Data Structures 2

Lab 3
Perfect Hashing

Java implementation

Aya Sameh	*1*
Dahlia Chehata	*28*
Salma Hesham	*34*

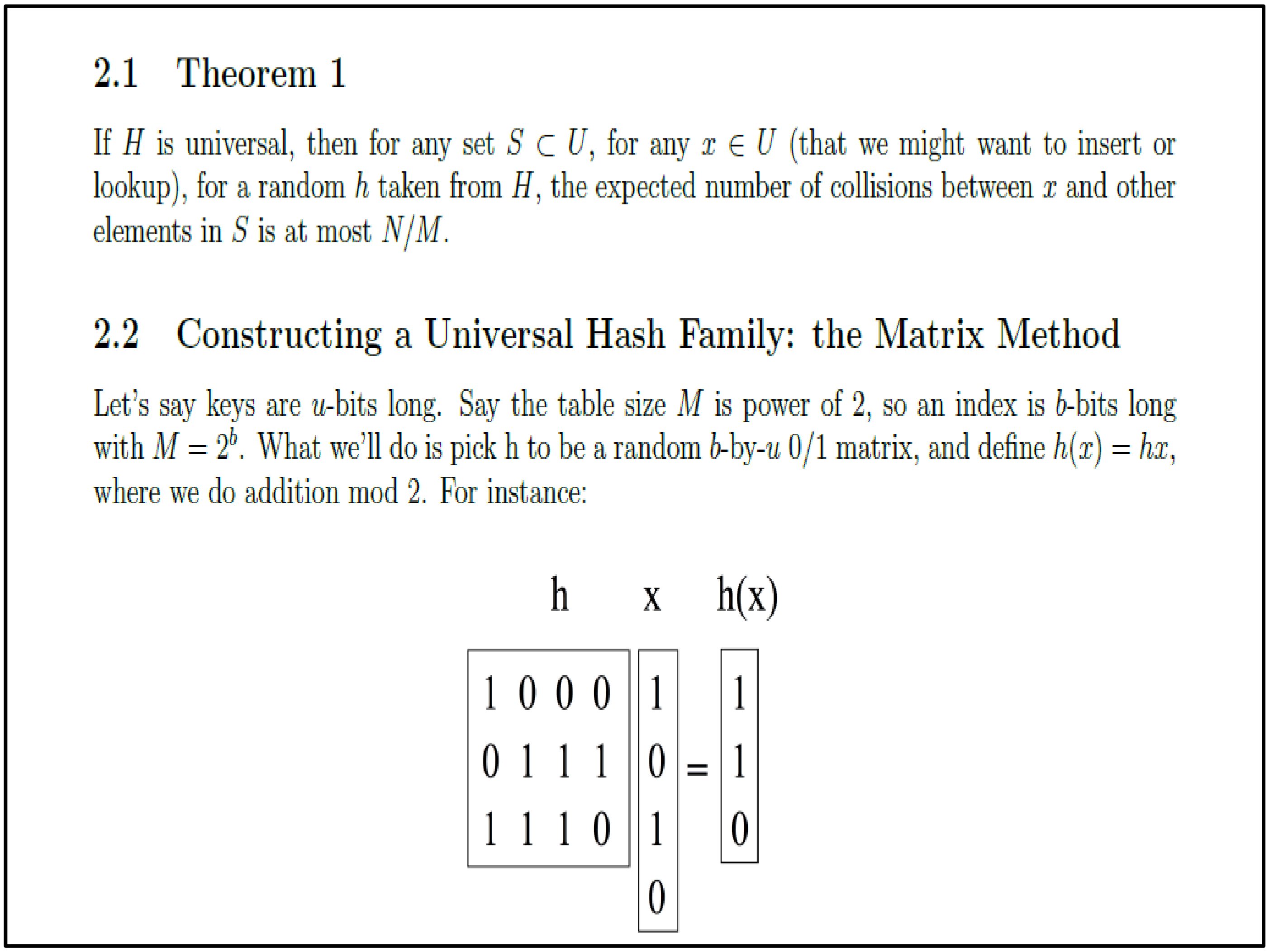
1. **Problem Statement:**

It is required to implement a perfect hashing data structure, to design, analyze and implement a perfect hash table.

