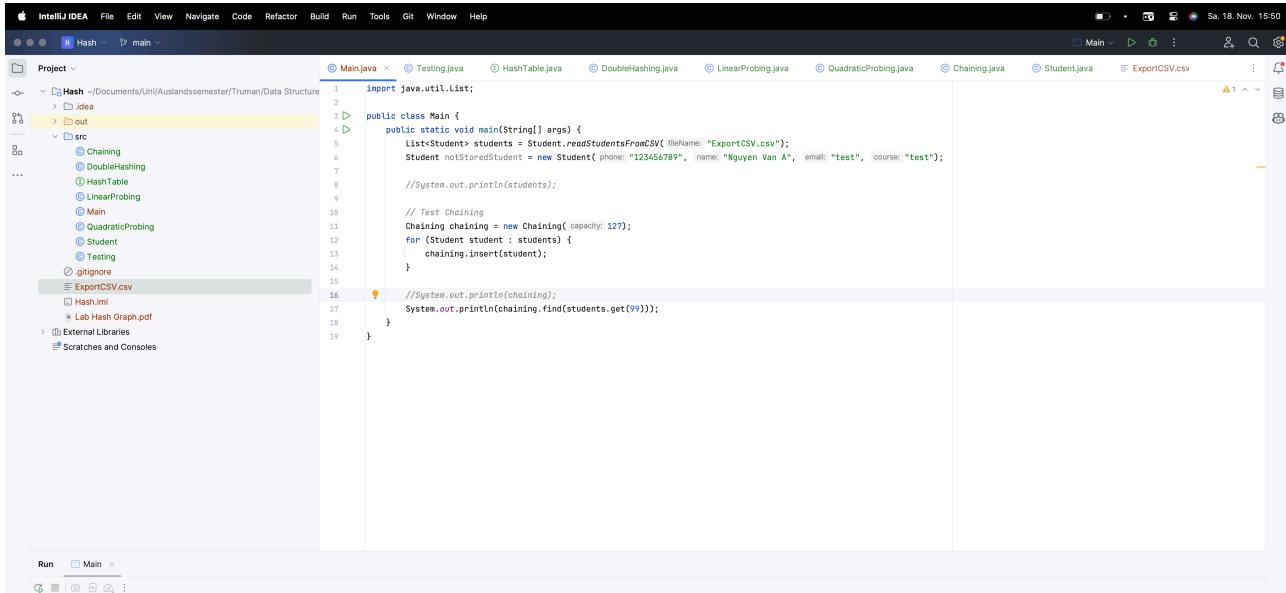


# **Deliverable 1 - 3**

Code

# Deliverable 4

## Chaining



The screenshot shows the IntelliJ IDEA interface with the Main.java file open. The code implements a chaining hash table. It reads student data from a CSV file, creates a student object, and inserts it into the hash table. The find method is used to search for a specific student by phone number.

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Student> students = Student.readStudentsFromCSV(fileName: "ExportCSV.csv");
        Student notStoredStudent = new Student(phone: "123456789", name: "Nguyen Van A", email: "test", course: "test");

        //System.out.println(students);

        // Test Chaining
        Chaining chaining = new Chaining(capacity: 127);
        for (Student student : students) {
            chaining.insert(student);
        }

