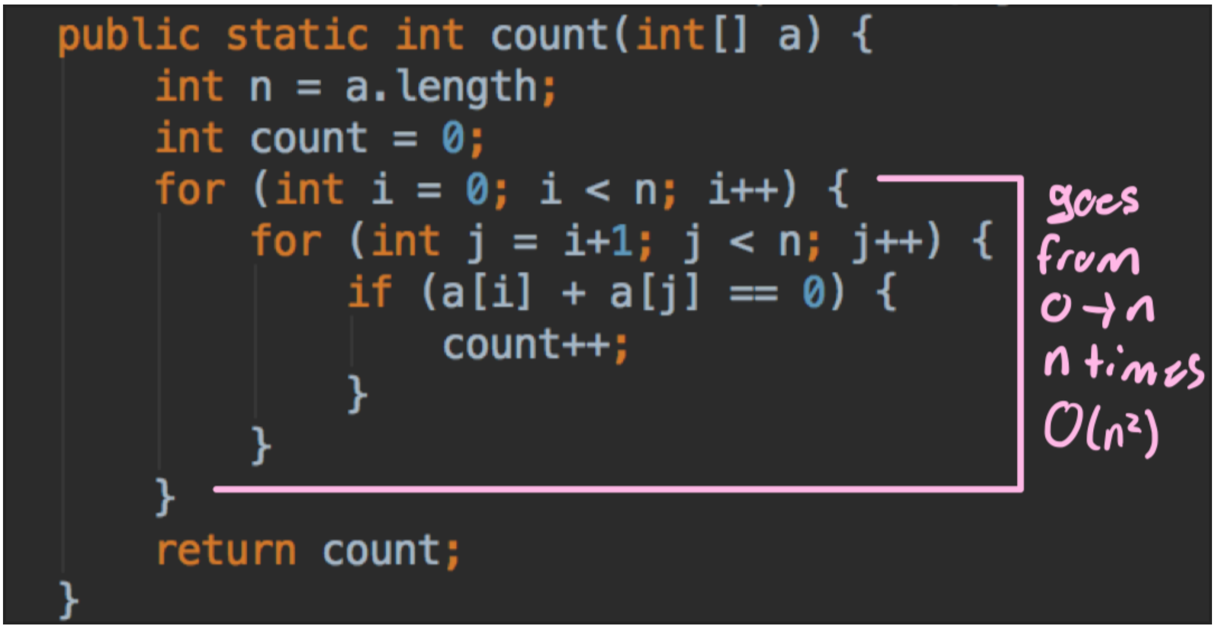
Computer Science Lab Report

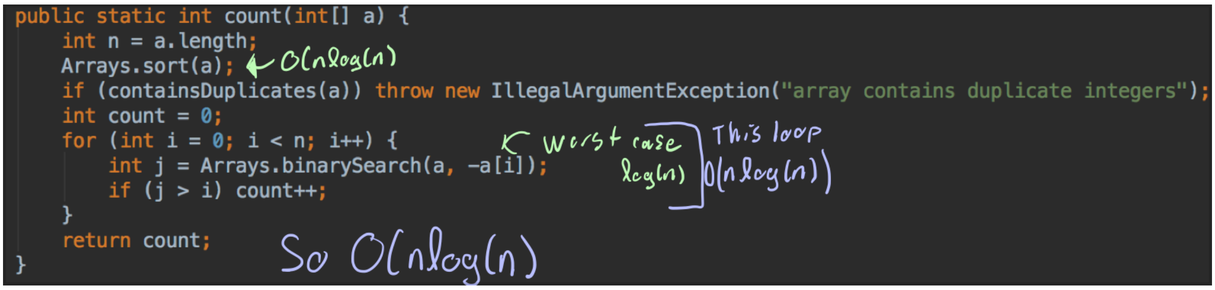
#csc

Task 1

1. 

```java
public static int count(int[] a) {
    int n = a.length;
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (a[i] + a[j] == 0) {
                count++;
            }
        }
    }
    return count;
}
```

goes from 0 → n n times $O(n^2)$

2. 