Universal Hashing:

A probability distribution H over hash functions from U to $\{1, ..., M\}$ is universal if for all $x \neq y$ in U, we have

$$Pr[h(x) = h(y)] \leq 1/M \qquad (1)$$



2.1 Theorem 1

If H is universal, then for any set $S \subset U$, for any $x \in U$ (that we might want to insert or lookup), for a random h taken from H, the expected number of collisions between x and other elements in S is at most N/M.

2.2 Constructing a Universal Hash Family: the Matrix Method

Let's say keys are u-bits long. Say the table size M is power of 2, so an index is b-bits long with $M = 2^b$. What we'll do is pick h to be a random b-by-u $0/1$ matrix, and define $h(x) = hx$, where we do addition mod 2. For instance:

$$\overset{\text{h}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}} \overset{\text{x}}{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}} = \overset{\text{h(x)}}{\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}}$$

O(N) space:

It is required to have a table whose size is quadratic in the size N of dictionary S.
Let H be universal and M = N_2. Pick a random h from H and
try it out, hashing everything in S. So, we just try it, and if we got any collisions, we just try a new h. On average, we will only need to do this twice.

O(N^2) space:

The main idea for this method is to use universal hash functions in a 2-level scheme.
The method is as follows. We will first hash into a table of size N using universal hashing.
This will produce some collisions. However, we will then rehash each bin using Method 1, squaring the size of the bin to get zero collisions. So, the way to think of this scheme is that we have a first-level hash function h and first-level table A, and then N second-level hash functions h1,…., hN and N second-level tables A1; :::;AN . To lookup an element x, we first compute i = h(x) and then find the element in Ai[hi(x)].

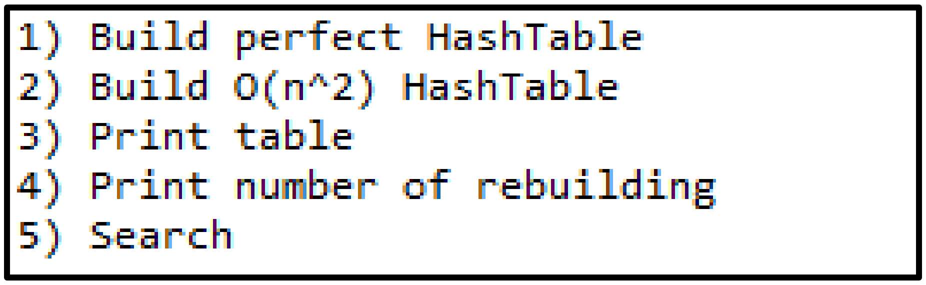
Structure, design, Description details, data structures used:

1.Structure:

- There are 2 main packages **HashingMethods** and **Reader**
- **HashingMethods package** contains an abstract class **PerfectHashing** which is extended in 2 main classes **PerfectHashingN** responsible for the O(N) space hash table and **PerfectHashingN2** responsible for the O(N^2) space hashtable.
- The **UniversalHashing** class is the one responsible for generating the hash functions
- Class **Main** is the main program and user interface.
- Class reader in package **Reader** responsible for the file handling

2. Design , description details and data structures:

- The main program has a friendly user interface. It reads the file and allow the user to select an option from those below



```
1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
```

- The abstract class PerfectHashing contains 3 principal objects

```
public abstract class PerfectHashing {
 protected UniversalHashing function;
 protected int RebuildingTimes;
 protected ArrayList<Integer> items;
 }
```

And allows to build table, search in the hash table by index, get has

Table, get the rebuilding times

- The concrete classes PerfectHashingN and PerfectHashingN2 extend PerfectHashing abstract class and overwrite the implements methods

- PerfectHashingN class is the main class responsible for the perfect hashing. It generates the main hash table and call the PerfectHashingN2 instance

  In case of rebuilding to rebuild a new table in the required slot of O(N*N) space .

```
private void generateNSquaredArrays() {
		ArrayList<Integer> temp;

		for (int i = 0; i < items.size(); i++) {
			temp = (ArrayList<Integer>) hashingArray[i];
			if (temp != null) {
				nSquaredArray = new PerfectHashingN2(temp);
				nSquaredArray.setMaxDataLength(maxWordLength);
				try {
					nSquaredArray.buildTable();
					RebuildingTimes +=
				 nSquaredArray.getRebuildingTimes();
				} catch (Exception e) {
//					e.printStackTrace();
				}
				table[i] = nSquaredArray;
			}
		}
	}
```

- PerfectHashingN2 class is responsible for generating the table in each slot Of order O(N*N) space in case of collision in order to achieve the perfect hashing with zero collision.

