CS3310 Data and File Structures with Java
Instructor: Ajay K Gupta
Lab TA: Yu Guo, Zijiang J Yang
Assignment: 1

Jonah Kubath

Software Life Cycle Report – Assignment 1

Phase 1: Specification

- Use different techniques to search N x N array
- Practice developing solutions based on performance restraints
- Compare theoretical vs empirical outcomes
- Compare linear and binary search using arrays

Program Output is at the end of this file.

Phase 2 Design:

Modules and Basic structure

Module 1: Class Main

Fields:

1. int runAgain – user input if the program should run again
2. long startTime – when the program was started
3. long endTime – when the program ended
4. long totalTime – difference of endTime and startTime (total elapsed time)
5. Scanner scan – retrieves input from the user with System.in
6. inputArray – holds the data from the users
    a. length of array
    b. user's name
    c. number of characters to replace with random characters
    d. how many times to run the program with this data
7. boolean goodInput – used to test the input from the user

Methods:

1. main(String[] args)
    a. Input - command line arguments
    b. First instructions to run. Receives user input on whether or not to continue running the program.
2. program(String[] input)
    a. Input - Receives the user's input – name, array length, how many times to run the program, and how many times to swap a letter.
    b. Controls most of the program.

   c. Calls generateArray() to receive a randomly generated array
   d. Calls linearSearch() to perform the linear search
   e. Copies and sorts the array for Binary Search
   f. Calls binarySearchRow() and binarySearchCol()
   g. Replaces characters in the name with random characters
   h. Again calls linear search and binarySearchRow() and binarySearchCol()
3. getArrayDimension(Scanner scan)
   a. Input – Scanner object to receive keyboard information
   b. Handles the input from the user
      i. Length of the array
      ii. User's name
      iii. How many times to replace a character
      iv. How many times to run the program with given data
4. generateArray(int dimension)
   a. Input – The dimensions of the array given by the user.
   b. Takes in a dimension, generates a NxN array of random characters
5. searchArray(int[][] array, boolean[][] found, int letter)
   a. Input – random array, boolean array, the letter to search for in the array.
   b. Searches the given array for the letter
   c. Also checks again the boolean array to make sure that the position has not been used by a previous character
6. binarySearchRow(int[][] array, boolean[][] found, int letter, int beginning, int end)
   a. Input – sorted array, boolean array, the letter to search for, beginning segment to search in, the ending segment to search in.
   b. searches for the row that contains the letter
   c. this method is recursively called, so beginning and end are input
   d. Returns -1 if the letter is not found
7. binarySearchCol(int[][] array, boolean[][] found, int letter, int beginning, int end, int row)
   a. Input – sorted array, boolean array, the letter to search for in the array, beginning of the segment to search, the ending of the segment to search, and the row to search in.
   b. searches for the individual cell that contains the letter
   c. the row has already been found
   d. this method is recursively called, so beginning and end are inputs
   e. if the cell has already been used, another letter must be on either side of the cell because it has been sorted
   f. Returns -1 if the letter is not found
8. linearSearch(int[][] randomArray, boolean[][] found, String name)
   a. Input – random array, boolean array, the name to search
   b. linear search manager for the array
   c. calls searchArray() to perform the search
   d. [-1, -1] are printed if the letter is not found