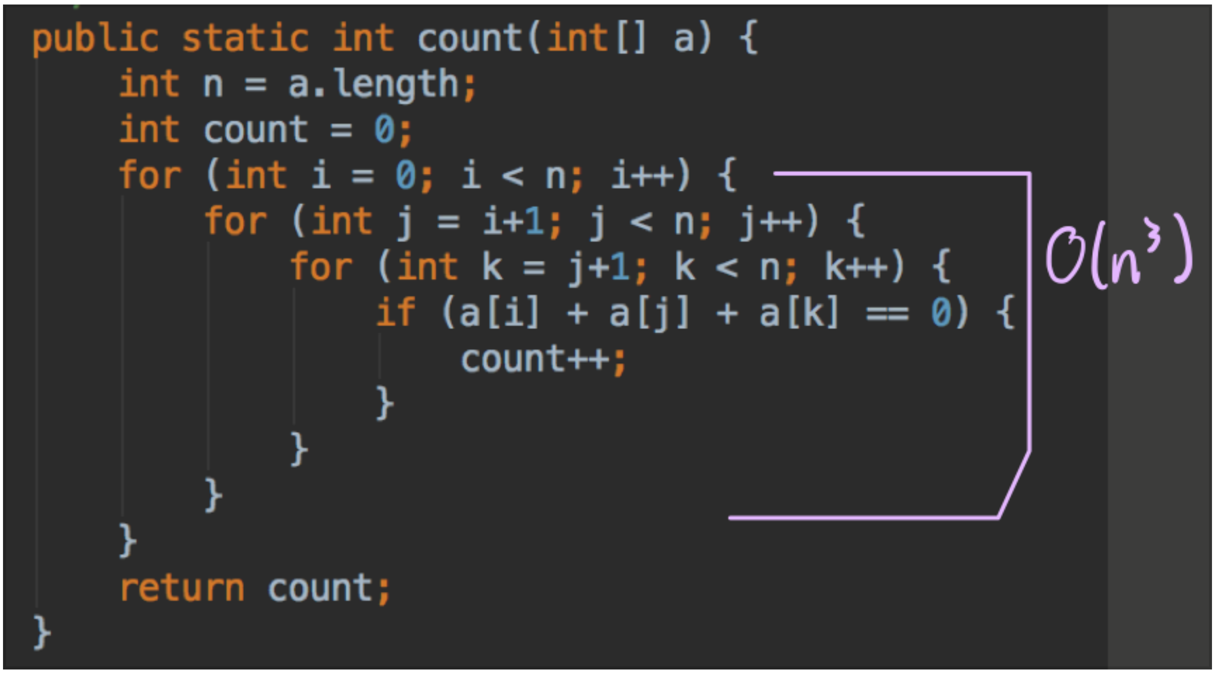
```java
public static int count(int[] a) {
    int n = a.length;
    Arrays.sort(a);
    if (containsDuplicates(a)) throw new IllegalArgumentException("array contains duplicate integers");
    int count = 0;
    for (int i = 0; i < n; i++) {
        int j = Arrays.binarySearch(a, -a[i]);
        if (j > i) count++;
    }
    return count;
}
```

← $O(n\log(n))$

← worst case $\log(n)$

This loop $O(n\log(n))$

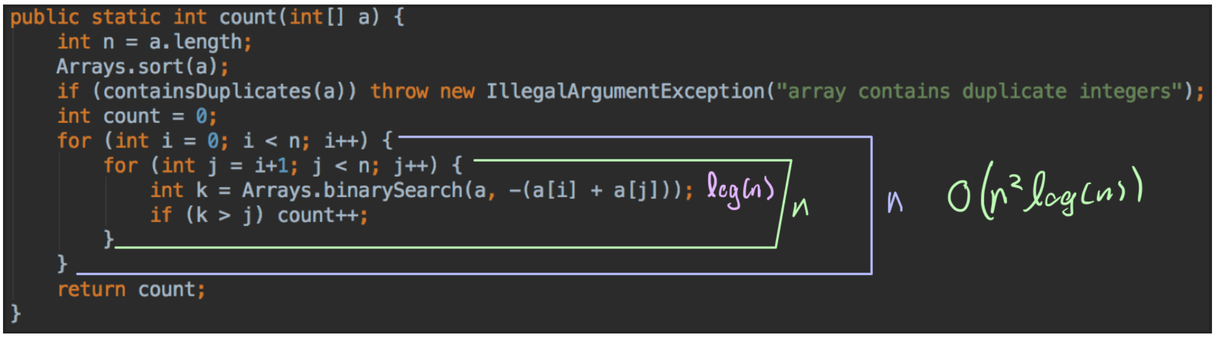
So $O(n\log(n))$

3. 

```java
public static int count(int[] a) {
    int n = a.length;
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            for (int k = j+1; k < n; k++) {
                if (a[i] + a[j] + a[k] == 0) {
                    count++;
                }
            }
        }
    }
    return count;
}
```

