Question - 1

Table

SCORE: 10 points

Considering a hash table, of which implementation is based on Array and Linked Node. So what is the time complexity to find a certain comparable element from it?
N is the number of elements, and m is the length of the array.

○ o(N)

● O(N/m)

○ O(logN -logm)

○ O(m)

Question - 2

Linear Probing

SCORE: 10 points

Suppose we are using Hash(k) = 3 * k % 13, and an array of size 13 as a Hash Table, what's the result after putting the below number into the hash table if we use linear probing? (* represent there is no value in the hash table) Number in order: 22 -> 40 -> 36 -> 55 -> 24 -> 27 -> 28

○ * 22 * 40 36 27 * 24 * 55 28 * *

○ 22 * 40 36 27 28 24 * 55 * * * *

○ 22 * 27 36 28 * 24 * 55 * * * *

● * 22 * 40 36 27 28 24 * 55 * * *

○ * 22 * 40 27 36 * 24 * 55 * * *

○ * 22 * 27 36 28 * 24 * * * * *

○ * 22 * 40 36 * * 24 * 55 * * *

Question - 3

Output

SCORE: 10 points

```
public class Output {

        static class A{
                int v;
                protected int get() {
                        int tmp = v++;
```

Help

```
                        return --tmp;
                }
                protected A(int v){
                        this.v=v;
                }
        }

        static class B extends A{
                int v =2;

                protected B(int v) {
                        super(v);
                }

                protected int get() {
                        return v;
                }
        }

        public static void main(String[] args) {
                // TODO Auto-generated method
stub

                A o = new B(5);
                System.out.println(2 << o.get());

        }
```

○ 16

● 8

○ 4

○ Exception

Question - 4
Implementation

SCORE: **20 points**

Considering simulating a file system, there are two different kinds of elements which are directories (folders) and files. A **directory** is able to contain subdirectories and files.
Let's say this system supports creation and deletion only.
Given a list of creation commands in the form of ["operation, "target", "type"] and deletion commands in the form of ["operation, "target"],
After operations, print all elements in your system, from root to bottom, depth by depth. And elements in each directory should be printed **in dictionary order**.
After printing all files of the same depth, print "/n". Use **commas** to separate directories and files of the same depth.

*Print() has **already** been implemented, do not modify it.*

Validations are needed. For deletions, the element must **exist**. For creations, the directories must be **valid and no duplicate** file names are in the directory.
If an operation is invalid. **drop** all operations left including invalid one.

For example:

Given {["creation", "root/tests","dir"],
["creation","root/tests/log_19","file"],
["creation","root/tests/log_11","file"]}
output:

```
   root
   root/tests
   root/tests/log_11,root/tests/log_19
```