Pseudocode

```
Main()
    Initialize variables – startTime, endTime, totalTime
    Create a scanner object to take user input from the keyboard
    Use a while loop to continue running the program
        Receive input from user for this instance of the program
            Size of array, Search Name, how many times to replace a character, how
            many times to run the program with the given data
        Run the program in a for loop however many times the user wants
            Take start time
            Call program()
            Take end time
            Find total time = end time – start time
            Print total time
        Ask the user if they want to run the program another time with new data
        If they do – continue in while loop
        Else – break and end program

Program()
    Create object for random characters (integers casted to characters)
    Initialize a boolean used to decide if the arrays should be printed (array length <= 10)
    Initialize a random character array by calling generateArray()
    Create another array to hold the sorted characters
    Create another to hold Booleans if a character has been used by a previous character
    while searching
    Create a 1d array to run Arrays.sort() for binary search
    Call linearSearch() on the random array
    Copy the random array to the 1d array
    Sort the 1d array
    Copy to the 2d array – sortedArray
    If the array length <= 10
        Print the sorted array
    Run binary search on the sorted array
    Run a for loop a number of times indicated by the user to replace a letter
        Randomly choose the letter to replace
        Generate a random character to replace that letter
    Reset the boolean array to false so letters can be searched again
    Call linearSearch() using the random name
    Reset the boolean array to false so letters can be searched again
    Run binary search on the sorted array with the random name

getArrayDimension()
    initialize a string array to hold the input from the user
```

```
        Use a boolean to exit a while loop when good input is received
        Use a while loop with a try catch to receive input and catch errors on bad input
        Store the users input in the array
        Once good input has been received – return the array

searchArray()
        initialize an array to hold the location of the found character
        Run a linear search on the array using 2 for loops
                If the cell == the letter and found == false
                        Save the location
                Else if found == true
                        Continue searching
        If the letter is not found set the location to [-1, -1]
        Return the array holding the location

binarySearchRow
        initialize a middle variable to hold the middle location (begin + end) / 2
        if the array is out of bounds return -1
        if end < beginning return -1
        if beginning char < letter and end char > letter
                return the row
        if beginning char > letter
                call binarySearchRow using the middle -1 row as the end and beginning as
                beginning
        if end char < letter
                call binarySearchRow using the middle + 1 row as the beginning and end row as
                the end row
        if none of these catch
                return -1
binarySearchCol()
        initialize a variable to hold the middle column location
        if the middle char == letter and found == false
                return the location
        if the middle char == letter and found == true
                search the letters next to this column
                if the char == letter and found == false
                        return the location
                else continue looking
        if beginning > end return -1
        if end < beginning return -1
        if the char at the col is > letter
                call binarySearchCol using middle + 1 as beginning and end as the end
        if the char at the col is < middle
                call binarySearchCol using middle – 1 as end and beginning as beginning
```

```
    if none of these catch the issue
        return -1

linearSearch()
    initialize an array to hold the location of the found letter
    hold the start time
    use a for loop to iterate through the characters in the given name
        call searchArray() to look for each letter
        output the location
    hold the end time
    find total time = end time – start time
    print the total time
```

Summary Analysis

Linear Search

Two searching techniques were used: linear searching and binary search. The linear searching was very efficient with small array dimensions and small search input. In a large matrix of values, it is very probably that a linear search will find the values somewhere in the first row that is searched which made it quick. It started to fall off in time, when there was a large input by the user with repeated values because it would have to search multiple rows to find a value that was not previously used.

Binary Search

The binary search because more effective as the array size grew along with a longer search string with multiple repeated values. Binary search was always fairly quick on searching, but that is not including the time that it took to sort the array. In production software, it would be a little different for each instance, but most of the time the array would only need to be sorted once. So on the first run, the sorting would add to the time cost, but each search after that would benefit using the log(n) time of binary search.

Linear Search vs Binary Search

Binary search in theory should over take linear search fairly quickly in time performance. Solving the equation of $n = n (\log(n))$. The question is when will log(n), base 2, be equal to 1. The issue with using the theoretical data in this case is that most names will fall into a general character pattern. There will be nearly no instances that cause the linear search to look through the entire array O(n). This probability greatly decreases as the matrix size is increased because most letters will be available in the early rows.

