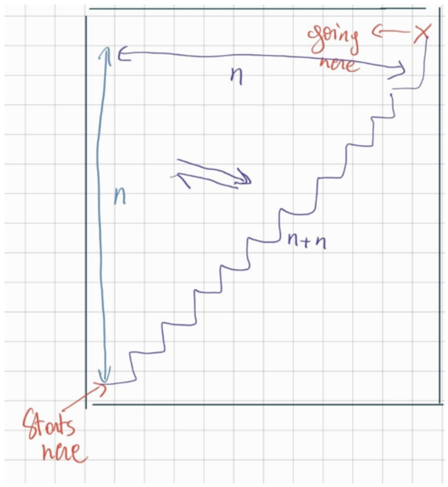
Oxl51

Lab Report CSDS 233 HW2:

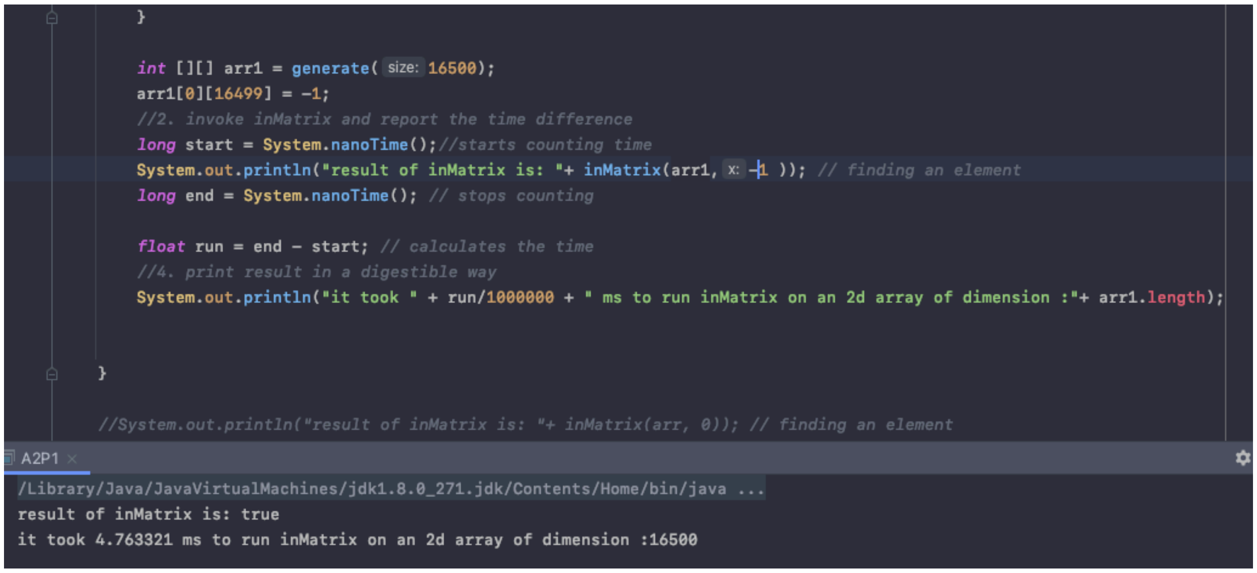
What kind of growth is displayed by the runtime? Provide both an analytical answer (by analyzing your code), and an empirical answer (by analyzing the time measurements).
Is this program efficient or inefficient (e.g., how does this relate to the number of elements in the matrix?)

Analytical Answer:



My in Matrix method grows linearly because the worst case scenario is that it traverse (n-1) squares plus n squares, which makes it a O(2n) which is equivalent to O(n). The worst case scenario happens when we are looking for the upper right square of the 2d array. As shown on this image, the worst case goes through 2n squares.

Empirical Answer:
I tested my code using an array generator (included at the bottom of my class). And recording the time taken using System.nanotime.
I generated arrays as big as my heap allowed(until around 25000 in dimension) further than that I would get an exception out of memory.
To test my method, I generated an array with positive numbers and replaced the worst case scenario(upper right) with (-1) and ran the search to find -1. That way, I was sure my result would the worst case scenario and the growth would be my big O value.
Here is the code I used:



```
    }

    int [][] arr1 = generate( size: 16500);
    arr1[0][16499] = -1;
    //2. invoke inMatrix and report the time difference
    long start = System.nanoTime();//starts counting time
    System.out.println("result of inMatrix is: "+ inMatrix(arr1, x: -1 )); // finding an element
    long end = System.nanoTime(); // stops counting

    float run = end - start; // calculates the time
    //4. print result in a digestible way
    System.out.println("it took " + run/1000000 + " ms to run inMatrix on an 2d array of dimension :"+ arr1.length);


}

//System.out.println("result of inMatrix is: "+ inMatrix(arr, 0)); // finding an element
```

A2P1

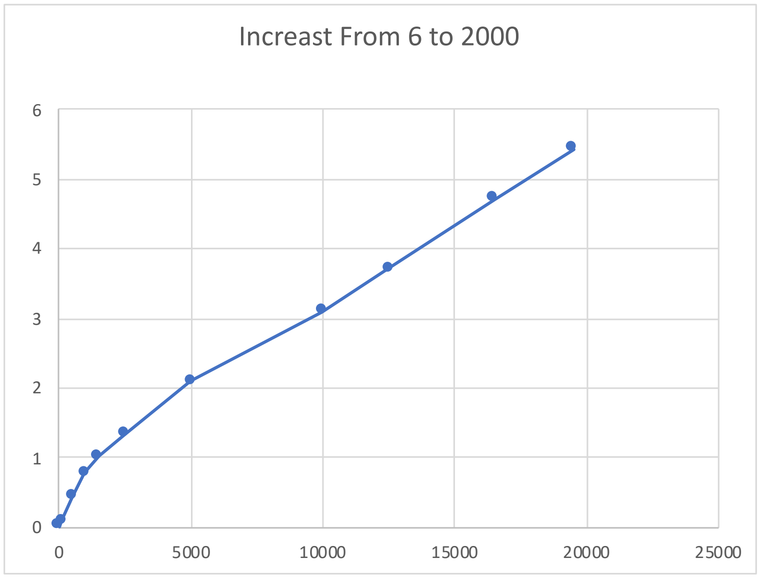
```
/Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java ...
result of inMatrix is: true
it took 4.763321 ms to run inMatrix on an 2d array of dimension :16500
```

Because the output would usually vary greatly with each time I run the method, I averaged 10 tries of each dimension and I plotted that

Oxl51

Here are the average results of the dimensions I tried my method on

10	0.018189
100	0.092919
500	0.438681
1000	0.777747
1500	1.026949
2500	1.339747
5000	2.09181
10000	3.10192
12500	3.719619
16500	4.723431
19500	5.436113



Conclusion
We can notice that the curve becomes linear as the dimension of the array grows bigger and bigger. Which proves my expectations of O(n).

This method is efficient, the time it takes to visit an array grows linearly and therefore there is no huge increase in time consumption opposing what would happen with a O(n^2) method. My method doesn't travers as many useless positions in the array and moves methodically.