Data Structures and Algorithm Analysis

Lab 2, measure the time of running programs.

Contents

- Measure the performance of three sum solver using algs4.
- Draw some graph with ploting software.
- Learn to improve ThreeSum algorithm.
- Basic of binary search.

The 3-sum problem

Recall the 3-sum problem you have seen in previous class.

The 3-sum problem

Given N distinct integers, how many triples sum to exactly zero?

You need to choose 3 number. They must sum to zero.

```
Example array:
  30, -40, -20, -10, 40, 0, 10, 5
```

a[i]	a[j]	a[k]	sum
30	-40	10	0
30	-20	-10	0
-40	40	0	0
-10	0	10	0

The 3-sum problem

We can easily calculate the 3-sum problem using algs4 library.

Just run the following code:

```
public static void main( String[] args ) {
  int[] array = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
  int count = ThreeSum.count(array);
  if( count <= 1 )
    StdOut.printf("There is %d triple in the array.\n",
      count);
  else
    StdOut.printf("There are %d triples in the array.\n",
      count);
  }
}
```

It's in the "Simple3SumTest.java" with the lab material.

The 3-sum problem

Let's see how the ThreeSum is implemented. On the algs4 site: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/ThreeSum.java.html

```
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;
```

It is the same triple for loop you have seen in the class tutorial.

Measure the time using Stopwatch

The algs4 library provides a Stopwatch class. It is used when you need to measure the time spent on some of your code.

```
Stopwatch stopwatch = new Stopwatch();
// Some Code you want to measure its time.
double elapsedTime = timer.elapsedTime();
```

Generate test data

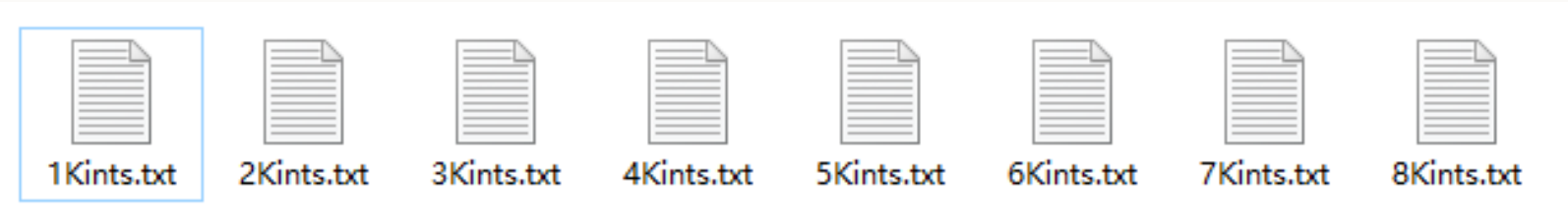
Before we test the the performance of ThreeSum. We may need to generate some test data.

The test data may be quite large, we don't recommand you to write them by hand.

You could write some code to generate random data and store them in a file to be used repeatedly.

Generate test data

A simple example is in "GenData.java" in the lab material. Run it and data files will be generated automatically.



The smallest "1Kints.txt" contains 1000 unique integers, while the largest "8Kint.txt" contains 8000 integers. You can freely modify the code to generate larger file.

Measure the time of ThreeSum

With what we already have we can easily measure the efficiency of the ThreeSum algorithm we just seen.