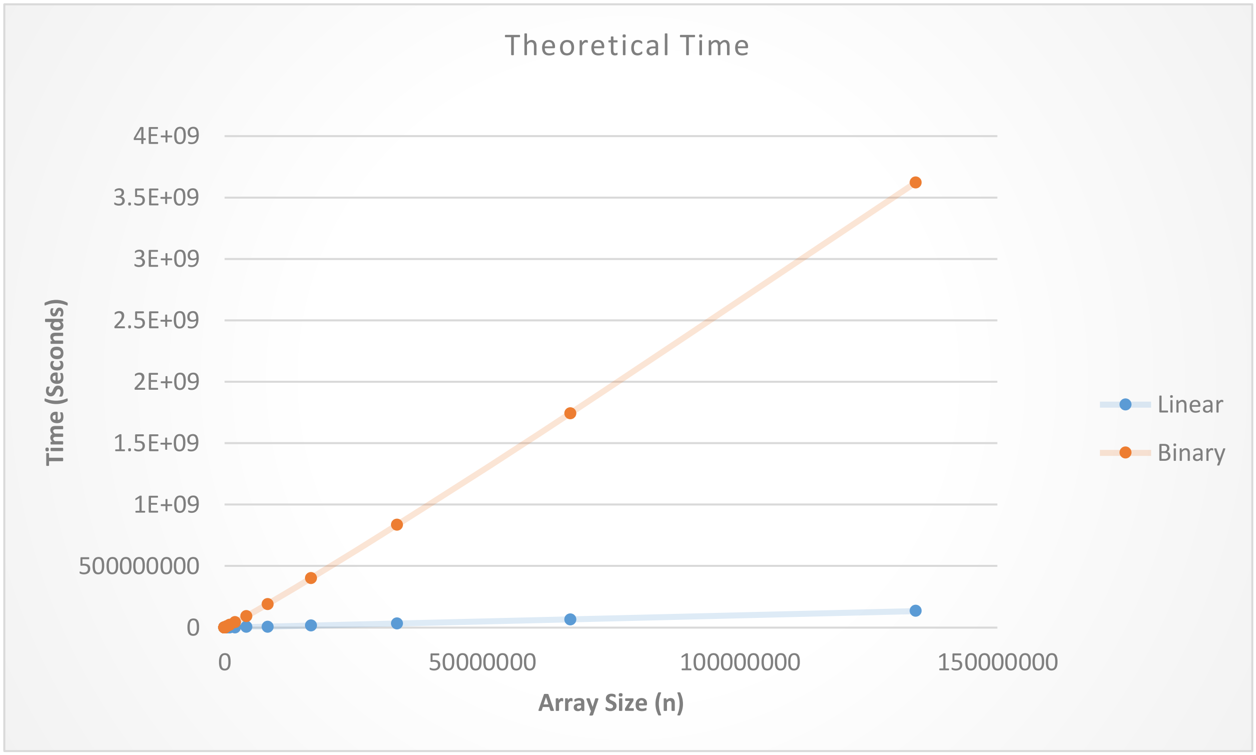
My code and search input did not have a section where the Binary Search really started to overtake the linear search. If the search strings were longer and had more repeated values, then the linear search would start to show its slowness.

Empirical Data vs. Theoretical

During my research and running of the program, the empirical data did not follow the theoretical data of the searches. The Linear Search should be O(n) and the Binary Search O(log n). I used the same string throughout the testing to keep the variables as consistent as possible. The issue is that unless I used letters that were duplicated or even characters that weren't generated, such as []{} or ?!@#, then the linear search would search the entire array. The only issue with doing that is that I would be tailoring my research to force a worst possible case.

What can I apply to production software?

Using this program as my base, I would learn that if my searching is going to be done by humans that would make it not completely random. My searches would, on average, fall into a fairly short name that is less than 50 characters and all letters. Using this knowledge, the linear search is a good choice because as the array of characters is larger, the likely hood of finding the letters in the first or second row increases. This would make searching quick.