  .

- The universalHashing class responsible for generating the hash functions It generates first the random universal matrix and the function H maps each element to the equivalent index in the hash table

- The hash table itself is an array of size N and each slot in case of collision is extended to another table of size N *N

- The reader class reads the given file, sets its name and get the file name as well as the keys

Code Snippet:

Class UniversalHashing:

```java
public void generateFunction(int tableSize, int maxDataLength) {
        this.tableSize = tableSize;
        cols = maxDataLength; // max no of bits of x
        rows = (int) Math.ceil(Math.log(tableSize) / Math.log(2));
        // b = Log2(M)
        if (rows < 1) {
            rows = 1;
        }
        matrix = new int[rows][cols];
        Random randomGenerator = new Random();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = randomGenerator.nextInt(2);
                // max value 1
            }
        }
    }
```

```java
private int multipleFunction(int[] binaryArray) {
        int[] multipliedBinaryNumber = new int[rows];
        int convertedNumber =0;
        int sum;
        int mul;
        for (int i = 0; i < rows; i++) {
            sum = 0;
            for (int j = 0; j < cols; j++) {
                mul = matrix[i][j] * binaryArray[j];
                sum = Math.floorMod(sum + mul, 2);
            }
            multipliedBinaryNumber[i] = sum;
        }
        String multipliedNumberString = toString(multipliedBinaryNumber);
        convertedNumber = Integer.parseInt(multipliedNumberString, 2);
        if (convertedNumber == 0) { // cannot make mod 0
            return convertedNumber;
        }
        System.out.println("\n >>>>>>> converted no >>>> " +
        convertedNumber);
        convertedNumber = Math.floorMod(convertedNumber, tableSize-1);
        return convertedNumber;
    }
```

PerfectHashingN class:

```
@Override
    public void buildTable() throws Exception {
        if (items.size() == 0) {
            throw new Exception("The list is empty!");
        }
        ArrayList<Integer> collisions;
        function = new UniversalHashing();
        function.generateFunction(items.size(), maxWordLength);
        for (int n : items) {
            int index = function.h(n);
            if (hashingArray[index] != null) {
                collisions = hashingArray[index];
            } else {
                collisions = new ArrayList<>();
            }
            collisions.add(n);
            hashingArray[index] = collisions;
        }
        generateNSquaredArrays();

    }
```