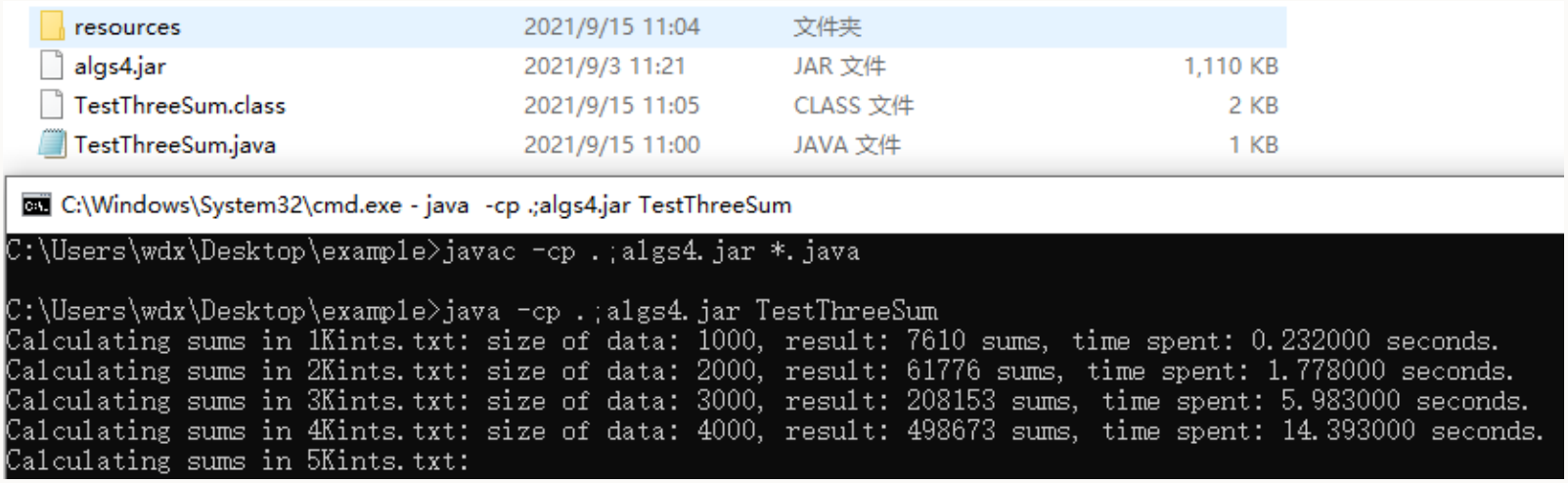
```
In fin = new In("./resources/data/"+i+"Kints.txt");
int[] arr = fin.readAllInts();
fin.close();

StdOut.printf("Calculating sums in %dKints.txt:", i);
Stopwatch timer = new Stopwatch();
int count = ThreeSum.count(arr);

StdOut.printf(" size of data: %d, result: %d sums, time
  spent: %f seconds.\n", arr.length, count, timer.
  elapsedTime());
```

Measure the time of ThreeSum

The above code is in the "ThreeSumTest.java" in the lab material. It reads data from the data files generated in the above step, one by one, and then run "ThreeSum" on the data. It uses Stopwatch to measure the time.



Measure the time of ThreeSum

The algs4 library especially provides a utility class "DoublingTest" for testing ThreeSum.

It uses a infinite loop to generate data and measure the time. Each time it doubles the data size.

You need nothing but algs4.jar to run it.

Measure the time of ThreeSum

For smaller data size, such as 250, 500, ..., the algorithm is fast. But as the data size increase, it became very slow. It takes 2 minutes for 8000 int data!

Measure the time of ThreeSum

As you have aquired a table of running time with respect of data size, you may want to draw a graph to show the relavance.