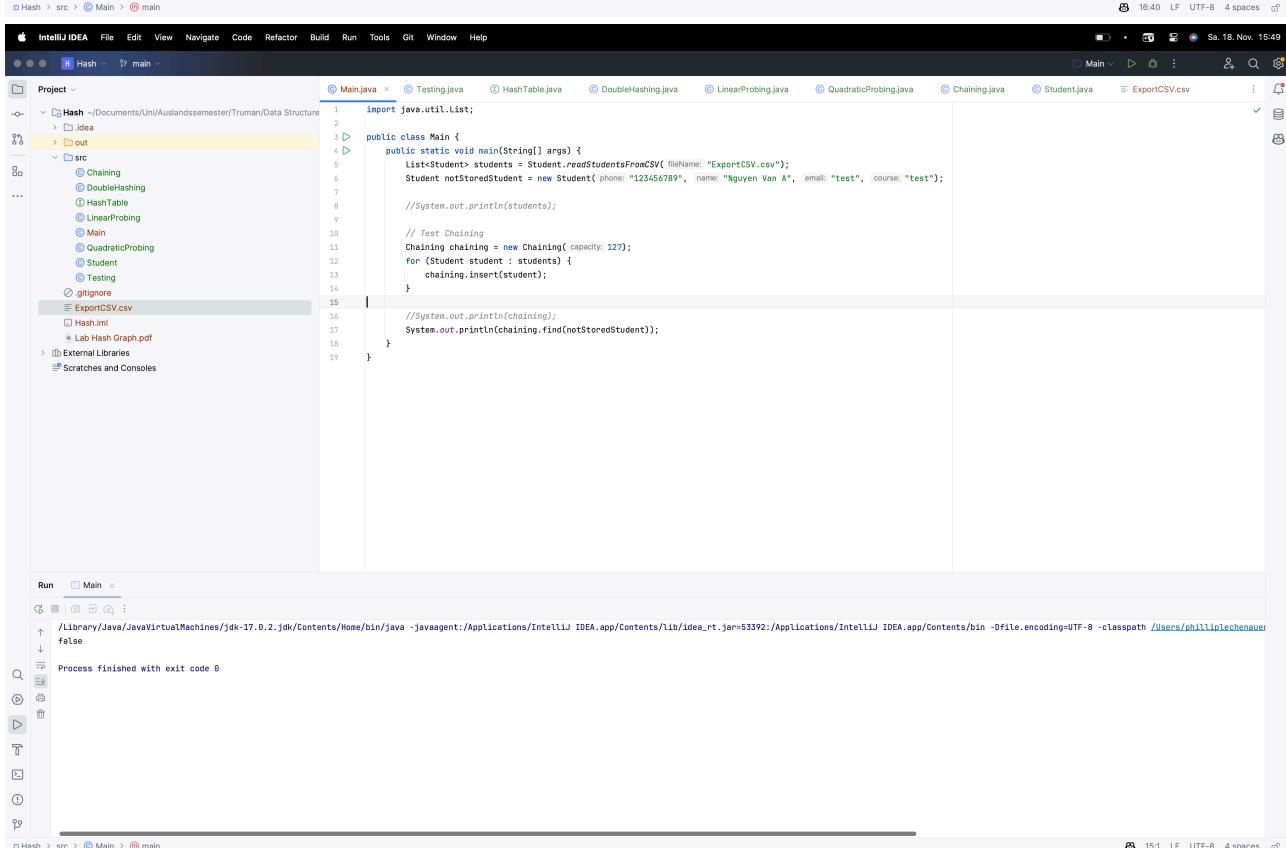
        //System.out.println(chaining);
        System.out.println(chaining.find(notStoredStudent));
    }
}
```



The run output shows the command run and the process finishing with exit code 0.

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53429:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/philliplechenauer/Hash/target/classes
```

Process finished with exit code 0



The run output shows the command run and the process finishing with exit code 0. The output is identical to the first execution.

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53392:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/philliplechenauer/Hash/target/classes
```

Process finished with exit code 0

# linear Probing

The screenshot shows the IntelliJ IDEA interface with the project 'Hash' selected. The 'src' directory contains several Java files: Chaining, DoubleHashing, HashTable, LinearProbing, Main, QuadraticProbing, Student, and Testing. The 'Main.java' file is open in the editor, showing the following code:

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Student> students = Student.readStudentsFromCSV( fileName: "ExportCSV.csv", limit: 100);
        Student notStoredStudent = new Student( phone: "123456789", name: "Nguyen Van A", email: "test", course: "test");

        // Test LinearProbing
        LinearProbing linearProbing = new LinearProbing( capacity: 127);
        for (Student student : students) {
            linearProbing.insert(student);
        }
        System.out.println(linearProbing);
        System.out.println(linearProbing.find(notStoredStudent));
    }
}
```

The 'Run' tool window shows the command used to run the application: '/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=53545:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/philliplechenauer'. The output indicates a process finished with exit code 0.

