Lab 4 Report

Name	ID
Abdelmoneim Hany Abdelmoneim Mohamed	19017359
Youssef Ahmed Saeed	19016903

Description:

Our Program Implements the Perfect Hashing using 2 methods of different scape complexities:

- O(N)
- O(N^2)

Implementation:

- We have a class for each method
    - Method1
    - Method2
- Each takes input array as a parameter passed to constructor
- Each uses the universal hash family: matrix method to store elements
- Method1:
    - Generates the random hash matrix of size log(n^2)
        - n: number of elements in input array
    - inserts the n elements in the hash table of size n^2 one by one
        - get index I hash table by doing matrix multiplication between hash matrix and passed key
        - Done efficiently using bitwise operations
        - In case of collision, we repeat steps and rehash all elements with a new hash matrix till no collision takes place
    - Final hash table of size n^2 contains all elements with zero collisions
    - Searching for specific key:
        - Use matrix and key to get index
        - Checks if required key exists at this index
- Method2:
    - Generates the random hash matrix of size log(n)
        - n: number of elements in input array
        - Called main hash matrix
    - Firstly, inserts all elements in a hash table of size n using same technique of method1 in generating hash indices:
        - If collision takes place m times at index I:

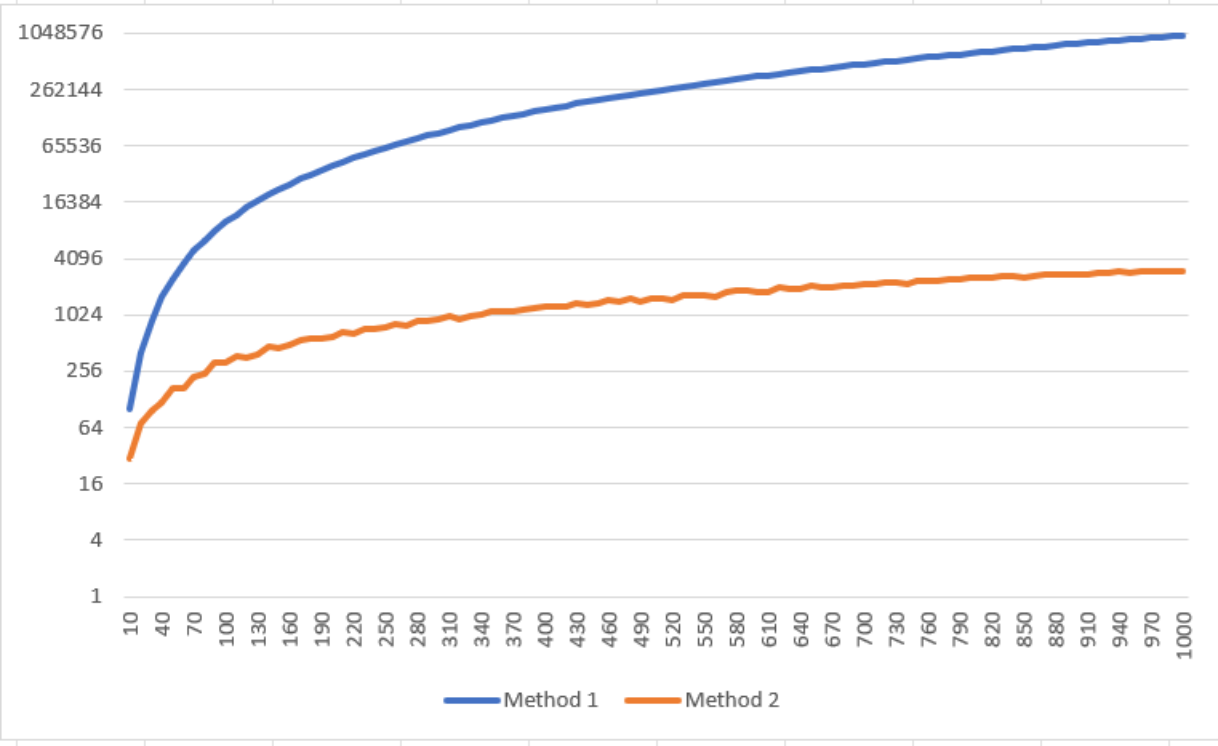
- Call it case x
- We apply method1 on the array of collision elements
- We receive from method1:
    - hash table of size m^2 having m non-zero elements
    - the hash matrix used to get indices in hash table
    - Number of collisions that took place
- Save these data structures in method2 arrays
- We skip method1 part if m=1 as we only got one element and its index in hash table is 0 inevitably (case xi)

- Searching for specific key:
    - Case x:
        - Use main matrix in generating first index
        - Get inner hash table of size m^2
        - In case xi:
            - We access first key
        - Else:
            - We use the corresponding hash matrix of current table to get second index
            - Access current key
        - Either way, we check if reached location contains required key
    - Else:
        - Same as method1

Analysis:

When you run main method in Driver class you'll get 100 test cases with input ranging from 10 to 1000 with step 10. Showing each time number of input, space used (Hash Table Entries) and number of collisions. Also there is an excel file containing space complexity related data and a graph verifying the space complexity of each method.

Graph below uses log Scale.



We can see Method 1 uses much more space O(N^2) while Method 2 uses O(n).