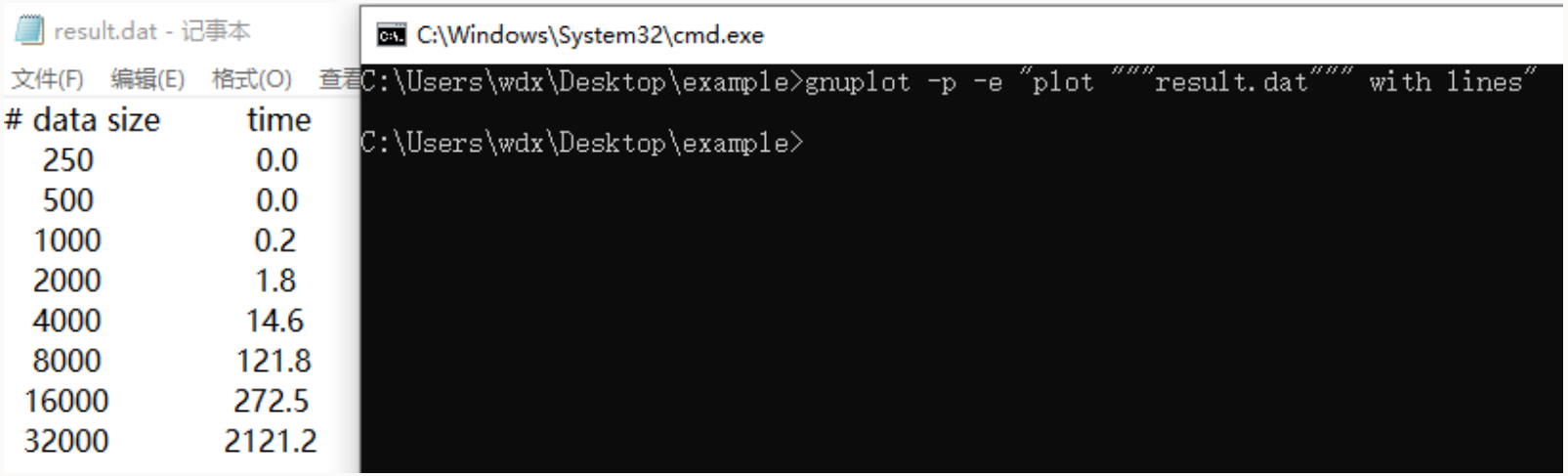
If you know how to program in Python, I may use matplotlib. https://matplotlib.org/

Otherwise, you could use gnuplot. http://www.gnuplot.info/

Just download the archive, uncompress to somewhere in your computer, and add the "bin" directory to your "Path" environment variable.

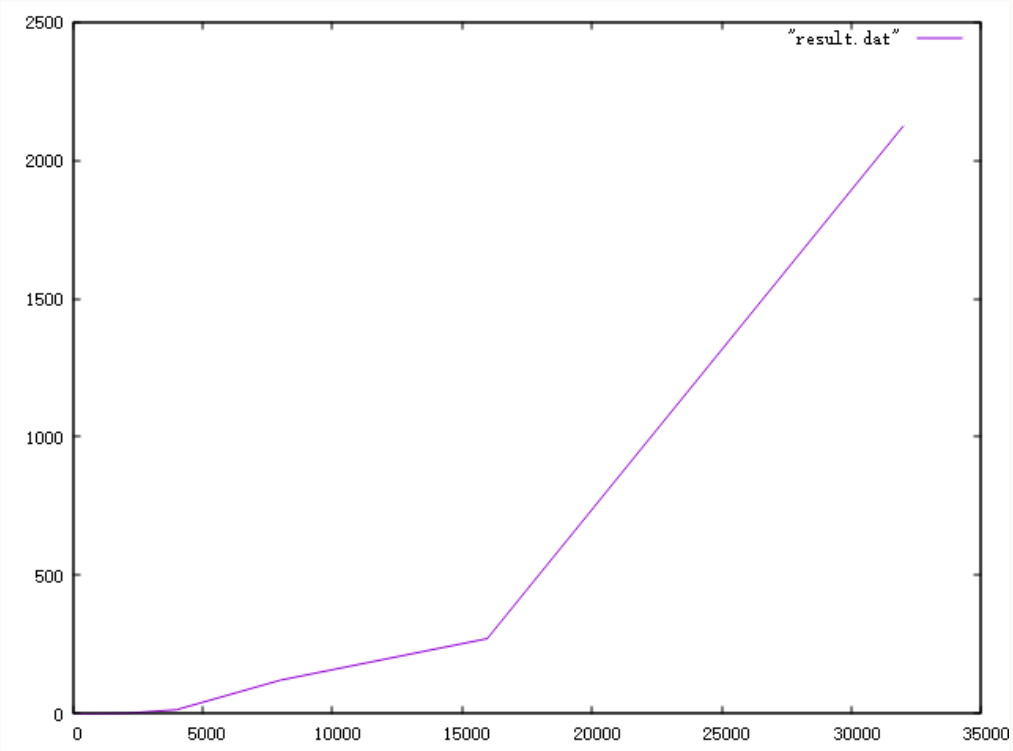
Measure the time of ThreeSum

Just put the data you want to draw into a file and run the following command. The graph will show automatically.