The screenshot shows the IntelliJ IDEA interface with the project 'Hash' selected. The 'src' directory contains several Java files: Chaining, DoubleHashing, HashTable, LinearProbing, Main, QuadraticProbing, Student, and Testing. The 'Main.java' file is open in the editor, showing the following code:

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Student> students = Student.readStudentsFromCSV( fileName: "ExportCSV.csv", limit: 100);
        Student notStoredStudent = new Student( phone: "123456789", name: "Nguyen Van A", email: "test", course: "test");

        // Test LinearProbing
        LinearProbing linearProbing = new LinearProbing( capacity: 127);
        for (Student student : students) {
            linearProbing.insert(student);
        }
        System.out.println(linearProbing);
        System.out.println(linearProbing.find(students.get(99)));
    }
}
```

A yellow dot marker highlights the line 'System.out.println(linearProbing.find(students.get(99)));'. The 'Run' tool window shows the command used to run the application: '/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=53561:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/philliplechenauer'. The output indicates a process finished with exit code 0.

# Quadratic Probing

The screenshot shows two instances of the IntelliJ IDEA IDE running side-by-side. Both instances display the same Java code for quadratic probing, which reads student data from a CSV file and inserts it into a hash table using quadratic probing.

**Code Snippet:**

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Student> students = Student.readStudentsFromCSV(fileName: "ExportCSV.csv", limit: 100);
        Student notStoredStudent = new Student(phone: "123456789", name: "Nguyen Van A", email: "test", course: "test");

        // Test QuadraticProbing
        QuadraticProbing quadraticProbing = new QuadraticProbing(capacity: 127);
        for (Student student : students) {
            quadraticProbing.insert(student);
        }
        //System.out.println(quadraticProbing);
        System.out.println(quadraticProbing.find(notStoredStudent));
    }
}
```

**Run Tab Output (Left Window):**

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53612:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/philliplechenauer/Downloads/Hash/Hash.iml
false
Process finished with exit code 0
```

**Run Tab Output (Right Window):**

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53619:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/philliplechenauer/Downloads/Hash/Hash.iml
true
Process finished with exit code 0
```

# Double Hashing

The screenshot shows the IntelliJ IDEA interface with the project 'Hash' open. The 'src' directory contains several Java files: Main.java, Testing.java, HashTable.java, DoubleHashing.java, LinearProbing.java, QuadraticProbing.java, Chaining.java, Student.java, and ExportCSV.csv. The Main.java file is selected and shown in the editor. It imports java.util.List and defines a Main class with a static void main method. The method reads students from 'ExportCSV.csv' into a list, creates a DoubleHashing object with capacity 127, iterates through the list, inserts each student into the hash table, and then prints the result. The output window shows the command run and the process finishing with exit code 0.

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Student> students = Student.readStudentsFromCSV(fileName: "ExportCSV.csv", limit: 100);
        Student notStoredStudent = new Student(phone: "123456789", name: "Nguyen Van A", email: "test", course: "test");

        // Test DoubleHashing
        DoubleHashing doubleHashing = new DoubleHashing(capacity: 127);
        for (Student student : students) {
            doubleHashing.insert(student);
        }
        //System.out.println(doubleHashing);
        System.out.println(doubleHashing.find(notStoredStudent));
    }
}
```

This screenshot is nearly identical to the one above, showing the same project structure and Main.java code. However, there is a yellow warning icon in the top right corner of the editor area. The output window shows the command run and the process finishing with exit code 0.

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Student> students = Student.readStudentsFromCSV(fileName: "ExportCSV.csv", limit: 100);
        Student notStoredStudent = new Student(phone: "123456789", name: "Nguyen Van A", email: "test", course: "test");

        // Test DoubleHashing
        DoubleHashing doubleHashing = new DoubleHashing(capacity: 127);
        for (Student student : students) {
            doubleHashing.insert(student);
        }
        //System.out.println(doubleHashing);
        System.out.println(doubleHashing.find(students.get(99)));
    }
}
```

