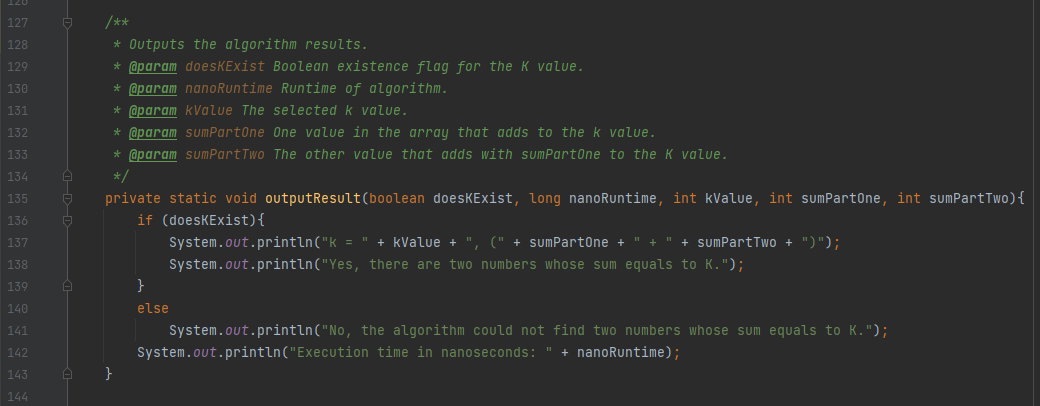
Travis Munyer

Programming Assignment 1

7/20/21

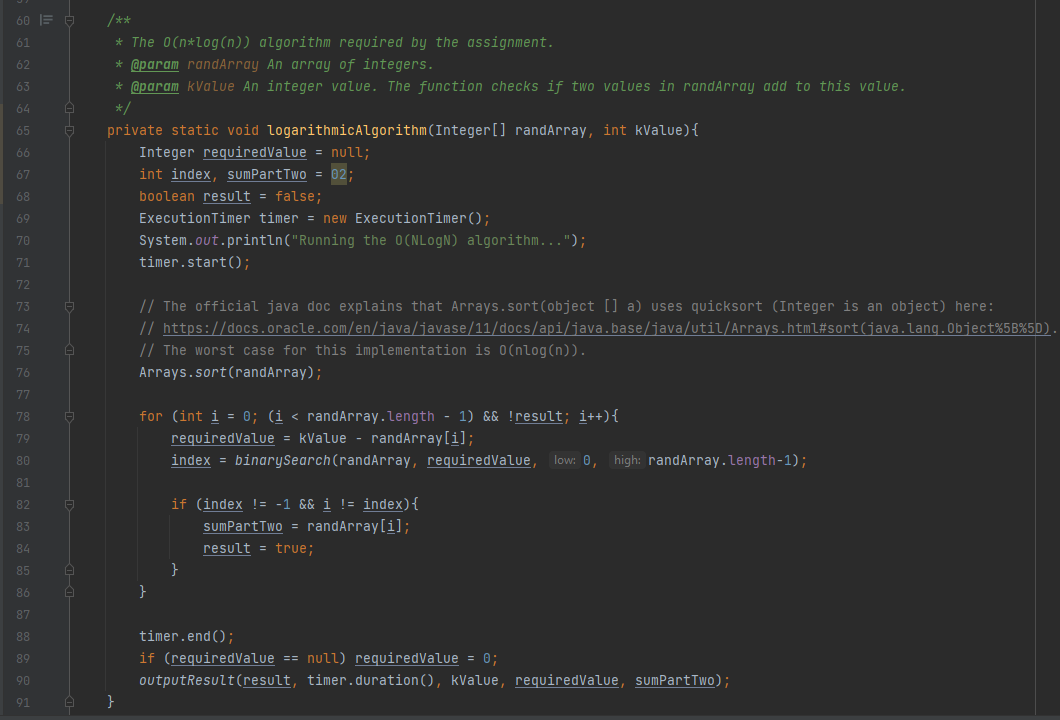


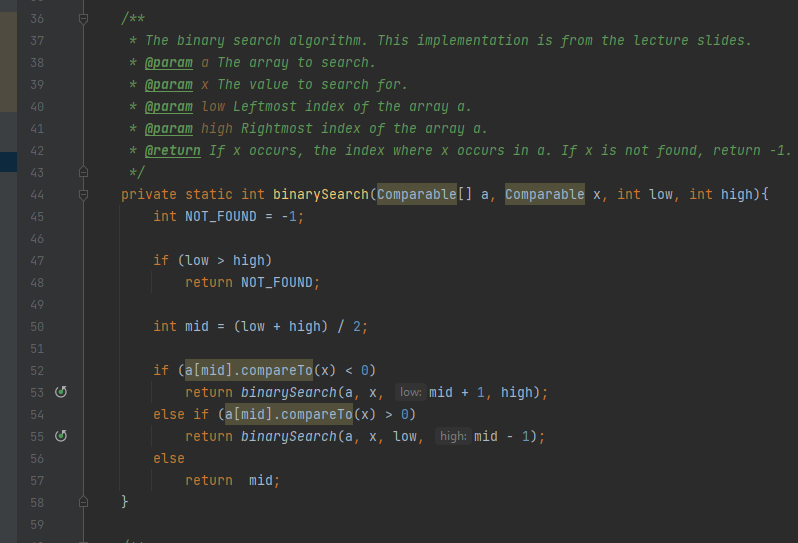
O(N^2) algorithm



This algorithm simply checks every possible combination of values in the array. Lines 99-105 are constant (c1) operations. The if statement on line 107 almost always occurs. The for loops on lines 108 and 109 are both N complexity (where N is the size of the array), so when multiplied together this is N^2 complexity. The work inside the for loops (lines 110-116) is constant (c2). Also, the operations on lines 122-123 is also constant (c3). (See the image above this algorithm for the outputResult() method). The outputResult() method only displays the result and is constant number of operations. This in total means we are doing about N^2 + c1 + c2 + c3) operations. Simplified, this is O(N^2) complexity.

O(nlog(n))





First, this algorithm sorts the values using mergesort. As we discussed, since I could find official documentation of the algorithm used by java, I use java’s sort algorithm (line 76). The documentation is here: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html#sort(java.lang.Object%5B%5D)>.

As we discussed in class, mergesort is always O(nlogn), but as added verification, the java documentation confirms the complexity of its algorithm. Once sorted, this algorithm simply loops all values in the sorted array (n complexity, line 78), and then computes the required value to make the sum exist (line 79). This way we need only to find a single value to check if the sum exists. To find this value, I use binary search.

Complexity of binary search:

(In binary search image) Line 45-50 is constant (c1). Then, we either have T(n/2) operations (lines 53 and 55), or an additional constant in the average/best case (line 57), but we will consider worst case here (T(n/2)). This means the recurrence relation is as follows: T(n) = T(n/2) + c1. By the masters theorem, this is O(logn) complexity.

Total complexity:

Lines 66-71 is constant complexity (c1). As discussed above, line 76 is O(nlog(n)) complexity. Line 79 (c2) and lines 82-85 (c3) is constant complexity. The for loop on line 78 is n complexity. As shown in the complexity of binary search discussion, line 78 is O(logn) complexity. So, this loop (line 78) is O(nlogn) complexity. Also, line 88-90 is constant (c4) complexity. Therefore, the complexity is about (without some constants) 2\*nlog(n) + c1 + c2+ c3 + c4. Thus, this algorithm is O(nlog(n)) complexity.