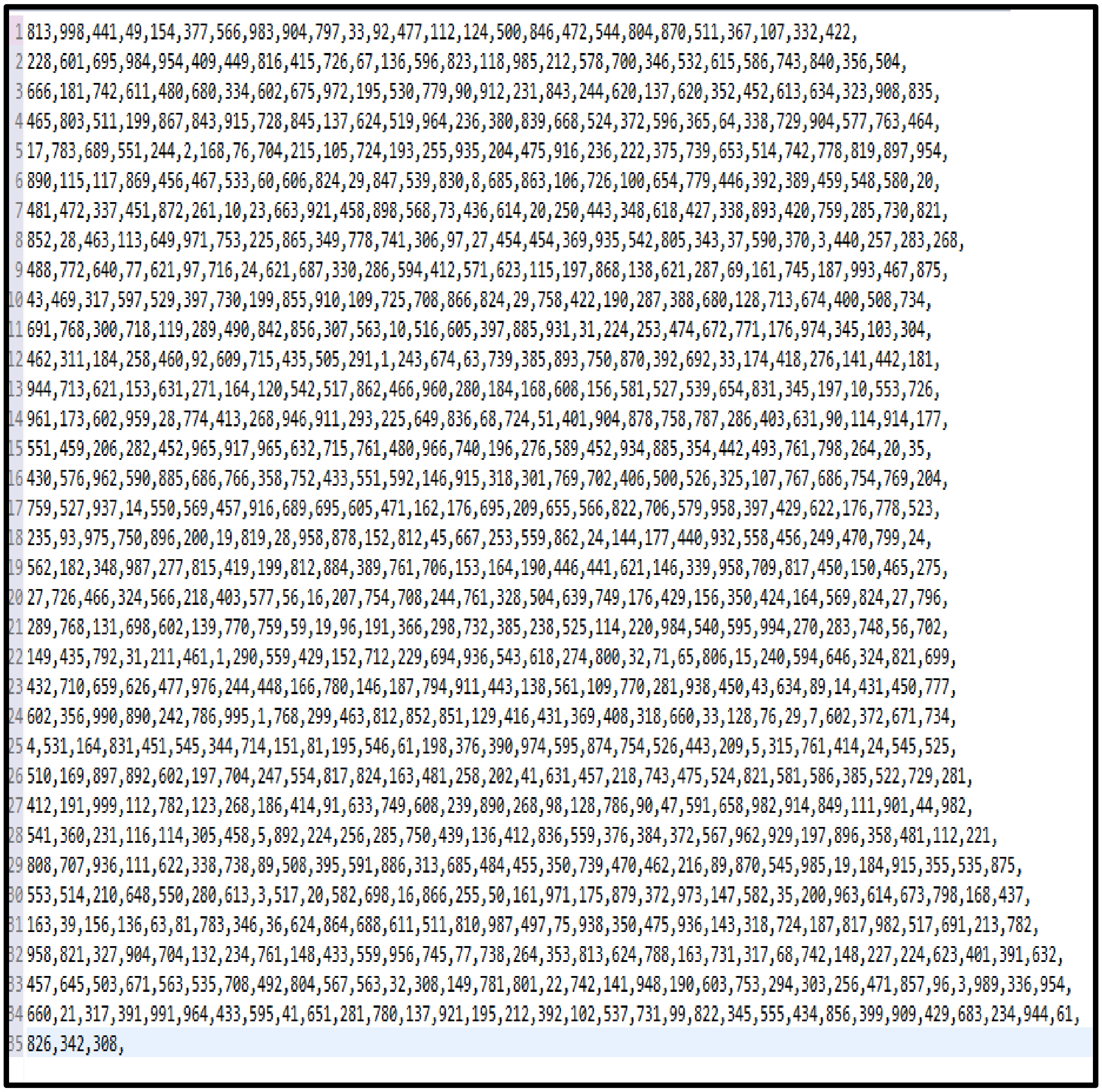
PerfectHashingN2 class:

```
@Override
    public void buildTable() throws Exception {
    if (items==null)
        throw new Exception("Empty list Exception");
      boolean collision;
      function=new UniversalHashing();
      int n =items.size();
      int index,cnt=0;
      do {
          cnt++;
          table =new Integer[n*n];
          collision=false;
          function.generateFunction(n * n, maxDataLength);
          for (int i=0;i<items.size();i++){
              index=function.h(items.get(i));
              if (table[index]!=null && (int)table[index]!=items.get(i)){
              collision=true;
              RebuildingTimes++;
              cnt = 0;
              break;
          }
          table[index]=items.get(i);
          }
      }while (collision && cnt<items.size());
      items=null;

    }
```

