) is the first success is

$$\Pr(X = k) = (1 - p)^{k-1} p$$

In our case,  $p=0.9$ .

For the record, for those of you who tried to calculate the height of the skip list (which btw wasn't required), you were essentially computing the expected value of the geometric distribution which is  $1/p = 10/9$ . However, it is important to stress that this is the expected value of any given (single) insertion. Computing the expected maximum height of the list is a bit more involved. Here is a

[link to a resource](http://tuvalu.santafe.edu/~aaronc/courses/5454/csci5454_spring2013_L3.pdf)  ([http://tuvalu.santafe.edu/~aaronc/courses/5454/csci5454\\_spring2013\\_L3.pdf](http://tuvalu.santafe.edu/~aaronc/courses/5454/csci5454_spring2013_L3.pdf)) if you are interested (note you will have to change the value of  $p$  from 0.5 to 0.9 for our purposes).

Now for the second part of the question. A regular skip-list has an insertion cost of  $O(\log n)$ . In broad terms, the expected maximum height of the skip list will be smaller if we use the biased coin (for obvious reasons) and therefore, the skip list would look very similar to a linked-list which has an insertion cost of  $O(n)$ . Therefore, insertions would be much slower. Of course, this is a very hand-wavy answer (without rigorous mathematical proof) and that is ok. That is the expectation (the question says "in broad terms"). Also, just to clarify, the runtime of insertion when using a biased coin is not  $O(n)$  in case you were wondering. It is still  $O(\log n)$  but it has a different constant factor (specifically, the base of log is dependent on the value of  $p$ ). I won't provide the exact equation but it is quite straight-forward to figure out if you follow the resource I linked above.

**Q2.3:** State the storage requirements of a graph with  $n$  nodes and  $m$  edges using:

1. an adjacency list (1 marks), and
2. an adjacency matrix (1 marks)

briefly justify each answer. (2 marks)

Comments:

- It is important to realise that here you are being asked the space complexity. You should be talking in terms of Big-O and not bytes or megabytes (you cannot assume the data to be integers and assume 4bytes per vertex etc.). Also you should not be using  $n$  or  $m$  without Big-O i.e. you can't say the storage requirement is  $n^2$  for e.g. it would have to be  $O(n^2)$  (I didn't penalise anyone for it it though).
- If the question uses  $n$  and  $m$ , please just use  $n$  and  $m$ . Don't use  $|V|$  and  $|E|$  or something different (unless you explicitly define what  $|V|$  and  $|E|$  are in terms of  $m$  and  $n$ ). I did not penalise anyone for this (my job is to award marks and not take them away whatever opportunity I get) but it is important to mention.

For adjacency list, the answer is  $O(n+m)$ .

Some people said  $O(m)$  because "the number of values stored is proportional to the number of edges". While it is true, you need to think about the actual adjacency list data structure. An adjacency list is essentially an array of linked lists. You need  $O(n)$  space for just the array (that stores the pointers to the linked lists) and  $O(m)$  space for all the linked lists combined. Therefore, in total you need  $O(n+m)$  space (imagine if your graph has no edges i.e.  $m=0$  in that case you'd still need an array of size  $n$  where each element points to null).

For adjacency matrix, the answer is  $O(n^2)$ .

This is quite straight forward, you have an  $n \times n$  matrix which requires  $O(n^2)$  space.

As a side note,  $O(m)$  space would be correct if you were using the edge-list representation of a graph (which is different from adj list/matrix).

Some students discussed the advantages and disadvantages of using adj list vs adj matrix. This wasn't the intent of the question and while I personally appreciate your insights, I was unable to award any marks for it.

Another side tangent:

A very common statement I hear/read is "Adj list is more efficient for sparse graphs whereas Adj matrix is more efficient for dense graphs". What I don't hear very often is a concrete justification for it. As we established above, the space complexity for adjacency list is  $O(n+m)$ . When  $m$  is very small i.e. a sparse graph,  $O(n+m) < O(n^2)$  for obvious reasons. So adj list is preferred. However, in a dense graph, the maximum number of edges is still in the order of  $O(n^2)$  i.e. the space of complexity of adj list would be  $O(n+n^2)$  which is basically  $O(n^2)$ . Then how is it any better than adj matrix? But it has to be somehow, considering we hear it all the time, right?. The key is - the implementation. When we use adj matrix, each entry in the matrix can be represented using a single bit 0 or 1. This is not true for adj lists (you need more information than just a bit). Therefore, if the graph is dense, adj matrix just ends up being more space efficient (due to the differences in the constant factors of the Big-O complexities).

If you have any questions or if you spot a mistake in the solutions above, feel free to leave a comment to this announcement. Also, if you feel you have been unfairly marked (or I made a mistake) for Question 2 (which is possible considering there are a lot of students to mark), please reach out to me directly via my email. Replying to the comment on your submission will likely go unnoticed.

For any queries regarding Question 1, I'd encourage you to reach out to Guanhua Wang.

Thanks

**Unread**[← Write a reply...](#)