DataStructure2 Lab4

NAME	ID
Antoine Talaat Zaki	19015486
Amr Magdy Mahmoud Hanafy	19016116

Requirements Covered:

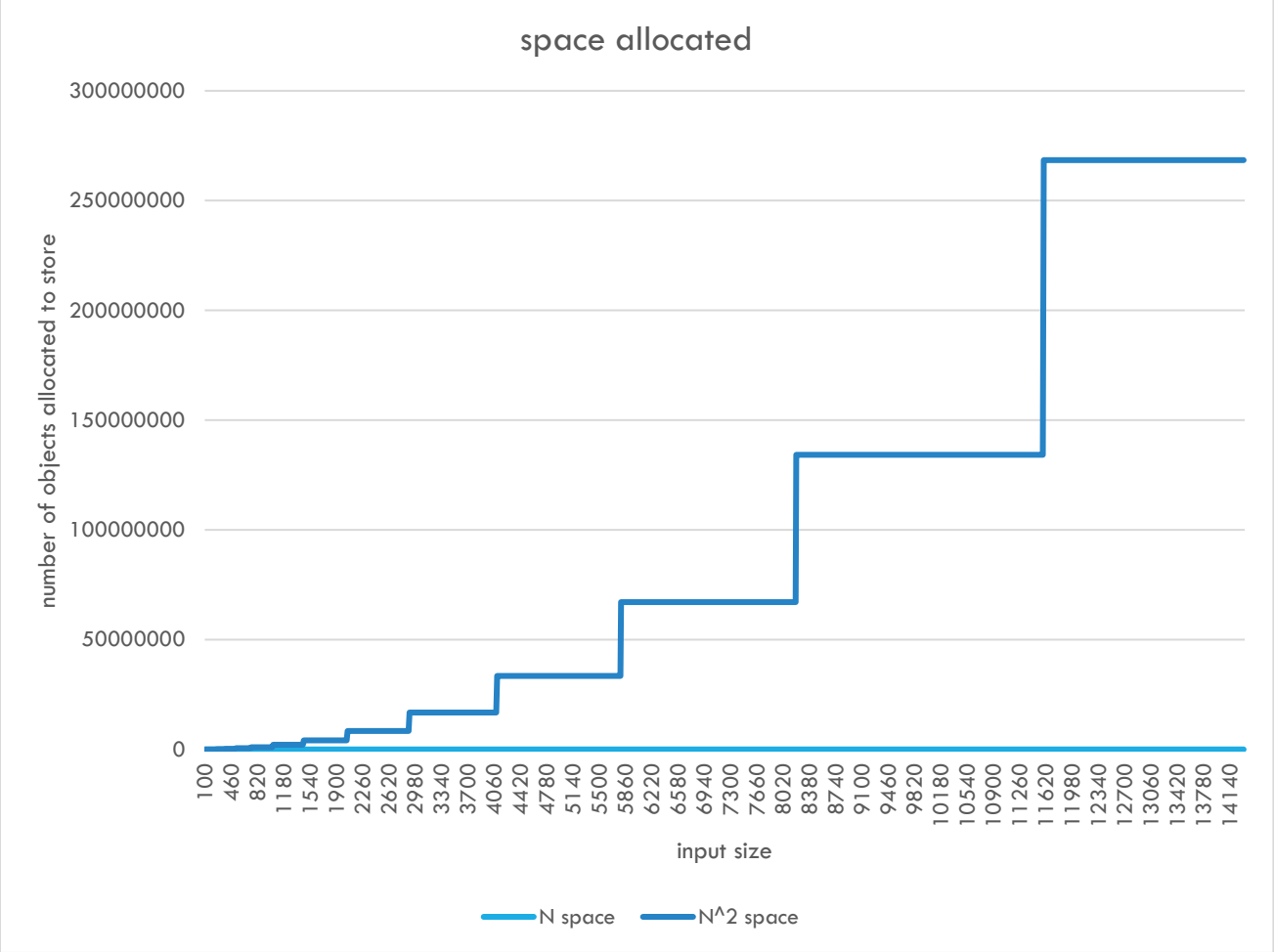
- Implementing Perfect hashing class using N^2 method and N method.
- Implementing universal hashing using matrix method.
- Having the Hash table store generic objects (in tests integer objects are used).

