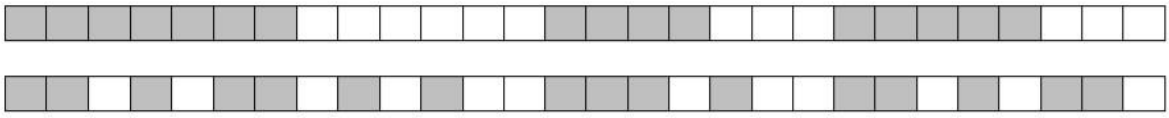
Question 1: Radix sort

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
Original array	513	145	320	235	141	433	2
UNITS	320	141	2	513	433	145	235
TENS	2	513	320	433	235	141	145
HUNDREDS	2	141	145	235	320	433	513
Sorted array	2	141	145	235	320	433	513

Question 2: Hash tables *(open addressing)*



1- The first hash table seems to have the issue of clustering happened to it. Indeed, we can clearly see the long blocks of data already hashed. This makes the cluster grow faster, which results in worsening the performance of future insertions. So, the second has table demonstrates a better

2- A hash table has 1000 slots and 200 items have already been hashed in it. What is the load factor? The load factor is **: α =** $\frac{items\ in\ the\ table}{table\ size}$ **=** $\frac{200}{1000}$ **= 0.2**

Question 3: Hash tables *(collision)*

1- Open addressing used : Double hashing
For the plus: we have a jump of size 3 from 5 to 8, because there's no wrapping around, the next probe should be: 8+3 = 11, the next one 11+3 = 14 … so on which is what happened.
Same thing check ***for the star,*** we have a jump of 4.

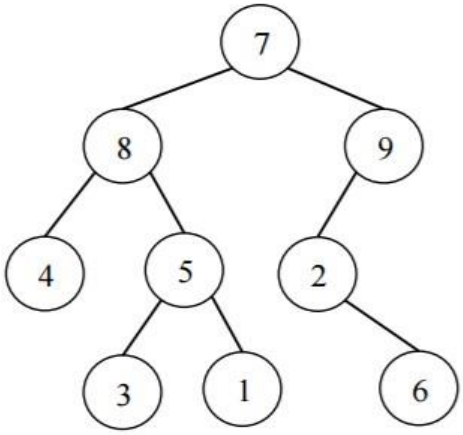
2- The next slot to be checked for the "plus" (if 14 is already occupied): **14+3 = 17**
The next slot to be checked for the "star" (if 17 is already occupied): **17+4 = 21**

3- Example of a bad hash function for strings: h(word, M) = L % M (where L is the length of the word and M is the size of the table)
Example, a hash table of size 10 to store the colors:

Blue : 4%10 = 4
Pink : 4%10 = 4
Grey : 4%10 = 4

Any words with the same length will create collisions no matter what the table size is!

Question 4: Tree order



Preorder(root, left , right) :

7, 8, 4, 5, 3, 1, 9, 2, 6

Inorder(left, root, right) :

4, 8, 3, 5, 1, 7, 2, 6, 9

Postorder(left, right, root) :

4, 3, 1, 5, 8, 6, 2, 9, 7

Question 5 : Trees

Write a function that takes three arguments: a binary tree, an array that has the items from the tree in a certain order (preorder, inorder or postorder) and the length of the array. The function should identify the order and print its name: “preorder”, “inorder” or “postorder”. The function must run in O(1). (No credit will be given if it does not run in O(1). You can assume that: - There are no duplicates in the tree (no two items are equal). - The tree has at least three items and the left and the right subtrees are NOT empty (there are one or more nodes in the left subtree and one or more nodes in the right subtree).

```
typedef struct node *link;

struct node {
Item item;
 link left;
 link right; };

void which_order(link h, int* array_of_items, int size_of_array)
{
        If (h->item == array_of_items [0])
                Printf(“Preorder \n”);

        Else if (h->item == array_of_items [size_of_array - 1])
                Printf(“Postorder \n”);
        Else
                Printf (“Inorde r\n”);
}
```