Question 1

d)

First alternative:

We know that ip addresses are fragmented to classes :

Class A : 1 – 127

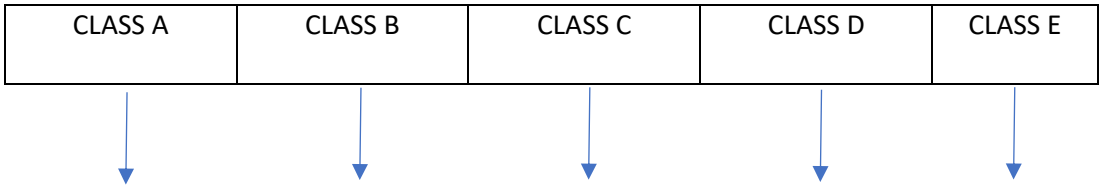
Class B : 128-191

Class C : 192 – 223

Class D : 224-239

Class E : 240 – 255

So, we can create an array of 5 indexes where each index hold a pointer to a root node that holds 16 pointers (like in question 1 c) :



If the ip address we want to add corresponds to Class B for example, then we got to index 1 in the array and make the same process as question 1 c. Same thing for other classes.

If we are looking for an ip address that is in class D for example, we go to the 4^{th} index in the array, and start looking for the ip address. (Same process that was presented in question 1 c).

Advantages :

- This method allows use to have a better control on our addresses. There are well organized and classified, so in O(1) , we can access the right corresponding tree and start the look up. If we want to check if the ip addresses of a particular class are existing , we already have a structure that allows use to return all of the existing addresses for this particular class.

Disadvantage :

- A lot of data is accumulated and there is a bigger need is memory.

Second Alternative :

We can create an array or vector of x indexes.

The more we need to add new ip addresses , the more the vector will resize and get bigger.

My intention is not to enter the ip address in the vector but rather :

(Please go to the next page)

Lets say that we start with a vector of size 10 :

0	1	2	3	4	5	6	7	8	9

Now , each Ip address will get a particular id code:

For example if we want to add : 192.144.12.3/32 A, this address will get an id of 3 (for the example only)

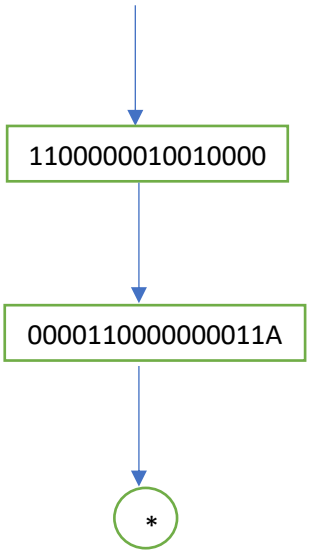
The id means that at index 3 , the ip address will be added as follows :

192.144.12.3 in binary : 11000000100100000000110000000011

We break it in 2 blocks of 16 bits : 1100000010010000.0000110000000011

Now lets add it :

0	1	2	3	4	5	6	7	8	9



In other words, every ip address is mapped with an integer.

Several ip addresses can have the same id number. That's why we end each ip address with a "*",

so that if another ip address gets the same id number, we add it after the * symbol.

After a certain number of times that one id number was affect to a set of ip addresses, we stop affecting it to other ip addresses and we increase the size of the vector and we continue the same process.

Advantages :

We broke the ip address in 2 parts, so we can check quickly enough if the ip address exists.

This method looks like hashmaping, the result is that it is very efficient and fast because we assign a particular integer to a set of ip addresses so that we can access them in a relatively short amount of time.

(Continue to next page please)

Disadvantages :

In case of many ip addresses, the vector could get bigger which needs a lot of memory allocation.

It is not easy to tell how many ip addresses have to be affected to a particular index.