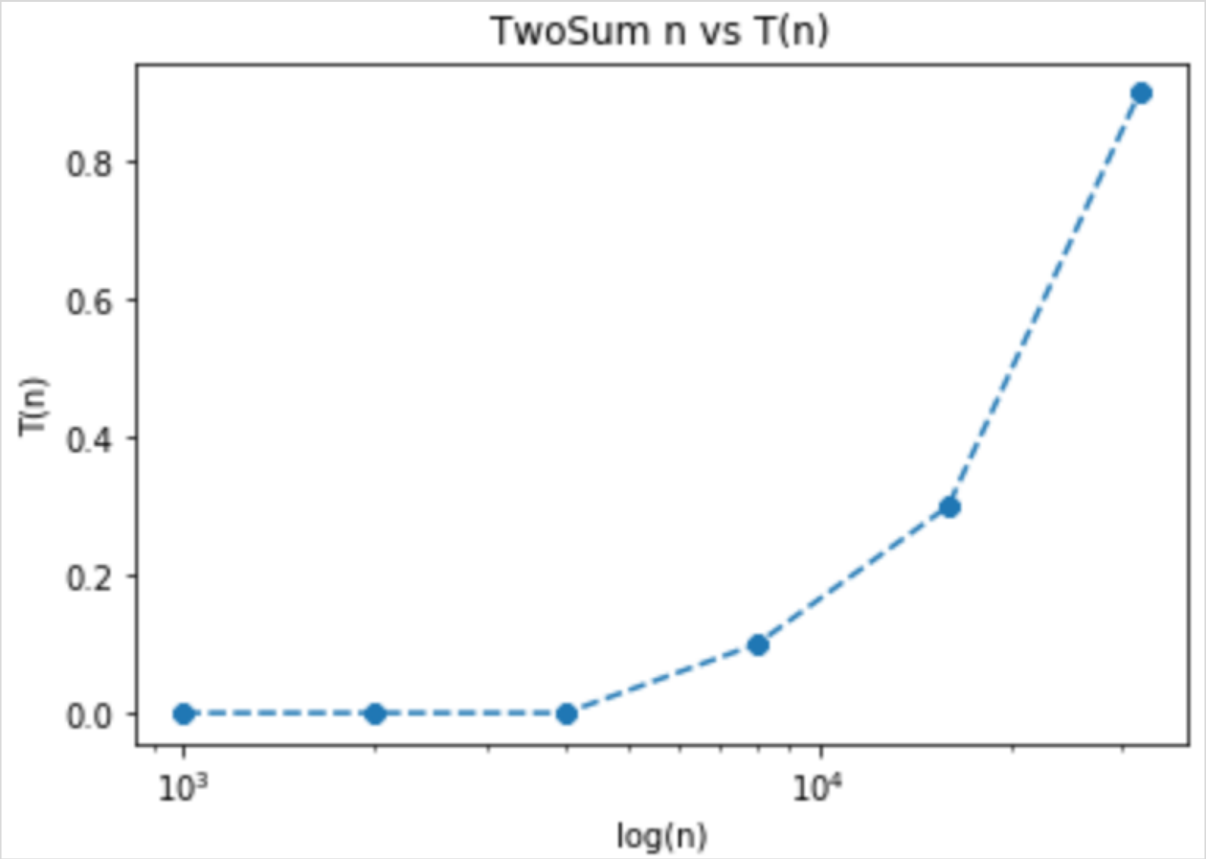
$O(n^3)$

4.

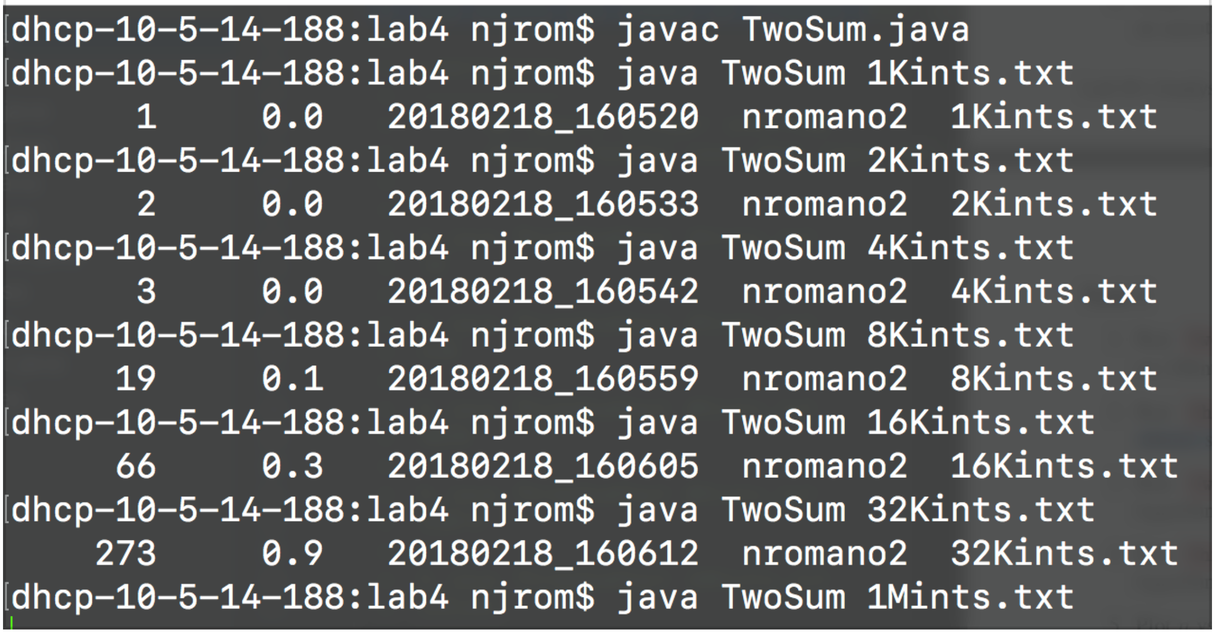


```
public static int count(int[] a) {
    int n = a.length;
    Arrays.sort(a);
    if (containsDuplicates(a)) throw new IllegalArgumentException("array contains duplicate integers");
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            int k = Arrays.binarySearch(a, -(a[i] + a[j])); 
            if (k > j) count++;
        }
    }
    return count;
}
```