Implementation Details:

- Universal Hashing:
  - Makes a random matrix of bits (implemented as Booleans). Its size depends on the number of bits in the keys and the number of bits in the hashed result (depends on size of input).
  - The class converts keys to binary representation and does matrix multiplication to get the hashed value.
  - Hashed value is index to be used to access the array of Elements.
  - Key bits are passed 32 bits from the creator of that object (given)
- Element class:
  - Each key-value pair are stored inside Class Element.
  - Key value is always integer.
  - Value is generic type.
- PerfectHashingN2Solution:
  - Handles the n^2 method.
  - Contains insert method that inserts to the given size.

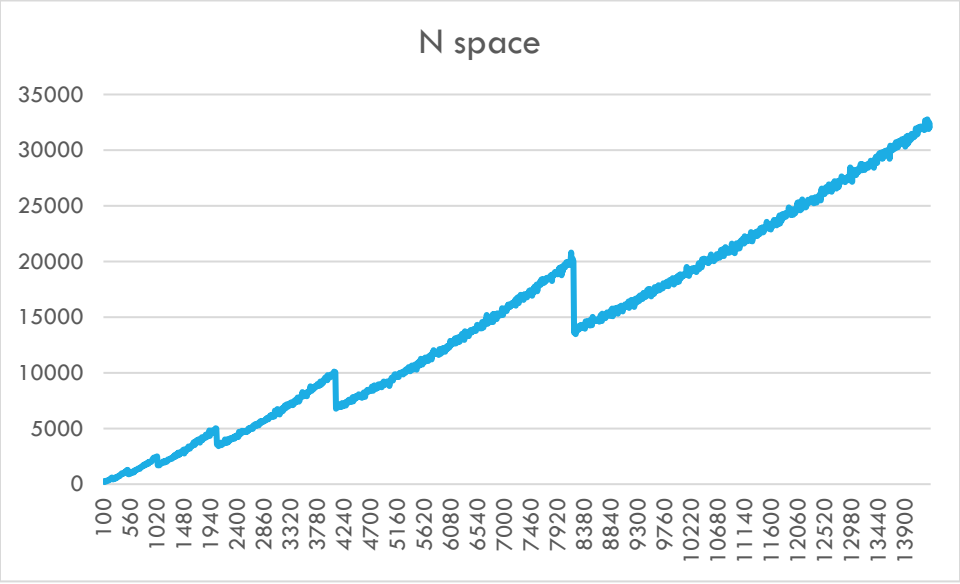
    - Contains insert batch method that inserts a list to the hash table.
    - Contains method called rehash that replaces the storage and creates new object of universal hashing class and rehash all elements, returns Boolean true when successful false otherwise.
    - Contains insert Additional method that inserts beyond given size (will cause creating new storage and rehashing).
    - ASSUMPTION: if we need storage for 9 elements, binary representation ranges from 0000 to 1000 which takes 4 bits. Since we can't guarantee the hash value resulted from the matrix multiplication is in that range (hash value may be 1111) so we allocate 16 places for that size. This can unused memory but will decrease number of rehashing.
    - Also contains function to print hash table contents.
- PerfectHashingNSolution:
    - Uses objects of perfectHashingN2Solution in each bucket.
    - Contains methods for inserting too.
- Solution Class:
    - Demo class for trying either input through user (exhaustive).
    - Using random key generator to generate input of given size.
- Analyzer Class:
    - Appends to files to collect data.
    - Runs the two methods with multiple sizes ranging from 100 to 15000 with step size 10 (used these for the graphs in the analysis below. Heap got full at certain point so we can decrease the limit if necessary).

Analysis Results:



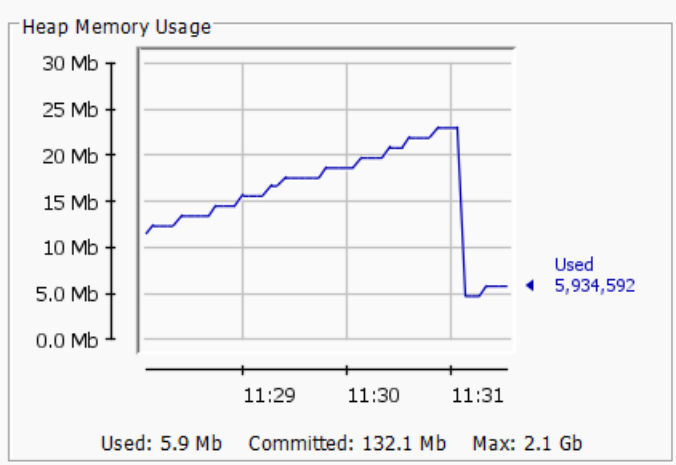
Here is a graph showing comparison between both methods in terms of size, the input grows very high that is why we can n curve compressed.