Time Complexity (n = the total number of cells in the array)

main() – O(AB)

A = The number of times the user runs the program with new data

B = The number of time the user runs the program with repeated data

program() – O(n)

The linear search would be worst case O(n), but in my tests linear search never got near the end of the array.

getArrayDimension() – O(1)

generateArray() – O(n)

searchArray() – O(n)

binarySearchRow() – O(log (n))

binarySearchCol() – O(log (n)) or O(sqrt(n))

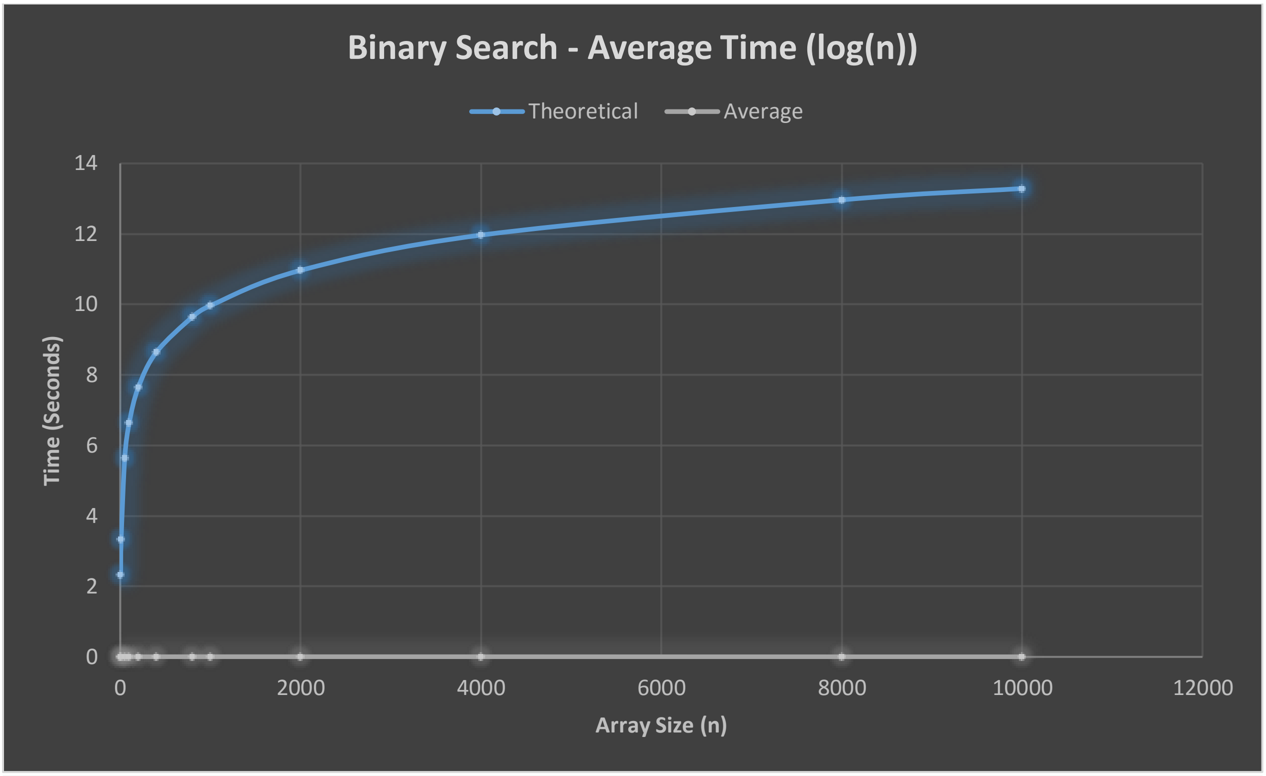
The reason I added the square root of n is because the binarySearchCol() will binary search until the letter is matched. At this point it looks for a matched letter that has not be previously used before and this is done linearly. Given the array is sorted, if I am a matched letter, the same letter could only be on either side of the current letter. The reason this could be square root n, the total number of cells, is that a row could be completely filled with the same letter with an end being the only cell that previously used. This would make the program step through each letter in the row (square root of n).