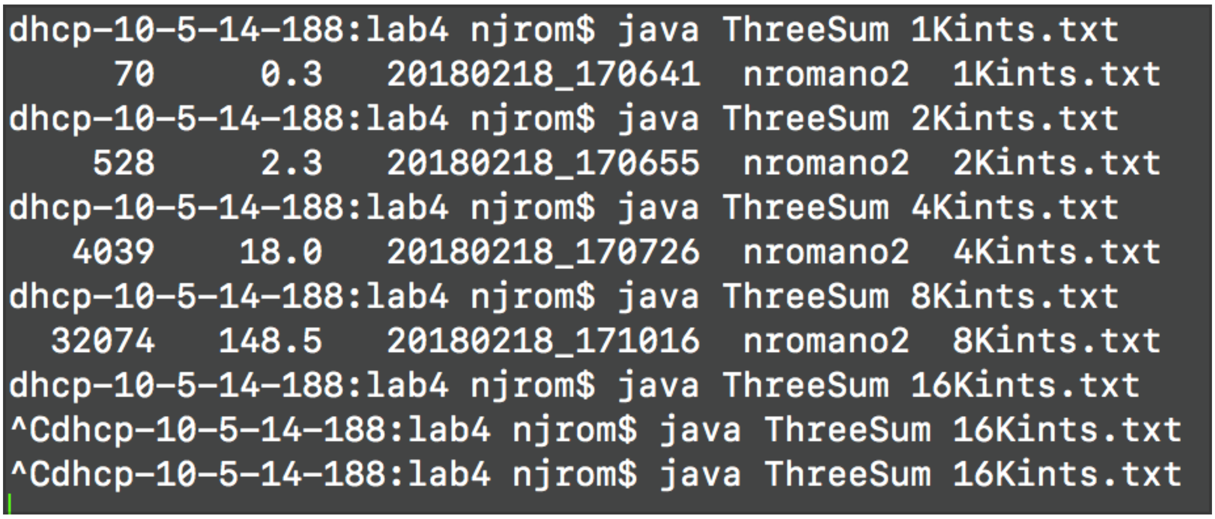
Task 2



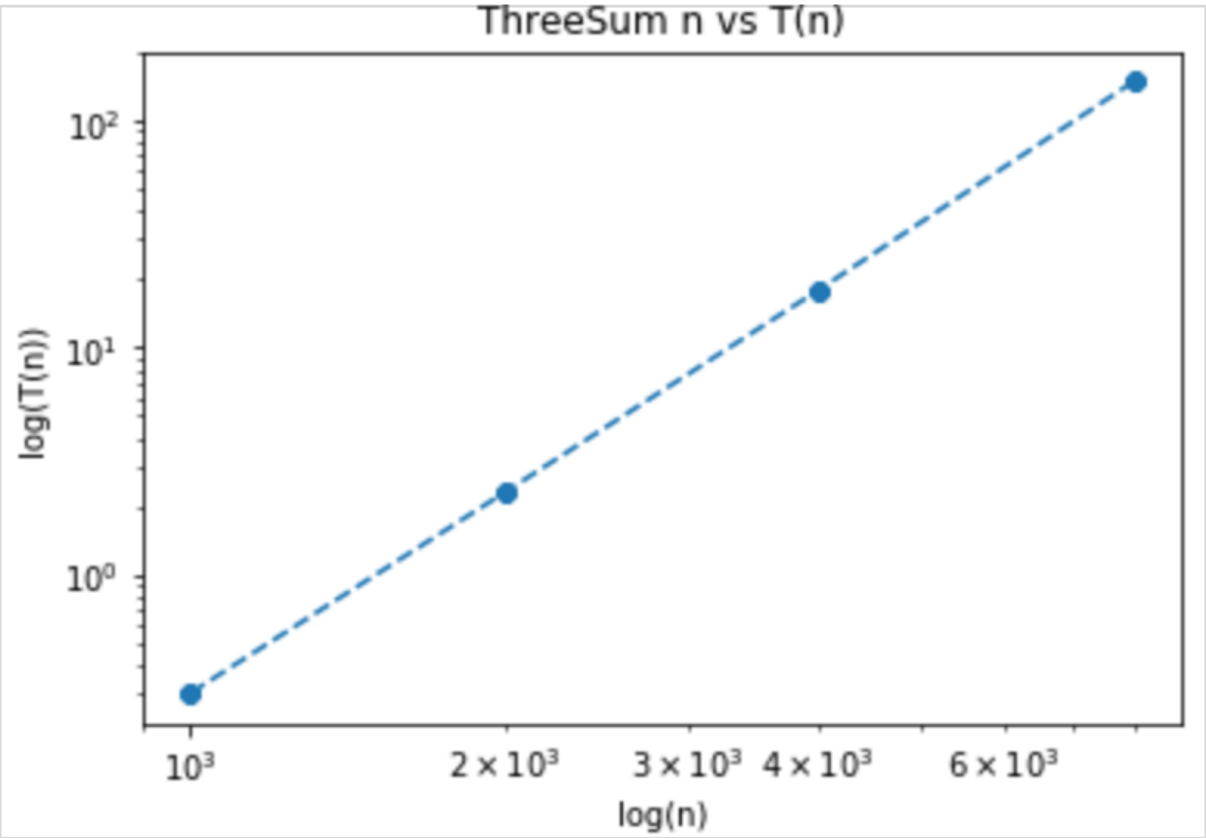
1.