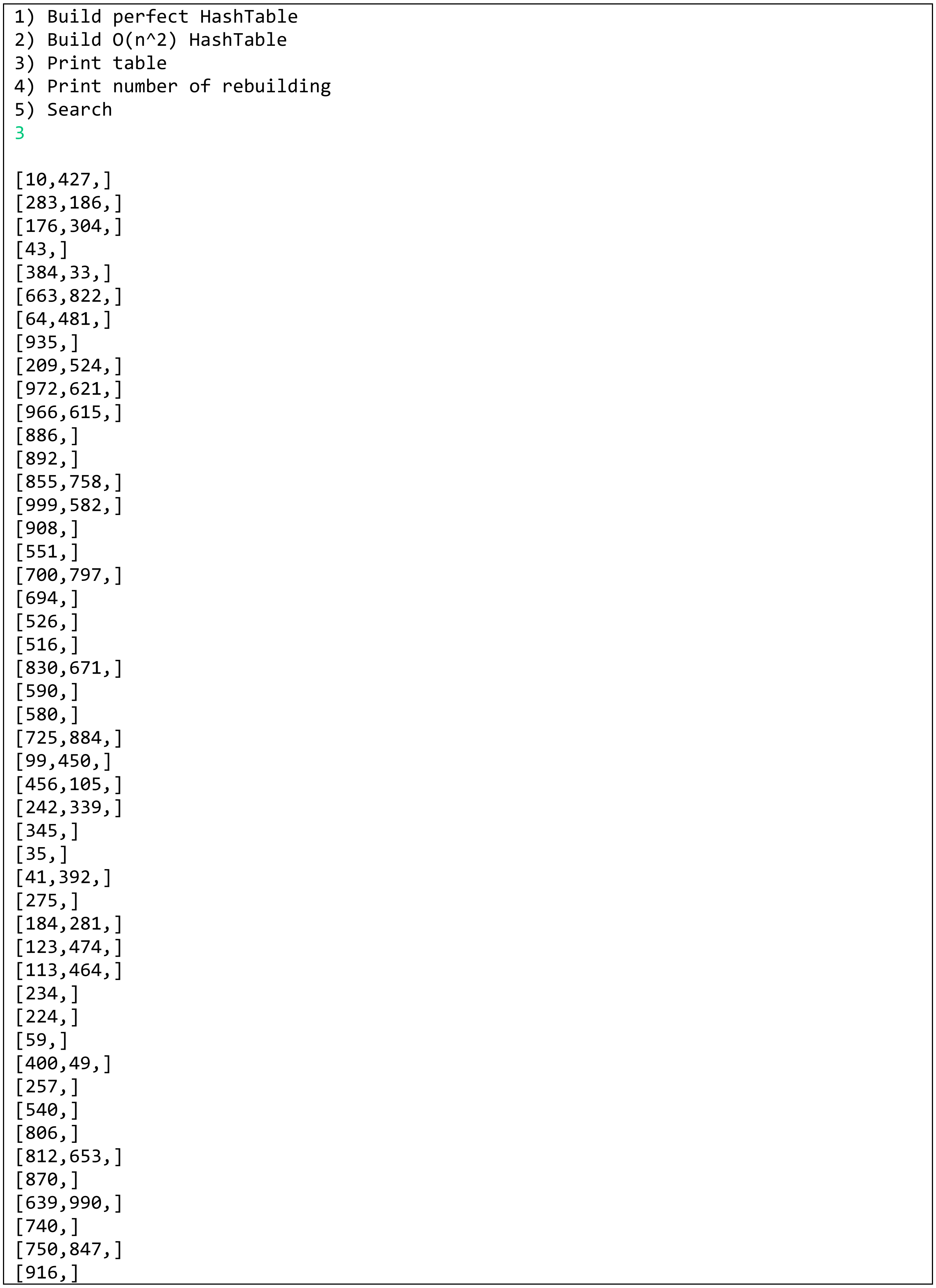
Sample runs:

Taking the file as input:



```
813,998,441,49,154,377,566,983,904,797,33,92,477,112,124,500,846,472,544,804,870,511,367,107,332,422,
228,601,695,984,954,409,449,816,415,726,67,136,596,823,118,985,212,578,700,346,532,615,586,743,840,356,504,
666,181,742,611,480,680,334,602,675,972,195,530,779,90,912,231,843,244,620,137,620,352,452,613,634,323,908,835,
465,803,511,199,867,843,915,728,845,137,624,519,964,236,380,839,668,524,372,596,365,64,338,729,904,577,763,464,
17,783,689,551,244,2,168,76,704,215,105,724,193,255,935,204,475,916,236,222,375,739,653,514,742,778,819,897,954,
890,115,117,869,456,467,533,60,606,824,29,847,539,830,8,685,863,106,726,100,654,779,446,392,389,459,548,580,20,
481,472,337,451,872,261,10,23,663,921,458,898,568,73,436,614,20,250,443,348,618,427,338,893,420,759,285,730,821,
852,28,463,113,649,971,753,225,865,349,778,741,306,97,27,454,454,369,935,542,805,343,37,590,370,3,440,257,283,268,
488,772,640,77,621,97,716,24,621,687,330,286,594,412,571,623,115,197,868,138,621,287,69,161,745,187,993,467,875,
43,469,317,597,529,397,730,199,855,910,109,725,708,866,824,29,758,422,190,287,388,680,128,713,674,400,508,734,
691,768,300,718,119,289,490,842,856,307,563,10,516,605,397,885,931,31,224,253,474,672,771,176,974,345,103,304,
462,311,184,258,460,92,609,715,435,505,291,1,243,674,63,739,385,893,750,870,392,692,33,174,418,276,141,442,181,
944,713,621,153,631,271,164,120,542,517,862,466,960,280,184,168,608,156,581,527,539,654,831,345,197,10,553,726,
961,173,602,959,28,774,413,268,946,911,293,225,649,836,68,724,51,401,904,878,758,787,286,403,631,90,114,914,177,
551,459,206,282,452,965,917,965,632,715,761,480,966,740,196,276,589,452,934,885,354,442,493,761,798,264,20,35,
430,576,962,590,885,686,766,358,752,433,551,592,146,915,318,301,769,702,406,500,526,325,107,767,686,754,769,204,
759,527,937,14,550,569,457,916,689,695,605,471,162,176,695,209,655,566,822,706,579,958,397,429,622,176,778,523,
235,93,975,750,896,200,19,819,28,958,878,152,812,45,667,253,559,862,24,144,177,440,932,558,456,249,470,799,24,
562,182,348,987,277,815,419,199,812,884,389,761,706,153,164,190,446,441,621,146,339,958,709,817,450,150,465,275,
27,726,466,324,566,218,403,577,56,16,207,754,708,244,761,328,504,639,749,176,429,156,350,424,164,569,824,27,796,
289,768,131,698,602,139,770,759,59,19,96,191,366,298,732,385,238,525,114,220,984,540,595,994,270,283,748,56,702,
149,435,792,31,211,461,1,290,559,429,152,712,229,694,936,543,618,274,800,32,71,65,806,15,240,594,646,324,821,699,
432,710,659,626,477,976,244,448,166,780,146,187,794,911,443,138,561,109,770,281,938,450,43,634,89,14,431,450,777,
602,356,990,890,242,786,995,1,768,299,463,812,852,851,129,416,431,369,408,318,660,33,128,76,29,7,602,372,671,734,
4,531,164,831,451,545,344,714,151,81,195,546,61,198,376,390,974,595,874,754,526,443,209,5,315,761,414,24,545,525,
510,169,897,892,602,197,704,247,554,817,824,163,481,258,202,41,631,457,218,743,475,524,821,581,586,385,522,729,281,
412,191,999,112,782,123,268,186,414,91,633,749,608,239,890,268,98,128,786,90,47,591,658,982,914,849,111,901,44,982,
541,360,231,116,114,305,458,5,892,224,256,285,750,439,136,412,836,559,376,384,372,567,962,929,197,896,358,481,112,221,
808,707,936,111,622,338,738,89,508,395,591,886,313,685,484,455,350,739,470,462,216,89,870,545,985,19,184,915,355,535,875,
553,514,210,648,550,280,613,3,517,20,582,698,16,866,255,50,161,971,175,879,372,973,147,582,35,200,963,614,673,798,168,437,
163,39,156,136,63,81,783,346,36,624,864,688,611,511,810,987,497,75,938,350,475,936,143,318,724,187,817,982,517,691,213,782,