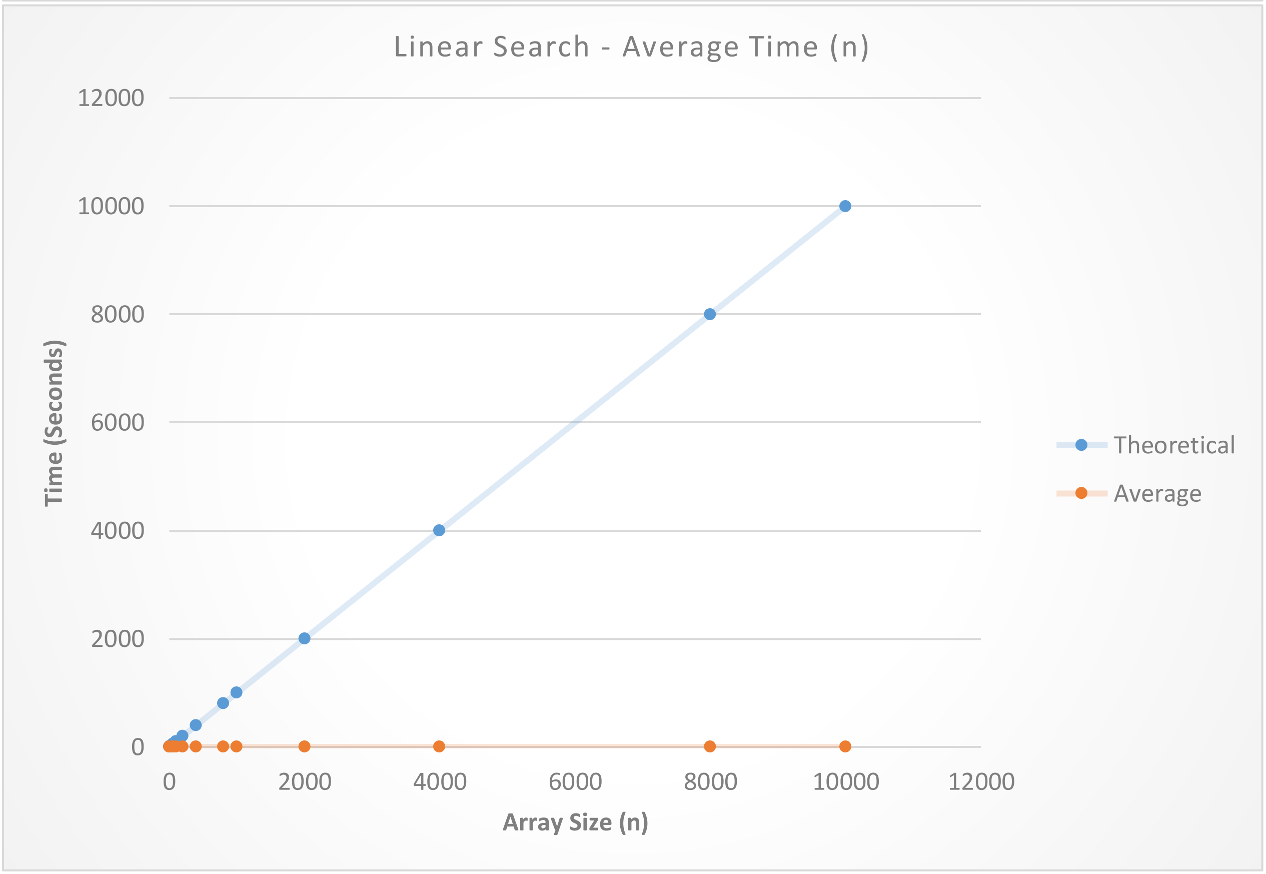
linearSearch() – O(C)

C = the length of the user's name

The time complexities were looking only at the instructions made with respect to the data in the arrays. This is why the main function is only O(AB) and not a max of all the methods. Even though the main function contains all the other methods, it helps view what the Big Oh notation is for individual methods if they are not overtaken by a child method.