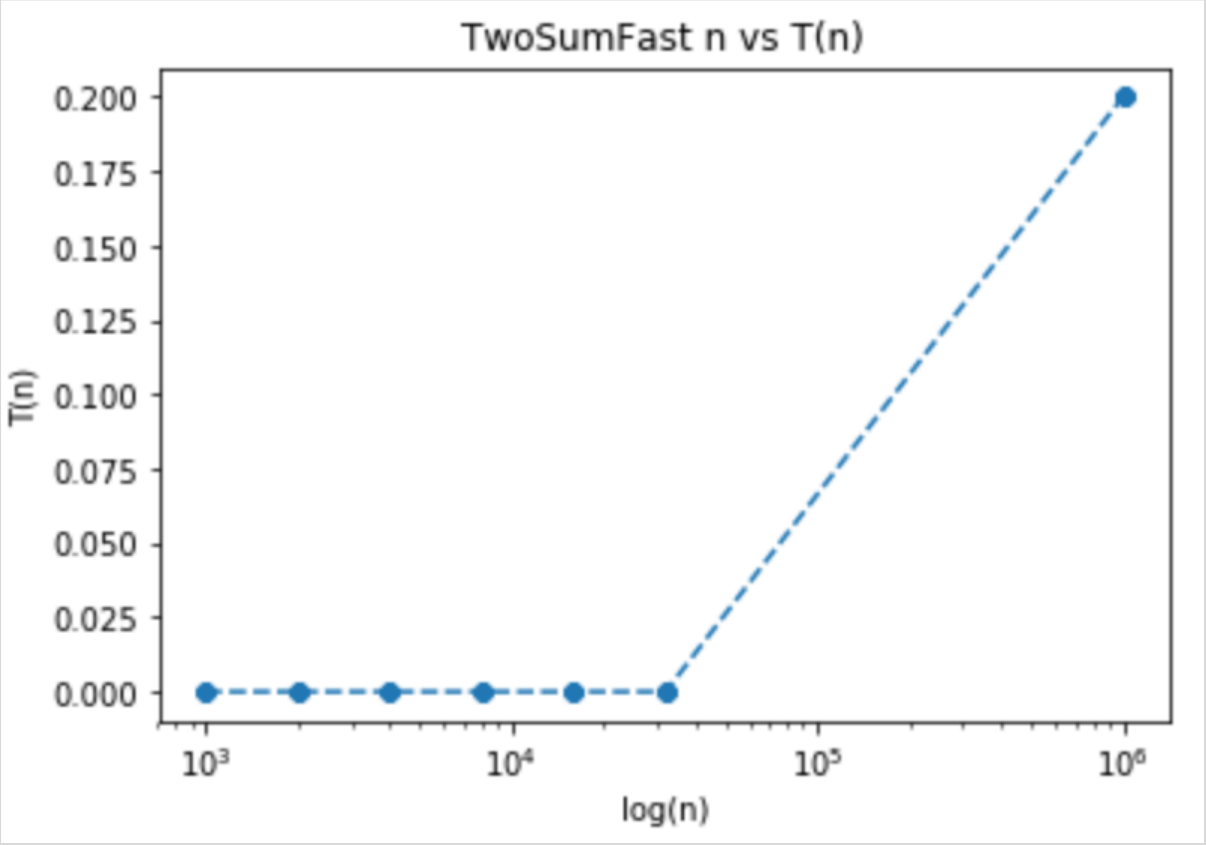
```
[dhcp-10-5-14-188:lab4 njrom$ javac TwoSum.java
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 1Kints.txt
       1      0.0    20180218_160520  nromano2  1Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 2Kints.txt
       2      0.0    20180218_160533  nromano2  2Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 4Kints.txt
       3      0.0    20180218_160542  nromano2  4Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 8Kints.txt
      19      0.1    20180218_160559  nromano2  8Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 16Kints.txt
      66      0.3    20180218_160605  nromano2  16Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 32Kints.txt
     273      0.9    20180218_160612  nromano2  32Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSum 1Mints.txt
```

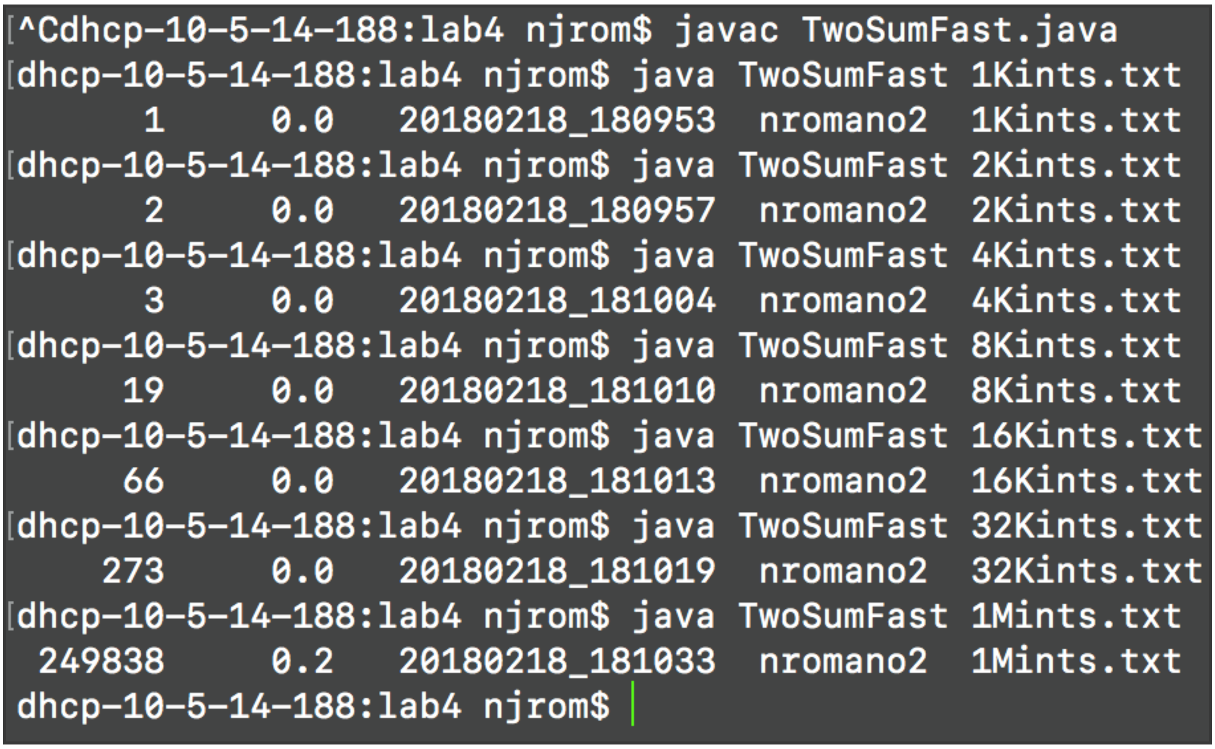


```
dhcp-10-5-14-188:lab4 njrom$ java ThreeSum 1Kints.txt
      70      0.3    20180218_170641  nromano2  1Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSum 2Kints.txt
     528      2.3    20180218_170655  nromano2  2Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSum 4Kints.txt
    4039     18.0    20180218_170726  nromano2  4Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSum 8Kints.txt
   32074    148.5    20180218_171016  nromano2  8Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSum 16Kints.txt
^Cdhcp-10-5-14-188:lab4 njrom$ java ThreeSum 16Kints.txt
^Cdhcp-10-5-14-188:lab4 njrom$ java ThreeSum 16Kints.txt
```