Measure the time of ThreeSum

Just put the data you want to draw into a file and run the following command. The graph will show automatically.



Measure the time of ThreeSum

The graph may look like something like the following. The x-axies is the data size, and y-axies is the time. Theoratically, it should look like a cubic function, $y = ax^3 + bx^2 + cx + d$, but in reality the curve may be unpredictable without a sufficiently large x.

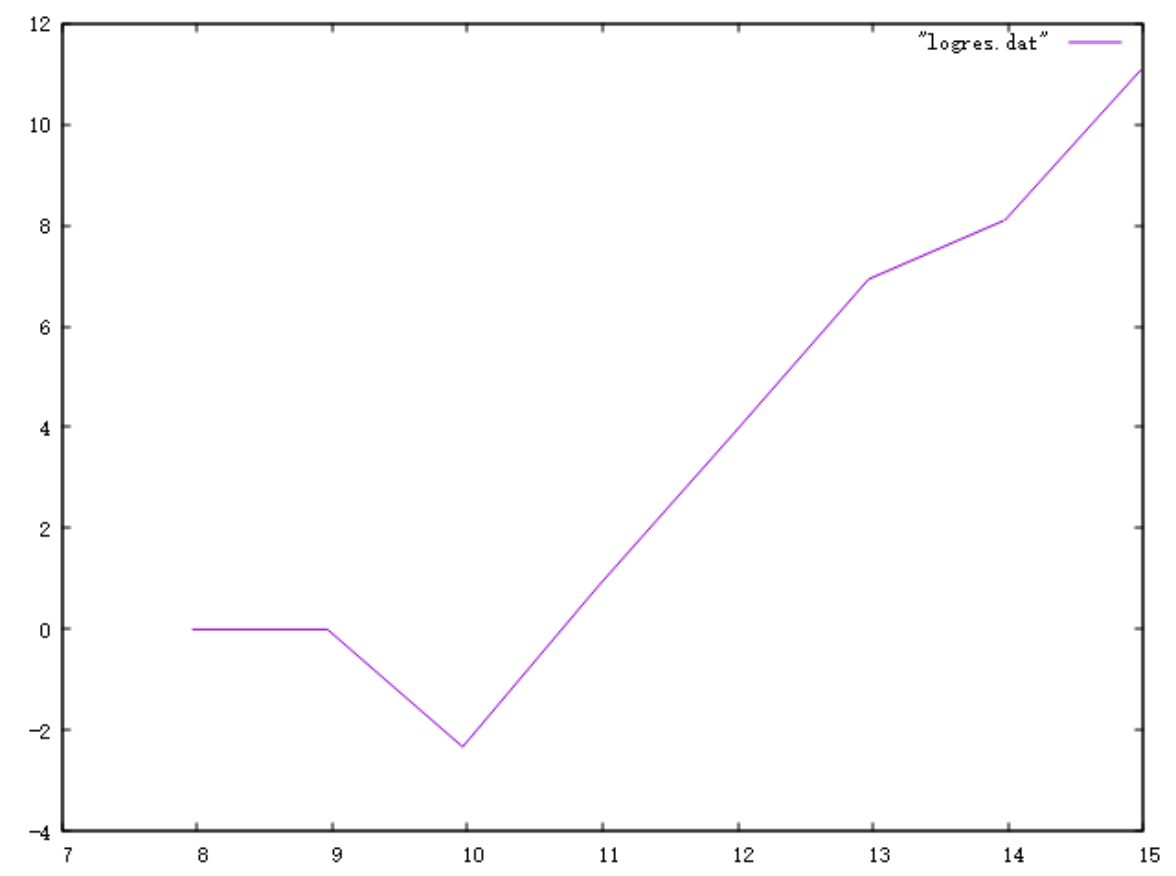
Measure the time in log-log scale

However, it is hard to tell what is the function from the curve above, so log-log scale is introduced in the class tutorial.