This aspect can vary for each problem / solution. In this assignment, the goal was to look at comparing the time of linear and binary search.


Binary Search - Average Time (log(n))
Theoretical
Average
Time (Seconds)
Array Size (n)
0
2
4
6
8
10
12
14
0
2000
4000
6000
8000
10000
12000


Linear Search - Average Time (n)
Theoretical
Average
Time (Seconds)
Array Size (n)
0
2000
4000
6000
8000
10000
12000
0
2000
4000
6000
8000
10000
12000

Phase 3: Risk Analysis

There is no risk associated with this application. The only issue that could arise is if bad input was given by the user. I mitigated this by using a try / catch if they didn't input numbers for the array length, number of times to replace a character, or how many times to run the program. They were allowed to enter any character for their name, but only A-Z are generated.

Phase 4: Verification

I ran the program with a large amount of variation in the variable input. This includes the length of the array from 5 – 10,000 nearly doubling the value each run. Also, changing the number of times a character is chosen and replaced with: {1,2,4,8,16,32,64,128,256}. The arrays are output if the length and width is 10 or less.

Phase 5: Coding

The code of this program is included in the zip file. The code is explained with comments and line breaks to make reading easier. A Javadoc is also made to explain the use of each method.

Phase 6: Testing

Test runs have been included at the end of this file.

Phase 7: Refining the program

No refinements have been made for this program.

Phase 8: Production

A zip file including the source files from eclipse and the output of the program have been included.

Phase 9: Maintenance

Changes can be made once feedback is received.

Program Output

Name: jonahkubath　　　　Randomly changed characters (m): 5

N = 5;

```
What should the length of the array be?
5
What is your name?
jonahkubath
How many times should we replace a letter in your name?
5
How many times should the program run?
1

Program Run #1 -------------------------------------
Random Matrix
v z h j a
o x d w e
r e v w v
d h j i z
b y g j t

Linear Search ---------------------------------------
j : [0,3]
o : [1,0]
n : [-1,-1]
a : [0,4]
h : [0,2]
k : [-1,-1]
u : [-1,-1]
b : [4,0]
a : [-1,-1]
t : [4,4]
h : [3,1]
Time - 0.0020 seconds

Sorted
a b d d e
e g h h i
j j j o r
t v v v w
w x y z z

Binary Search ---------------------------------------
j - [2, 2]
o - [2, 3]
n - [-1, -1]
```

```
a - [0, 0]
h - [1, 2]
k - [-1, -1]
u - [-1, -1]
b - [0, 1]
a - [-1, -1]
t - [3, 0]
h - [1, 3]
Time - 0.0030 seconds

Linear Search - Random Name--------------------------
j : [0,3]
o : [1,0]
v : [0,0]
a : [0,4]
h : [0,2]
k : [-1,-1]
u : [-1,-1]
b : [4,0]
a : [-1,-1]
t : [4,4]
h : [3,1]
Time - 0.0020 seconds

Binary Search - Random Name -------------------------
j - [2, 2]
o - [2, 3]
v - [3, 2]
a - [0, 0]
h - [1, 2]
k - [-1, -1]
u - [-1, -1]
b - [0, 1]
a - [-1, -1]
t - [3, 0]
h - [1, 3]
Time - 0.0010 seconds
Total Run Time - 0.0160 seconds
```