ThreeSum n vs T(n)
log(T(n))
10^2
10^1
10^0
10^3
2×10^3
3×10^3
4×10^3
6×10^3
log(n)

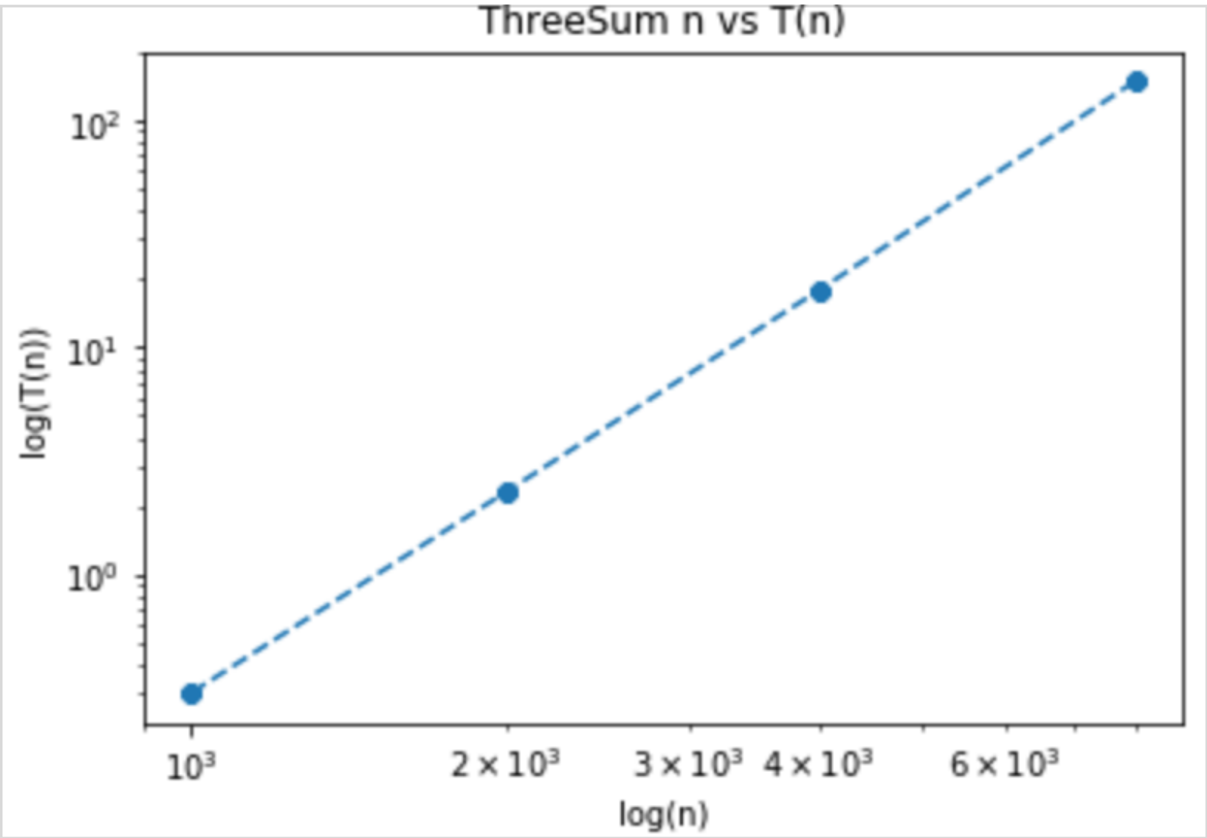
2.

