# Question 1:

**Which of the following are true statements regarding inbuilt library function that finds the length of a given string? (e.g. length())**

#### Select the correct choices:

Python/Java/C#/C++: It takes linear time to find length of a string every time, as you have to go thru each character.

Python/Java/C#/C++: It takes constant time to find length of a string every time, because length is cached at the time of String object creation and Strings are immutable after that.

C: It takes linear time to find length of a string every time, as you have to go thru each character.

C: It takes constant time to find length of a string every time, because length is cached at the time of String object creation and Strings are immutable after that.

# Answer 1:

Python/Java/C#/C++: It takes constant time to find length of a string every time, because length is cached at the time of String object creation and Strings are immutable after that. 

C: It takes linear time to find length of a string every time, as you have to go thru each character.

# Question 2:

**sub = original.substr(start, length) <==== Given that String objects are immutable in all modern languages, how does inbuilt library function Substring work in those languages?**

#### Select the correct choice:

Substring function creates a new String object. Every time. Every time you use Substring, you're taking a space complexity cost of O(N).

Substring function does not create a new object; just has pointers back to the original one. This is done for convenience, so that when the original one changes, the new one also changes.

Substring function breaks up the original string and points to the part it needs.

Substring function creates a new array (not a new String object), so that it can be manipulated as convenient.

# Answer 2:

Substring function creates a new String object every time. Every time you use Substring, you are taking a space complexity cost of O(N)

# Question 3:

**If L is the size of the given alphabet, and N is the size of the longest string in the input, what is the time complexity of Insert, Delete and Query operations on a Trie?**

#### Select the correct choice:

Insert: O(L), Delete: O(N), Query: O(N)

L, N, L

NL, NL, NL

LN, LN, LN (isn't this the same as the previous choice? Well, it is, just making sure you know that :-))

N, N, N (These 3 operations are independent of the size of the alphabet)

L, L, L (These 3 operations are independent of how long the input strings are)

N, N, L (Edits are dependent on the length of input strings, Query is dependent on the alphabet size)

L, N, N (reverse of the above choice)

# Answer 3:

N, N, N (These 3 operations are independent of the size of the alphabet)

# Question 4:

**Given a corpus of N strings, of length L each, what if we used a BST instead of a Trie, in order to store strings for efficient lookups? What would be the time complexity of inserting a new string in such a BST? Assume that it's a self-balancing BST.**

#### Select the correct choice:

O(L) \* O(L) Compare each incoming string to every other string in the BST. Each one is length L, so it's L^2.

O(N) Compare each incoming string to every other string in the corpus.

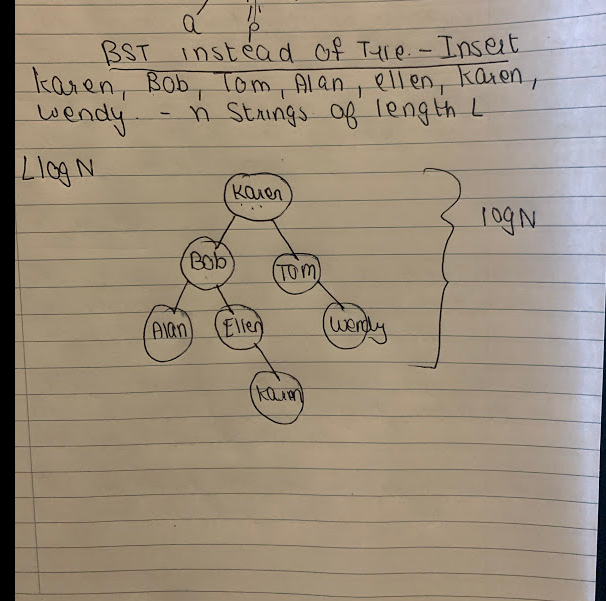
O(N \* L) Compare each incoming string to every other string in the corpus.

O(N \* log L) For every string, we need to make Log L decisions to find its place in the BST

O(L \* log N) Compare each incoming string to the given string in the tree node and make Log N such decisions to find its right place

# Answer 4:

O(L logN) Compare each incoming string to the given string in the tree node and make log N such decisions to find it’s right place.



# Question 5:

**Given a corpus of N strings, of length L each, what if we used a HashTable instead of a Trie, in order to store strings for efficient lookups? Which of the following are true statements?**

#### Select the correct choices:

HashTables are always faster than trees and tries.

Both Tries and HashTables have the same time complexity O(L) of insert, because in a HashTable, we need to compute a hashkey for each insert which takes O(L).

Even though both Tries and HashTables have the same complexity of insert [O(L)], all things equal, Tries are faster in implementation because we don't need to run the strings through a hashing function and handle collisions.

Which one is actually faster for our implementation, depends on what kind of queries we will get. e.g. If we simply need to find existence of a string, HashTables are faster. If however we need to find prefixes, then Tries are faster.

# Answer 5:

Both Tries and HashTables have the same time complexity of O(L) of insert, because in a HashTable, We need to compute a hashkey for each insert which takes O(L).

Even though both Tries and HashTables have the same complexity of insert [O(L)], all things equal, Tries are faster in implementation because we don’t need to run the strings through a hashing function and handle collisions.

Which one is actually faster for our implementation, depends on what kind of queries we will get. Eg. If we simply need to find existence of a string, HashTables are faster. If however we need to find prefixes, then Tries are faster.

# Question 6:

**What is one major concern with using a Trie as a solution?**

#### Select the correct choice:

It's very slow compared to all other data-structures that can be used for prefix-based lookups

It uses a lot of memory. Too many pointers at each node (O(L), where L is alphabet length). That is why compressed/radix Tries are useful

# Answer 6:

It uses a lot of memory. Too many pointers at each node (O(L), where L is alphabet length). That is why compressed/radix Tries are useful.

# Question 7:

**Given a corpus of N strings, maximum length L, which of the following is/are true?**

#### Select the correct choice:

If our predominant query is to find all strings matching a prefix, then we should use a self-balancing BST over a Trie

If our predominant query is simply to check the existence of a given String, then we should use a HashTable over a Trie

# Answer 7:

If out predominant query is simply to check the existence of a given String, then we should use a HashTable over a Trie.