Charts/Graphs

|  |  |  |
| --- | --- | --- |
| Input Size (8000) | O(N^2) algorithm | O(NlogN) algorithm |
| Test 1 | 309179300 | 10678800 |
| Test 2 | 314967100 | 5147299 |
| Test 3 | 215000799 | 4388800 |
| Test 4 | 214808100 | 6846101 |
| Test 5 | 214854900 | 9430900 |
| Test 6 | 222766999 | 11602501 |
| Test 7 | 216345000 | 7045399 |
| Test 8 | 214154599 | 6947901 |
| Test 9 | 215336001 | 4094300 |
| Test 10 | 215047601 | 4243300 |
| Average Worst-case time | 235246039 | 7042530 |

|  |  |  |
| --- | --- | --- |
| Input Size (16000) | O(N^2) algorithm | O(NlogN) algorithm |
| Test 1 | 938549000 | 10950400 |
| Test 2 | 897731500 | 5107800 |
| Test 3 | 1578138800 | 6029000 |
| Test 4 | 1544681300 | 6243200 |
| Test 5 | 1456691600 | 5920200 |
| Test 6 | 1490980900 | 4514700 |
| Test 7 | 1493223300 | 6609500 |
| Test 8 | 1507221700 | 4608800 |
| Test 9 | 1464734300 | 4471700 |
| Test 10 | 1463084300 | 4548300 |
| Average Worst-case time | 1383503670 | 5900360 |

|  |  |  |
| --- | --- | --- |
| Input Size (32000) | O(N^2) algorithm | O(NlogN) algorithm |
| Test 1 | 6082417500 | 9557000 |
| Test 2 | 5867940200 | 9492000 |
| Test 3 | 5981807700 | 12644300 |
| Test 4 | 5818755100 | 9610800 |
| Test 5 | 5930643300 | 9477900 |
| Test 6 | 5966068600 | 9778800 |
| Test 7 | 5964405000 | 9509500 |
| Test 8 | 6213388500 | 9487700 |
| Test 9 | 5900106800 | 9809600 |
| Test 10 | 5855155100 | 9507900 |
| Average Worst-case time | 5958068780 | 9887550 |

|  |  |  |
| --- | --- | --- |
| Input Size (64000) | O(N^2) algorithm | O(NlogN) algorithm |
| Test 1 | 23804586200 | 22101600 |
| Test 2 | 23556114200 | 20996900 |
| Test 3 | 23461499800 | 21093900 |
| Test 4 | 24137122200 | 21104000 |
| Test 5 | 23484956200 | 21749300 |
| Test 6 | 24068634700 | 21707300 |
| Test 7 | 23864662500 | 20794500 |
| Test 8 | 30100093000 | 20867200 |
| Test 9 | 27063005600 | 21152800 |
| Test 10 | 32364692500 | 21278400 |
| Average Worst-case time | 25590536690 | 21284590 |

|  |  |  |
| --- | --- | --- |
| Input Size (128000) | O(N^2) algorithm | O(NlogN) algorithm |
| Test 1 | 147973215200 | 45595400 |
| Test 2 | 139296214800 | 46425700 |
| Test 3 | 130903262700 | 52811500 |
| Test 4 | 141836849800 | 53288100 |
| Test 5 | 151381763300 | 44078800 |
| Test 6 | 131919662800 | 48262700 |
| Test 7 | 138656958300 | 47751000 |
| Test 8 | 122305389100 | 47917000 |
| Test 9 | 122798704500 | 50620500 |
| Test 10 | 122305267000 | 60263700 |
| Average Worst-case time | 134937728750 | 49701440 |

|  |  |  |
| --- | --- | --- |
| Input Size (156000) | O(N^2) algorithm | O(NlogN) algorithm |
| Test 1 | 195350194500 | 65929600 |
| Test 2 | 198100523300 | 58827400 |
| Test 3 | 198586223900 | 62633300 |
| Test 4 | 188903582900 | 61206700 |
| Test 5 | 196002810000 | 61699500 |
| Test 6 | 189250580000 | 56861500 |
| Test 7 | 189137953300 | 63044800 |
| Test 8 | 192793004300 | 59763400 |
| Test 9 | 192462404300 | 60511600 |
| Test 10 | 208079814500 | 60531900 |
| Average Worst-case time | 194866709100 | 61100970 |