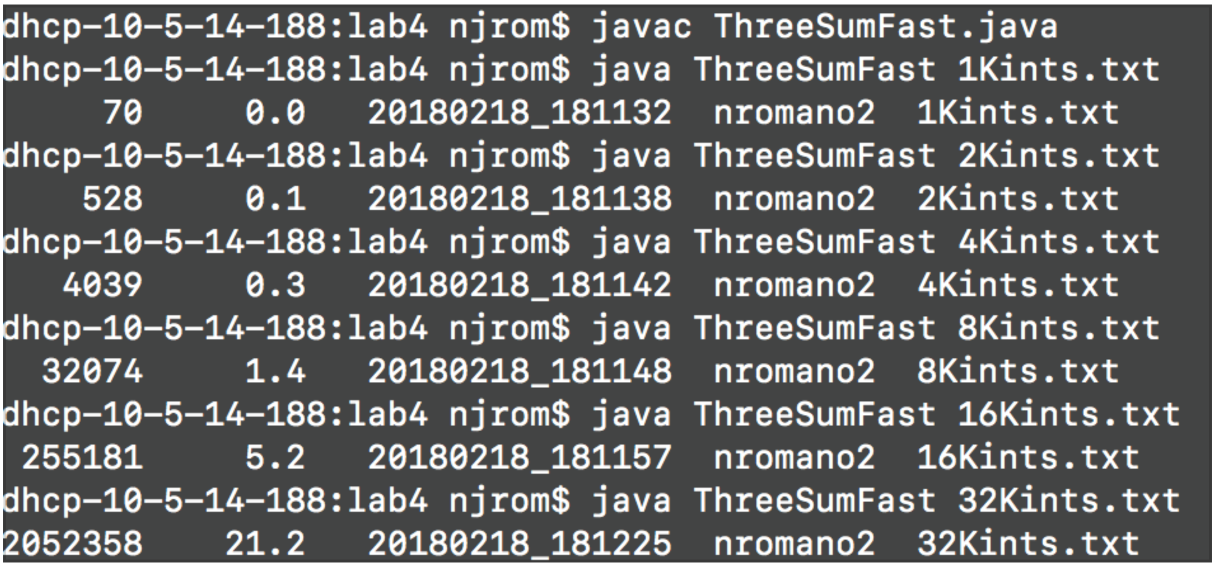

TwoSumFast n vs T(n)
T(n)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
10^3
10^4
10^5
10^6
log(n)



```
[^Cdhcp-10-5-14-188:lab4 njrom$ javac TwoSumFast.java
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 1Kints.txt
      1      0.0   20180218_180953  nromano2  1Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 2Kints.txt
      2      0.0   20180218_180957  nromano2  2Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 4Kints.txt
      3      0.0   20180218_181004  nromano2  4Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 8Kints.txt
     19      0.0   20180218_181010  nromano2  8Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 16Kints.txt
     66      0.0   20180218_181013  nromano2  16Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 32Kints.txt
    273      0.0   20180218_181019  nromano2  32Kints.txt
[dhcp-10-5-14-188:lab4 njrom$ java TwoSumFast 1Mints.txt
 249838      0.2   20180218_181033  nromano2  1Mints.txt
 dhcp-10-5-14-188:lab4 njrom$
```

3.

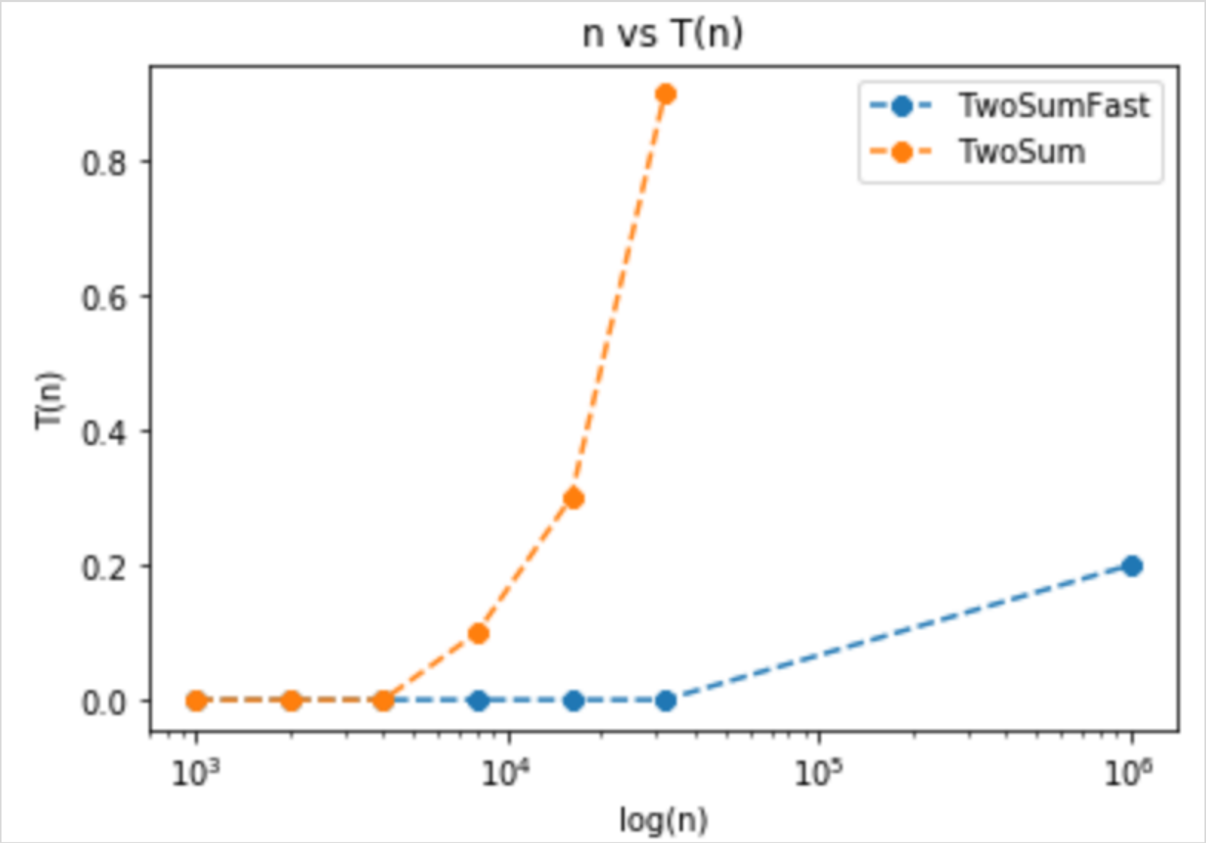




```
dhcp-10-5-14-188:lab4 njrom$ javac ThreeSumFast.java
dhcp-10-5-14-188:lab4 njrom$ java ThreeSumFast 1Kints.txt
     70      0.0    20180218_181132  nromano2  1Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSumFast 2Kints.txt
    528      0.1    20180218_181138  nromano2  2Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSumFast 4Kints.txt
   4039      0.3    20180218_181142  nromano2  4Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSumFast 8Kints.txt
  32074      1.4    20180218_181148  nromano2  8Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSumFast 16Kints.txt
 255181      5.2    20180218_181157  nromano2  16Kints.txt
dhcp-10-5-14-188:lab4 njrom$ java ThreeSumFast 32Kints.txt
2052358     21.2    20180218_181225  nromano2  32Kints.txt
```