958,821,327,904,704,132,234,761,148,433,559,956,745,77,738,264,353,813,624,788,163,731,317,68,742,148,227,224,623,401,391,632,
457,645,503,671,563,535,708,492,804,567,563,32,308,149,781,801,22,742,141,948,190,603,753,294,303,256,471,857,96,3,989,336,954,
660,21,317,391,991,964,433,595,41,651,281,780,137,921,195,212,392,102,537,731,99,822,345,555,434,856,399,909,429,683,234,944,61,
826,342,308,
```

```
>> Read file [keys10001000] sucessfully
 --------------------------------------
1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
1
>> HashTable built successfully
```



```
1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
3

[10,427,]
[283,186,]
[176,304,]
[43,]
[384,33,]
[663,822,]
[64,481,]
[935,]
[209,524,]
[972,621,]
[966,615,]
[886,]
[892,]
[855,758,]
[999,582,]
[908,]
[551,]
[700,797,]
[694,]
[526,]
[516,]
[830,671,]
[590,]
[580,]
[725,884,]
[99,450,]
[456,105,]
[242,339,]
[345,]
[35,]
[41,392,]
[275,]
[184,281,]
[123,474,]
[113,464,]
[234,]
[224,]
[59,]
[400,49,]
[257,]
[540,]
[806,]
[812,653,]
[870,]
[639,990,]
[740,]
[750,847,]
[916,]
```

```
[686,783,]
[24,441,]
[435,]
[137,]
[131,290,]
[505,]
[360,]
[195,354,]
[960,609,]
[752,849,]
[545,896,]
[555,]
[688,]
[698,]
[429,]
[151,]
[76,493,]
[221,380,]
[215,]
[111,462,]
[452,]
[244,]
[47,]
[388,37,]
[287,190,]
[277,]
[514,931,]
[937,]
[659,]
[824,]
[995,578,]
[729,]
[651,810,]
[800,]
[602,]
[592,]
[874,715,]
[864,]
[470,119,]
[327,]
[236,]
[406,]
[61,412,]
[166,]
[20,437,]
[143,]
[511,]
[197,356,]
[366,207,]
[984,633,]
[835,738,]
[569,]
[563,914,]
[777,680,]
[771,674,]
```

```
[461,]
[102,455,]
[348,253,]
[247,342,]
[44,397,]
[391,]
[523,938,]
[817,]
[666,]
[577,]
[730,]
[963,]
[761,856,]
[546,]
[691,786,]
[792,]
[430,15,]
[5,420,]
[148,]
[69,484,]
[222,]
[212,]
[23,]
[29,]
[301,]
[93,508,]
[198,]
[204,365,]
[634,987,]
[624,]
[842,]
[683,778,]
[673,768,]
[537,]
[946,531,]
[648,]
[803,]
[601,]
[595,]
[712,]
[867,706,]
[116,469,]
[229,324,]
[239,334,]
[415,]
[175,270,]
[525,]
[934,519,]
[668,]
[823,]
[589,]
[998,]
[893,732,]
[726,]
[96,449,]
```

```
[459,106,]
[336,]
[346,]
[32,385,]
[395,]
[177,]
[187,282,]
[418,3,]
[424,]
[307,146,]
[313,152,]
[67,]
[73,488,]
[210,]
[377,216,]
[975,622,]
[965,]
[767,862,]
[852,]
[558,911,]
[901,548,]
[798,]
[788,]
[983,]
[989,]
[743,]
[749,]
[566,]
[774,]
[780,685,]
[442,27,]
[17,432,]
[299,138,]
[128,289,]
[91,]
[81,]
[202,]
[353,]
[120,]
[114,467,]
[328,]
[227,]
[56,409,]
[403,50,]
[264,169,]
[163,258,]
[533,948,]
[958,543,]
[805,]
[815,654,]
[597,]
[869,708,]
[879,718,]
[291,]
[136,]
```

```
[19,434,]
[440,]
[355,]
[200,]
[89,504,]
[846,]
[741,836,]
[991,]
[782,687,]
[772,]
[917,]
[813,]
[646,]
[541,956,]
[535,]
[716,]
[710,]
[605,]
[225,]
[330,235,]
[465,112,]
[475,]
[256,161,]
[401,]
[249,344,]
[338,243,]
[457,]
[98,451,]
[280,]
[274,]
[821,660,]
[831,]
[932,517,]
[527,]
[885,724,]
[734,]
[581,]
[591,]
[759,]
[614,]
[973,620,]
[695,]
[796,]
[550,]
[909,]
[154,315,]
[305,144,]
[1,416,]
[218,]
[369,]
[490,75,]
[480,65,]
[739,]
[840,745,]
[626,]
```

```
[632,985,]
[770,675,]
[562,915,]
[921,568,]
[303,]
[293,132,]
[446,31,]
[436,21,]
[367,206,]
[196,]
[510,]
[500,]
[332,]
[231,]
[124,477,]
[118,471,]
[173,268,]
[413,60,]
[801,640,]
[529,944,]
[954,539,]
[704,865,]
[875,714,]
[603,]
[819,658,]
[936,]
[728,]
[579,994,]
[255,350,]
[100,]
[463,]
[276,181,]
[286,191,]
[36,389,]
[399,]
[150,311,]
[317,156,]
[7,422,]
[375,]
[220,]
[71,]
[77,492,]
[763,]
[753,]
[971,618,]
[961,608,]
[794,699,]
[689,]
[554,]
[544,897,]
[238,]
[325,228,]
[117,]
[174,271,]
[261,164,]
```

```
[63,414,]
[649,808,]
[530,]
[707,866,]
[713,872,]
[594,]
[843,]
[976,]
[769,672,]
[779,]
[561,912,]
[571,]
[141,300,]
[294,]