As x grows large enough, the curve can be seen as $y = ax^3$, we put log function on both sides, we have, $log(y) = 3log(x) + log(a)$, if we use $log(x)$ as the x coordinate and $log(y)$ as the x coordinate, then we get a straight line in the graph. Measuring the slope of the graph can give you a good hint about the time complexity.

Measure the time in log-log scale

Log-log scale:



Doubling hypothesis

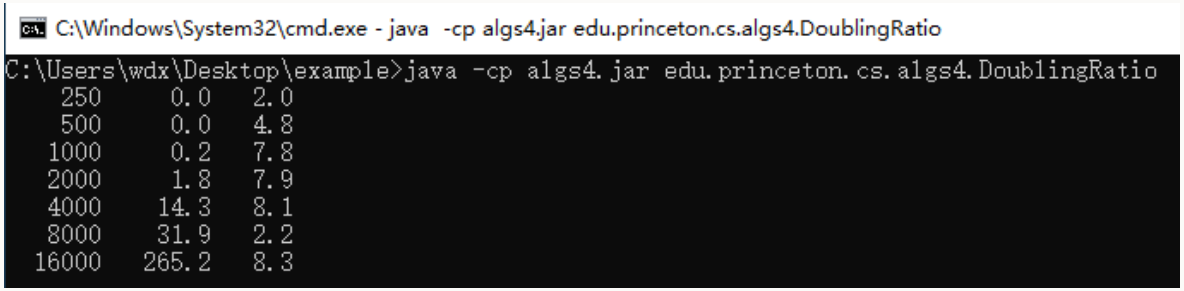
If the time(y) and the data size(x) have the relationship: $y = ax^3$, then we can draw a simple conclusion: if the data size is multiplied by 2, then time will multiple by 8.

It inspires people to run program multiple times, each time doubling the data size, and then calculate the ratio of time between each run.

The algs4 library provides the DoublingRatio class to do test.

Doubling hypothesis

From the figure below we can easily see that the ratio of running time between consecutive runs approaches 8 as data size increases[1].



[1]There are some exceptions however, the reason could be some JVM optimizations I do not know

Optimize Three Sum algorithm

The running time increase very fast with the increase of data size. Is there any way we can improve it?

Let's read the triple for loop again.

```
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;
```

Optimize Three Sum algorithm

It can be simplfied as follows:

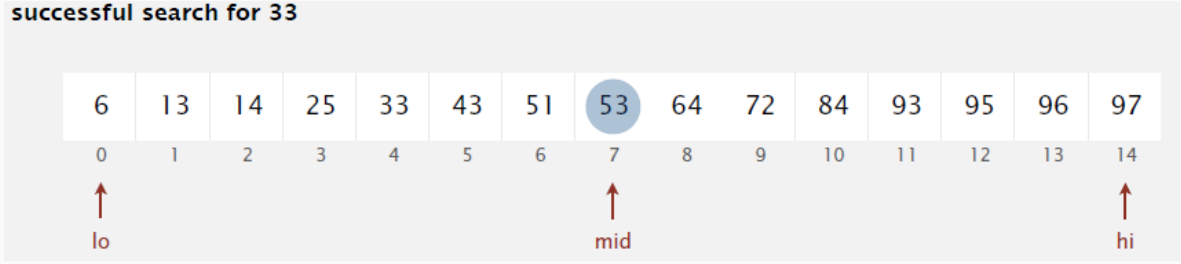
```
for (int i = 0; i < n; i++) {
  for (int j = i+1; j < n; j++) {
    "find an a[k] such that a[i]+a[j]+a[k] == 0 (j<k<n)"
  }
}
```

Question: do we really need to search the whole array to find the a[k]?

