2. Without coding, how would you change ThreeSumFast to become FourSumFast?

We would add a fourth integer l and a third for loop for k and call binarySearch() with the new integer l.

3.

a. 5 different classes of time complexities and an example of each
    i. Constant time (O(1)): print statement, calculating addition, subtraction, or other simple math
    ii. Logarithmic time (O(log n)): binary search
    iii. Linear time (O(n)): finding the largest/smallest element of an array
    iv. Cubic time (O(n^3)): array multiplication
    v. Quadratic time (O(n^2)): bubble sort

b. Code snippet showing a loop of constant time complexity

```
for (int i = 1; i <= a; i++) {
        System.out.println("Hello World!");
}
```

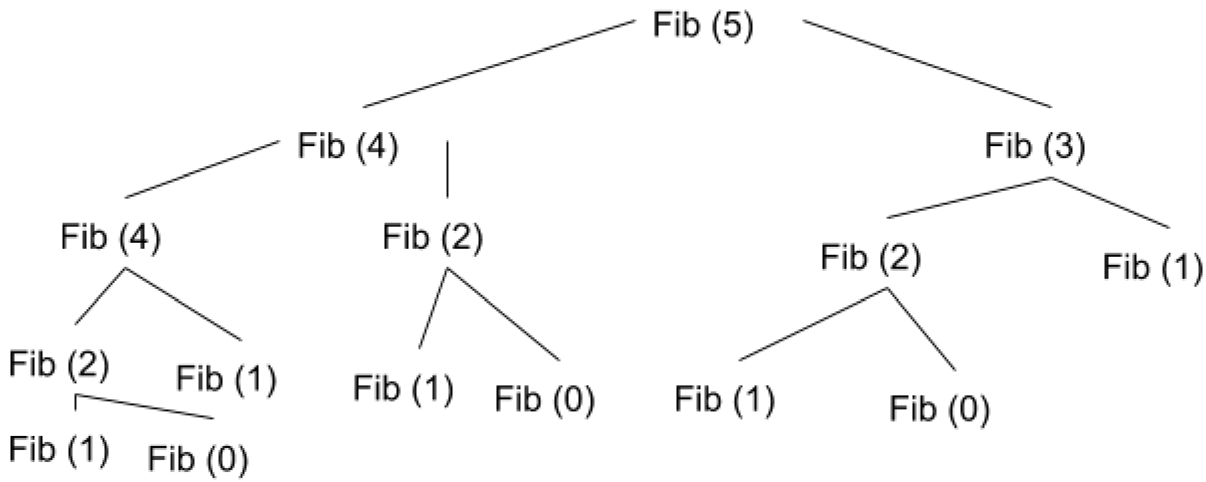
c. If an algorithm executes O(log n) time computation for each entry of an array storing n elements, what is the big O for storing the entire array?

    n*(logn)

d. Using an n of 5, show that time complexity of the recursive fibonacci algorithm is O(2^n) - write it out

    The time complexity is exponential because each time the function is called, you are calling the function two more times. When the function is called n times, you get a time complexity of O(2^n).

    For an n of 5, the tree has 5 levels:



e. Using the running time and big o from your programming assignment, predict what the running time of 8000 items would be

   Since the time complexity of ThreeSum was o(n^3), the time complexity of our new program would be o(n^4). Our program ran in 87 seconds with 1000 integers. We predicted that it would take 5939.2 minutes. Since, 1000 goes into 8000 8 times we multiplied our runtime for 1000 integers by 8^4.

f. Give me the big O of the algorithms with the following worst case runtimes (T(N)):
   i. T(N) = 3N^2 + 10N + 17
      O(n^2)
   ii. T(N) = N + 9999
      O(n)
   iii. T(N) = 734N
      O(n)