Nguyen, Abram - Lab 7 [Due: Sunday, November 11, 11:59 PM]
REPORT

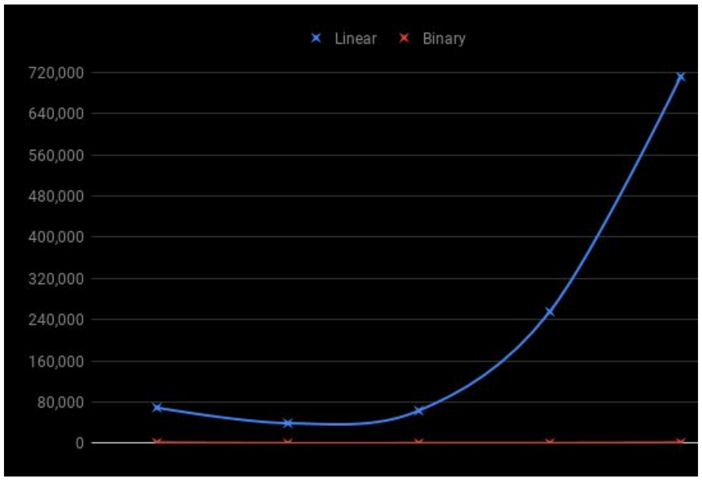
Running Environment:

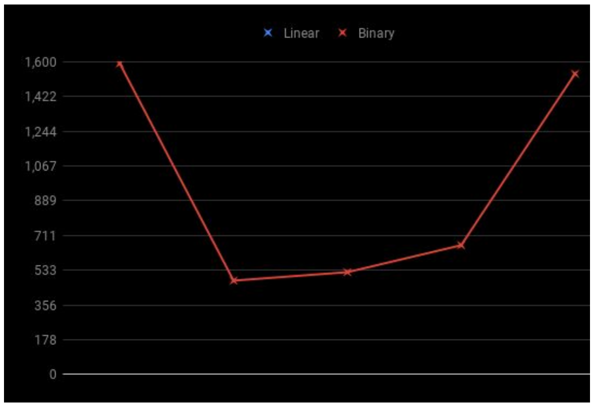
- Intel Core i5-4460 CPU @ 3.20GHz 3.20GHz
- 4.00 GB RAM
- Windows 10 Pro
- Java v1.8.0_181

Experiment:

1. Create 5 arrays of different sizes (10000, 40000, 160000, 640000, 1280000), sort them after creation
2. For **all 5 arrays**:
    a. Run through the array **30 times**, for each run through an array:
        i. Generate a random number within the array
        ii. Record the time (in nanoseconds) it takes to search for that random number in the array using a **linear** search algorithm
        iii. Add the linear search algorithm's time to a sum called '**timeLinear**'
        iv. Record the time (in nanoseconds) it takes to search for the same random number in the array using a **binary** search algorithm
        v. Add the binary search algorithm's time to a sum called '**timeBinary**'
3. Divide the sum 'timeLinear' by 30, which was the number of samples taken, then output that quotient.
4. Divide the sum 'timeBinary' by 30, which was the number of samples taken, then output that quotient.

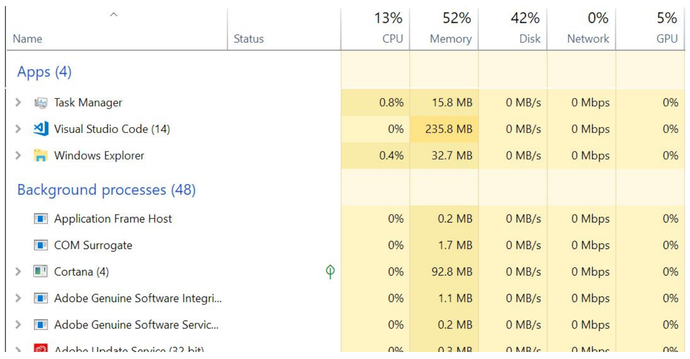
Results:





Possible Influences on Results:

The only program I had running at the time of my experiment was VS Code, which was the program I used to run the experiment. I used the nanoTime() method instead of currentTimeMillis(), which is not as accurate, but more precise. There is also the question of whether the 30 samples I took were good samples. For example, what if the majority of the numbers I searched for were relatively low numbers? Calculating the average, however, would eliminate most of the problems of accuracy. Other background apps on



Name	Status	13% CPU	52% Memory	42% Disk	0% Network	5% GPU
Apps (4)						
Task Manager		0.8%	15.8 MB	0 MB/s	0 Mbps	0%
Visual Studio Code (14)		0%	235.8 MB	0 MB/s	0 Mbps	0%
Windows Explorer		0.4%	32.7 MB	0 MB/s	0 Mbps	0%
Background processes (48)						
Application Frame Host		0%	0.2 MB	0 MB/s	0 Mbps	0%
COM Surrogate		0%	1.7 MB	0 MB/s	0 Mbps	0%
Cortana (4)		0%	92.8 MB	0 MB/s	0 Mbps	0%
Adobe Genuine Software Integri...		0%	1.1 MB	0 MB/s	0 Mbps	0%
Adobe Genuine Software Servic...		0%	0.2 MB	0 MB/s	0 Mbps	0%
Adobe Update Service (32 bit)		0%	0.3 MB	0 MB/s	0 Mbps	0%

my computer would affect the speed of my computer. My computer's disk speed and size, my RAM, and processor speed would affect my results. I've included information about some of the processes I had open from my task manage, including the background processes, while I ran my tests.

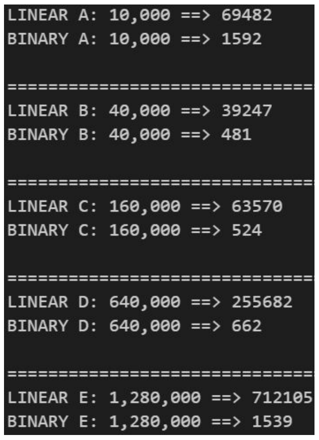
Big-O Notations

1. Linear Search: **O(n)**
    a. Theoretically, the actual run time versus Big-O run time should be a 1-to-1 ratio.
    b. Experimentally, this was only true for test B. An array of size 40,000 took about 40,000 nanoseconds to execute. Test A took longer than expected, but tests B, C, D and E seemed consistent in relation to each other. The times weren't accurate, but increased accordingly. For example, the size of array D was **4 times larger** than array C. The time it took to search through D was **4 times slower** than array C. The same goes for arrays D and E, by a factor of about 2.

2. Binary Search: **O(log n)**
    a. Theoretically, the actual run time verse Big-O run time should be a logarithmic ratio.
    b. Experimentally, this was only true for tests B, C and D. Tests A and E took longer than expected, A took longer than E, despite being a much smaller array. I've included a table to compare the theoretical and experimental numbers.

Tests	**Theoretical Run time**	**Experimental Run Time**
A, log(10,000)	4	1,592
B, log(40,000)	4.6	481
C, log(160,000)	5.2	524
D, log(640,000)	5.8	662
E, log(1,280,000)	6.1	1,539

Experiment output for all 5 tests, both linear and binary times in nanoseconds:



```
LINEAR A: 10,000 ==> 69482
BINARY A: 10,000 ==> 1592

=============================
LINEAR B: 40,000 ==> 39247
BINARY B: 40,000 ==> 481

=============================
LINEAR C: 160,000 ==> 63570
BINARY C: 160,000 ==> 524

=============================
LINEAR D: 640,000 ==> 255682
BINARY D: 640,000 ==> 662

=============================
LINEAR E: 1,280,000 ==> 712105
BINARY E: 1,280,000 ==> 1539
```