Optimize Three Sum algorithm

Question: do we really need to search the whole array to find the a[k]?

No, if the array is sorted, we can easily find a[k] with binary search.



Optimize Three Sum algorithm

The algorithm could be improved like the following:

```
Sort the array, small value on the left

for (int i = 0; i < n; i++) {
  for (int j = i+1; j < n; j++) {
    find a[k] with binary search within range [j+1, n)
  }
}
```

Use of binary search

The algs4 library is equiped with a binary search utility. Just use it like this:

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.BinarySearch;

public class TestBinarySearch {
    public static void main( String[] args ) {
        int[] arr = new int[] {2, 3, 5, 7, 11, 13, 17, 19};

        StdOut.printf("Search 7 in the array: %d\n",
          BinarySearch.indexOf(arr, 7));
        StdOut.printf("Search 100 in the array: %d\n",
          BinarySearch.indexOf(arr, 100));
    }
}
```

It's also in the lab2 material.

Optimize Three Sum algorithm

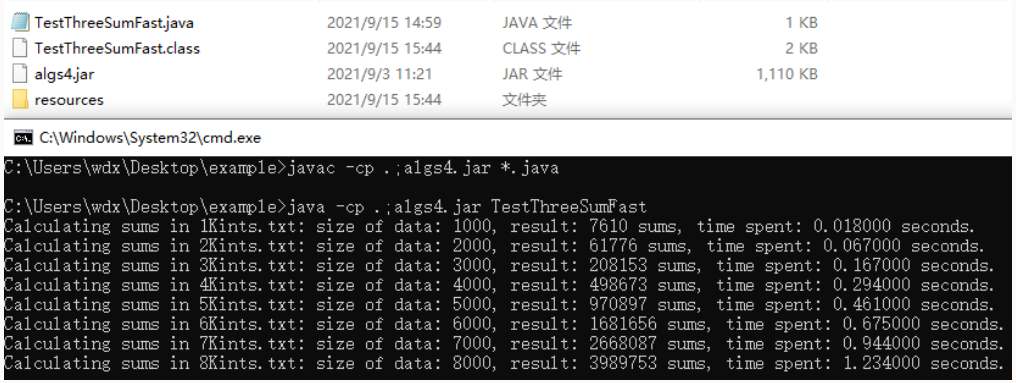
Now we are able to modify the previous ThreeSum.

```
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++;
  }
}
```

This is from the "ThreeSumFast" class in the algs4 library.

Optimize Three Sum algorithm

You can simply run "TestThreeSumFast" class in the lab material to test the ThreeSumFast. Note that is will use the same data files we used to test the triple for loop ThreeSum class.



It is much faster than the previous one. It finished processing 8000 data with more than 1 second.

Optimize Three Sum algorithm

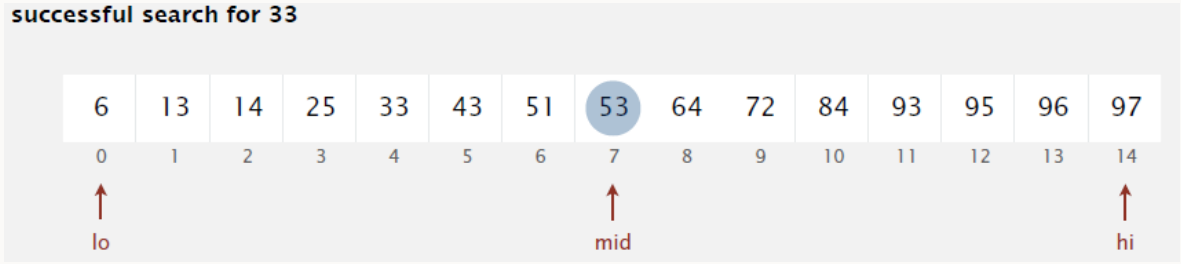
The "ThreeSumFast" algorithm with binary search is much better than the triple for loop version. It can finish 8000 data in approximately 1 second instead of 2 minites.

Except that it cannot handle duplications:

```
int n = a.length;
Arrays.sort(a);
if (containsDuplicates(a)) throw new
  IllegalArgumentException("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;
```

Optimize Three Sum algorithm

The reason is that the original binary search does not care about the range of values, it only find 1 target.



Optimize Three Sum algorithm

If dealing with duplications is necessary, the binary search should be changed to search for a range of targets.

What we have:

```
for (int i = 0; i < n; i++) {
  for (int j = i+1; j < n; j++) {
    find a[k] with binary search within range [j+1, n)
  }
}
```

What we need:

```
for (int i = 0; i < n; i++) {
  for (int j = i+1; j < n; j++) {
    find "All" a[k]s with binary search in [j+1, n)
  }
}
```

Optimize Three Sum algorithm

In our lab material, we also implemented a binary search that searches a range, named "BinaryRangeSearch":

With that you can implement a fast 3-sum algorithm which handles large data with duplications.

Optimize Three Sum algorithm

Is there any further improments? Let's see the triple for loop again:

```
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++;
      }
    }
  }
}
```

We reduce it to:

```
for (int i = 0; i < n; i++) {
  Find a[j] and a[k] such that a[j]+a[k] == -a[i]
}
```

Optimize Three Sum algorithm

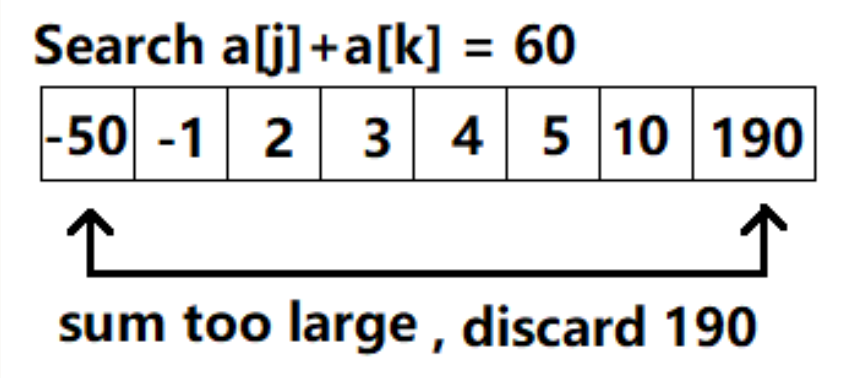
The major problem is that:

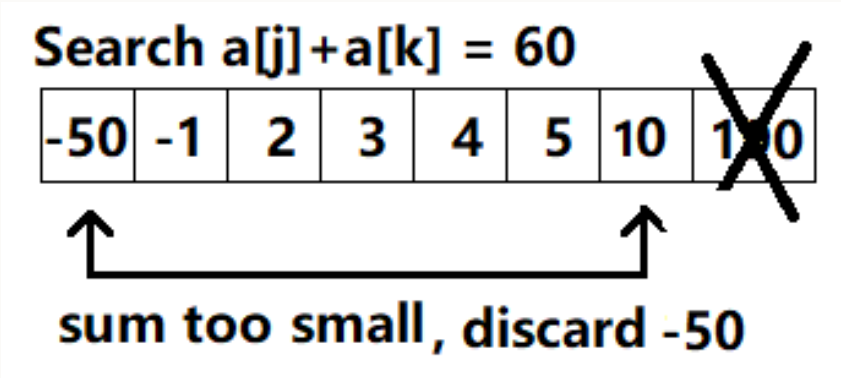
```
Find a[j] and a[k] such that a[j]+a[k] == -a[i]
```

Do we need to search the array again and again? No!

Optimize Three Sum algorithm

Find a[j] and a[k] such that a[j]+a[k] == -a[i]





Optimize Three Sum algorithm

After the above demonstration, you should be able to write 3-sum algorithm and measure its efficiency.

In class Exercise

Test the running time of the ThreeSum algorithm using any method described today.

If you have time you could compare different versions of 3-sum algorithm.

If you still have time you could write your own version!