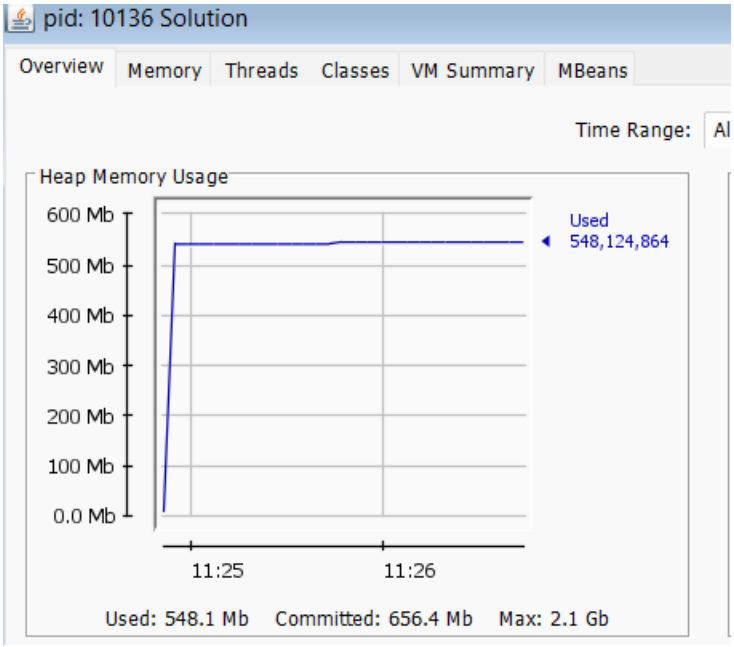
Curve of n looks like this



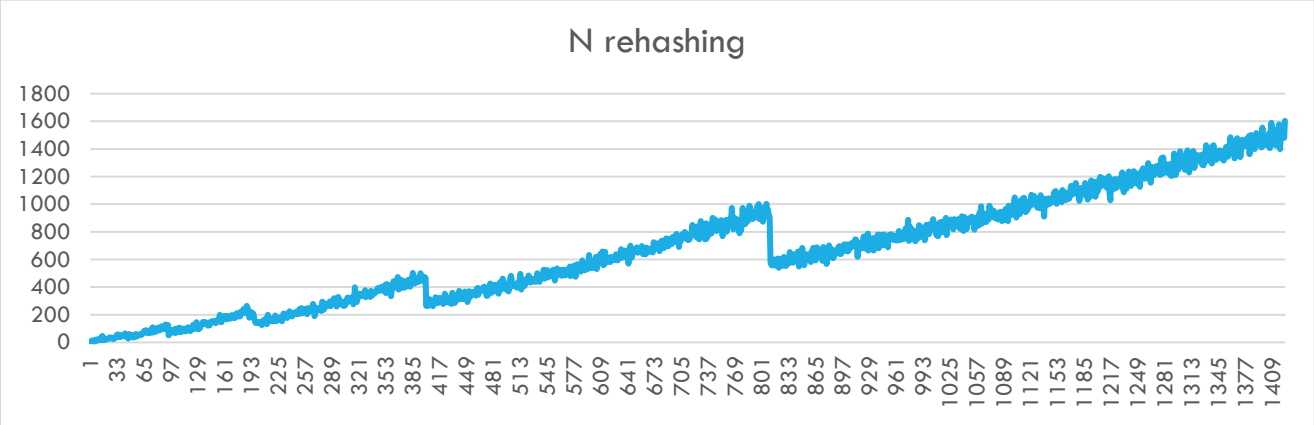
Analysis Notes:

- The results of the previous graphs shows the number of objects allocated with respect to object size.
- Another type of analysis is trying to track input size of 10k and watching the heap in progress by jconsole.
- Here n method uses maximum of 23Mb approximately. While n^2 method takes about 529Mb which is almost the square of the size in the previous method.





- It was noticed that usually the number of rehashing in the n^2 method is ranging from 0 to 6 but usually in between (0.469309463 on average).
- Number of rehashing is more in the n method because with each collision we reset the bucket.
- Graph of number of rehashings done:
- Graph shows that rehashings are proportional to input size as number of collisions increase.



Thank you