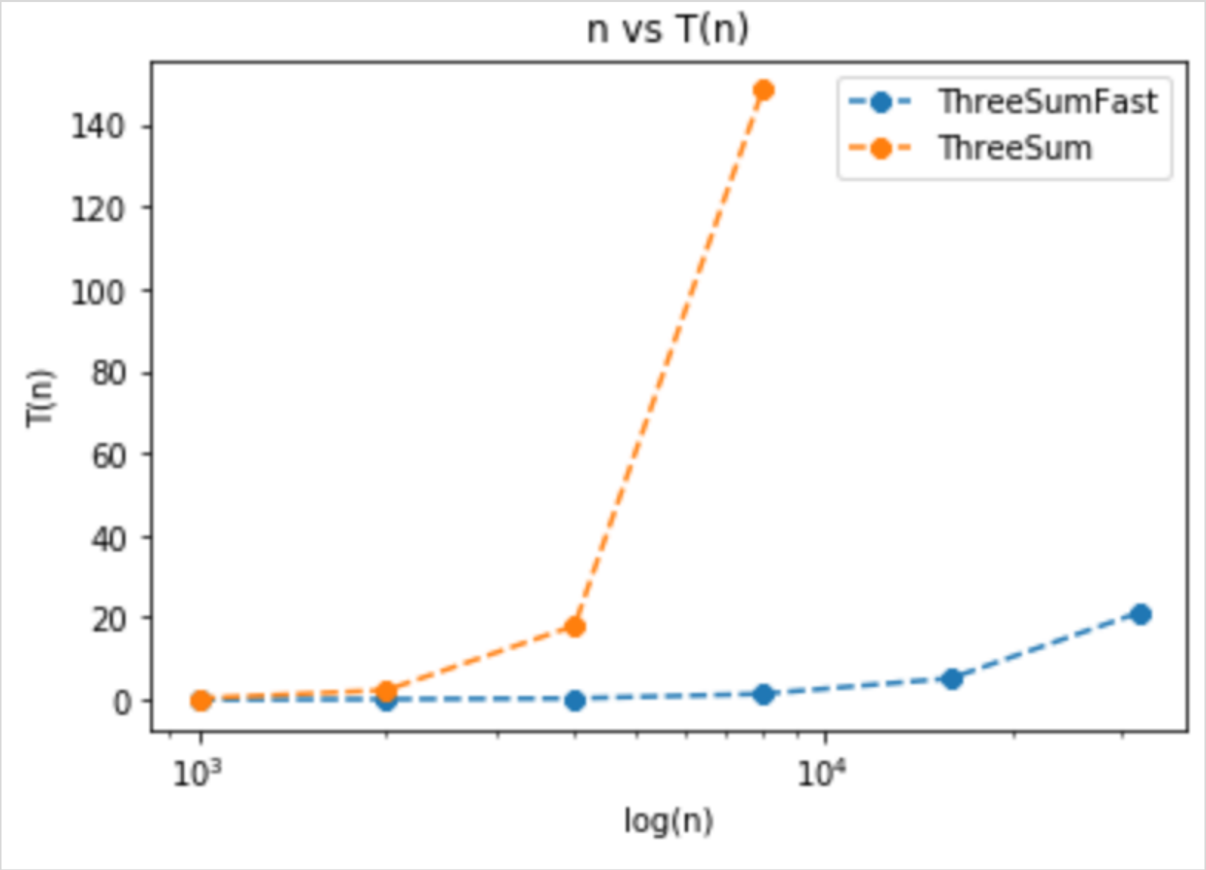
4.

5. It is very clear that as n increases TwoSumFast becomes much much more efficient.



6.

7. It is very clear that as n increases ThreeSumFast becomes much much more efficient. The difference from O(n^3) to O(n^2 log(n))



Task 3

1. TwoSum and TwoSumFast both run so fast with the 1Kint, 2Kint, and 4Kint files that the timer isn't printing enough sig figs to find a difference. Once n increases more than 4K we can start to see how the nLog(n) vs n^2 big-oh really matters. We know 16Kints takes around .3 seconds so c(16)^2 = .3 c = .0017 .0017(32)^2 = 1.1 seconds. The data shows it takes .9 seconds for 32k not 1.1, but this makes sense because the big-oh is a worst case upper bound for how long it should take. For 1Mints it should take .
0017(1000)^2 = 1700 seconds.

2. TwoSumsFast has a big Oh of nLog(n) so we cannot find the constant c to determine our estimate for 32k or 1M because every entry is 0 except for 1M
3. ThreeSum has a big oh of n^3 so we know c(4^3) = 18 seconds c = .2813 . So .
2813(32^3) = 9216 seconds and .2813(1000^3) = 2,812,500,000 seconds We didn't get a data points for results from 16k and up to compare with though, but these are the calculated estimates for 32Kint and 1Mint.
4. ThreeSumFast has a big oh of n^2 log(n) therefore c(4^2 log(4)) = .3 c = 0.0311 So for 32kints 0.0311(32^2 log(32)) = 19.17 seconds and the actual time was 21.2 which is interesting because it should have an upper bound of 19.17 according to the big-oh notation. For 1Mints we can estimate the time to be 0.0311(1000^2 log(1000)) = 93,300 seconds