N=10;

```
Program Run #1 --------------------------------------
Random Matrix
z j y e g h v z t z
b r f z n u u f z y
w d h d n p r n n k
p p b i c e y r s x
j b t v h z n c z h
b o z o u g w l x y
s l s n t w g i h m
b k l f w r m u p p
l t h a g i h y b a
w o p g n w c d a z

Linear Search ----------------------------------------
j : [0,1]
o : [5,1]
n : [1,4]
a : [8,3]
h : [0,5]
k : [2,9]
u : [1,5]
b : [1,0]
a : [8,9]
t : [0,8]
h : [2,2]
Time - 0.0020 seconds

Sorted
a a a b b b b b b c
c c d d d e e f f f
g g g g h h h h h
h h i i i j j k k l
l l l m m n n n n n
n n o o o p p p p p
p r r r r s s s t t
t t u u u u v v w w
w w w w x x y y y y
y z z z z z z z z z

Binary Search ----------------------------------------
j - [3, 5]
o - [5, 4]
n - [4, 7]
a - [0, 1]
```

```
h - [2, 7]
k - [3, 7]
u - [7, 4]
b - [0, 4]
a - [0, 2]
t - [7, 1]
h - [2, 8]
Time - 0.0010 seconds

Linear Search - Random Name---------------------------
j : [0,1]
o : [5,1]
n : [1,4]
a : [8,3]
u : [1,5]
k : [2,9]
u : [1,6]
b : [1,0]
a : [8,9]
t : [0,8]
h : [0,5]
Time - 0.0010 seconds

Binary Search - Random Name -------------------------
j - [3, 5]
o - [5, 4]
n - [4, 7]
a - [0, 1]
u - [7, 4]
k - [3, 7]
u - [7, 5]
b - [0, 4]
a - [0, 2]
t - [7, 1]
h - [2, 7]
Time - 0.0010 seconds
Total Run Time - 0.0200 seconds
```

N = 100;

```
Program Run #1 -------------------------------------
Linear Search ---------------------------------------
j : [0,43]
o : [0,1]
n : [0,6]
a : [0,18]
h : [1,36]
k : [0,4]
u : [0,99]
b : [0,66]
a : [0,32]
t : [0,11]
h : [1,45]
Time - 0.0030 seconds


Binary Search ---------------------------------------
j - [36, 49]
o - [55, 49]
n - [52, 49]
a - [2, 49]
h - [30, 24]
k - [42, 24]
u - [80, 49]
b - [5, 49]
a - [2, 50]
t - [74, 49]
h - [30, 25]
Time - 0.0030 seconds

Linear Search - Random Name--------------------------
t : [0,11]
o : [0,1]
n : [0,6]
a : [0,18]
h : [1,36]
k : [0,4]
u : [0,99]
b : [0,66]
a : [0,32]
t : [0,78]
h : [1,45]
Time - 0.0010 seconds
```

```
Binary Search - Random Name -------------------------
t - [74, 49]
o - [55, 49]
n - [52, 49]
a - [2, 49]
h - [30, 24]
k - [42, 24]
u - [80, 49]
b - [5, 49]
a - [2, 50]
t - [74, 50]
h - [30, 25]
Time - 0.0010 seconds
Total Run Time - 0.0190 seconds
```

N = 1000;

```
Program Run #1 -------------------------------------
Linear Search --------------------------------------
j : [0,48]
o : [0,0]
n : [0,3]
a : [0,5]
h : [0,23]
k : [0,13]
u : [0,21]
b : [0,80]
a : [0,61]
t : [0,7]
h : [0,56]
Time - 0.0020 seconds


Binary Search --------------------------------------
j - [374, 499]
o - [561, 499]
n - [530, 499]
a - [30, 499]
h - [280, 499]
k - [405, 499]
u - [780, 499]
b - [61, 499]
a - [30, 500]
t - [749, 499]
h - [280, 500]
Time - 0.0030 seconds

Linear Search - Random Name--------------------------
j : [0,48]
o : [0,0]
n : [0,3]
v : [0,27]
h : [0,23]
k : [0,13]
u : [0,21]
b : [0,80]
a : [0,5]
t : [0,7]
h : [0,56]
Time - 0.0020 seconds
```

```
Binary Search - Random Name --------------------------
j - [374, 499]
o - [561, 499]
n - [530, 499]
v - [811, 499]
h - [280, 499]
k - [405, 499]
u - [780, 499]
b - [61, 499]
a - [30, 499]
t - [749, 499]
h - [280, 500]
Time - 0.0040 seconds
Total Run Time - 0.1010 seconds
```