[28,]
[22,439,]
[358,199,]
[92,]
[503,]
[308,149,]
[318,]
[4,]
[14,431,]
[213,372,]
[68,]
[857,]
[851,754,]
[611,962,]
[787,]
[904,553,]
[898,]
[667,826,]
[816,]
[522,]
[929,]
[731,890,]
[586,]
[993,576,]
[343,]
[349,]
[103,454,]
[109,460,]
[182,]
[285,]
[390,39,]
[45,]
[655,]
[804,645,]
[959,542,]
[532,]
[878,]
[709,868,]
[606,]
[596,]
[323,]
```

```
[115,466,]
[472,]
[162,]
[168,]
[51,]
[408,]
[129,]
[298,139,]
[433,16,]
[443,]
[352,193,]
[497,]
[90,]
[845,748,]
[839,742,]
[631,982,]
[781,]
[567,]
[863,766,]
[964,613,]
[974,623,]
[692,]
[702,799,]
[559,910,]
[153,]
[306,147,]
[8,]
[419,2,]
[376,]
[370,211,]
[250,]
[240,337,]
[458,107,]
[448,97,]
1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
4
>> Number of rebuilding = 48

1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
5
>> Enter the element you want to search:

112
>> [112] is found

1) Build perfect HashTable
2) Build O(n^2) HashTable
```

```
3) Print table
4) Print number of rebuilding
5) Search

2
>> HashTable built successfully

1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
3

[215,109,186,467,578,559,446,318,632,687,339,533,388,706,892,128,966,92,334,139,409
,49,291,500,399,344,240,39,548,822,993,908,741,800,562,648,845,274,596,569,424,750,
193,22,123,874,44,65,976,150,511,622,724,325,794,973,713,542,354,437,271,472,177,10
2,220,244,35,153,330,413,603,566,863,904,442,365,471,256,105,190,4,211,962,529,710,
683,672,631,932,798,216,15,181,98,358,433,852,592,406,300,69,146,255,510,403,324,45
,250,64,151,849,597,568,176,103,221,10,355,436,270,712,543,626,972,886,929,938,893,
633,686,532,707,466,261,360,1,187,740,563,649,606,823,909,156,75,408,290,477,432,14
,99,799,673,716,539,68,147,41,301,646,824,898,862,819,653,602,567,294,497,412,152,9
63,788,191,5,210,443,470,257,982,769,517,451,276,430,377,16,199,546,591,806,843,141
,90,224,350,307,484,60,235,81,813,580,659,553,766,161,118,204,27,370,287,456,729,52
6,692,611,778,989,944,847,668,753,550,311,480,346,397,228,51,137,726,620,699,959,87
2,174,20,195,455,283,460,200,31,114,867,948,782,985,688,615,522,56,239,390,337,576,
663,915,836,581,658,767,812,61,234,313,779,870,728,527,420,286,457,119,17,198,124,4
50,277,431,376,702,516,954,983,768,392,306,91,225,842,590,689,614,732,523,866,783,9
84,164,115,282,461,375,416,808,914,763,577,391,336,490,317,131,238,229,50,136,481,5
86,752,551,846,921,803,427,380,454,120,175,21,772,958,621,698,212,3,464,786,965,936
,503,36,243,73,911,856,994,821,651,561,742,749,640,830,851,149,47,401,299,508,931,8
84,974,624,541,714,475,268,353,8,77,154,32,247,414,738,655,998,817,366,441,7,106,96
1,680,639,709,530,537,718,675,797,935,97,182,434,264,855,896,826,595,745,303,504,43
,327,400,298,148,67,249,831,901,748,571,222,100,474,439,352,540,715,885,975,792,787
,964,937,704,535,685,634,465,184,111,213,2,605,743,910,857,995,37,242,289,332,435,3
56,96,218,971,796,934,674,253,144,71,505,323,645,594,897,999,816,739,601,654,415,32
8,293,76,33,708,531,890,960,209,107,258,469,367,440,770,879,700,429,448,169,196,19,
667,545,758,840,805,227,89,304,349,342,385,315,492,63,835,916,554,24,207,117,162,45
9,422,369,608,695,525,730,868,990,777,801,754,671,345,308,93,138,231,623,725,514,87
5,956,774,23,173,275,452,418,463,280,113,166,28,781,864,734,691,236,59,129,488,338,
389,660,579,761,558,839,912,810,555,582,815,917,132,343,384,493,946,869,991,609,694
,524,731,458,285,206,116,163,168,197,449,519,618,771,878,305,395,348,143,804,589,66
6,544,759,613,987,780,865,112,29,202,419,372,462,281,]

1) Build perfect HashTable
2) Build O(n^2) HashTable
3) Print table
4) Print number of rebuilding
5) Search
```

Thanks