## Deliverable 5:

### Chaining

Hashtable size	Input size	insert steps	find steps	total steps	Average Steps
1100	1000	1000	1650	2650	1,325
2100	2000	2000	3420	5420	1,355
3100	3000	3000	5196	8196	1,366
4100	4000	4000	6663	10663	1,332875
5100	5000	5000	8422	13422	1,3422

### Linear Probing

Hashtable size	Input size	insert steps	find steps	total steps	Average Steps
1100	1000	12964	43600	56564	28,282
2100	2000	46101	360973	407074	101,7685
3100	3000	116981	1284596	1401577	233,5961666666
4100	4000	109723	1019662	1129385	141,173125
5100	5000	312458	3772307	4084765	408,4765

### Double Hashing

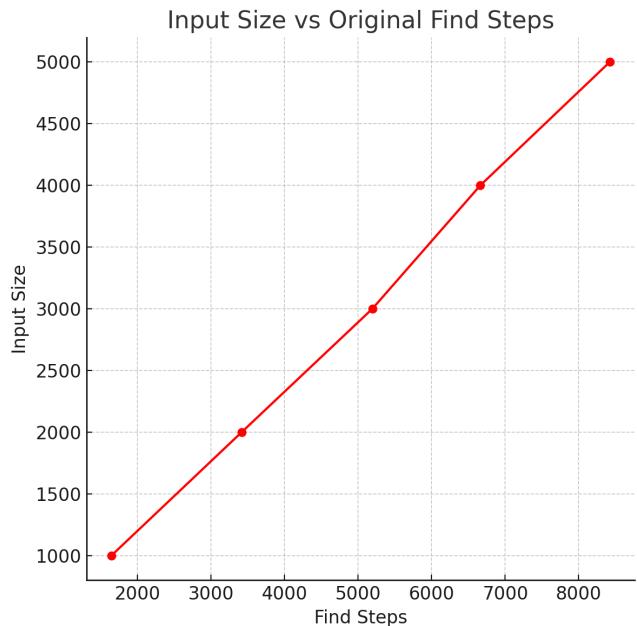
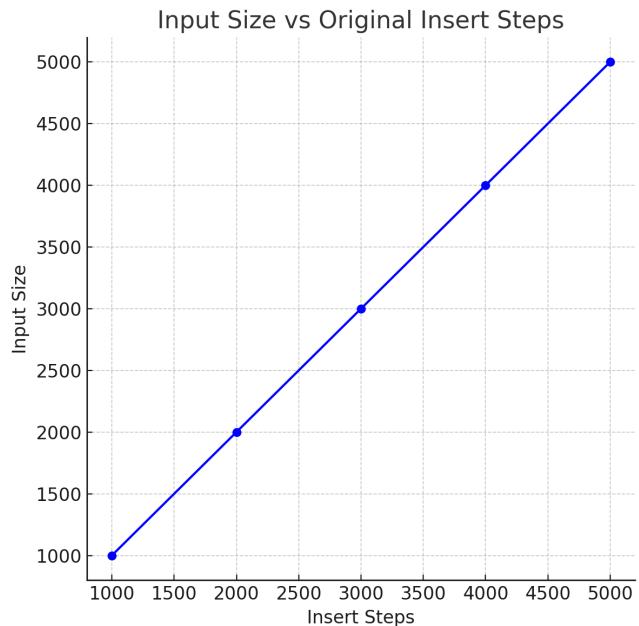
Hashtable size	Input size	insert steps	find steps	total steps	Average Steps
1100	1000	78094709	300980328	379075037	189537,5185
2100	2000	53697434	1278947102	1332644536	333161,134
3100	3000	55435030	570593904	626028934	104338,1556666
4100	4000	22107	411520843	411542950	51442,86875
5100	5000	56875265	2372158631	2429033896	242903,3896

### QuadraticProbing

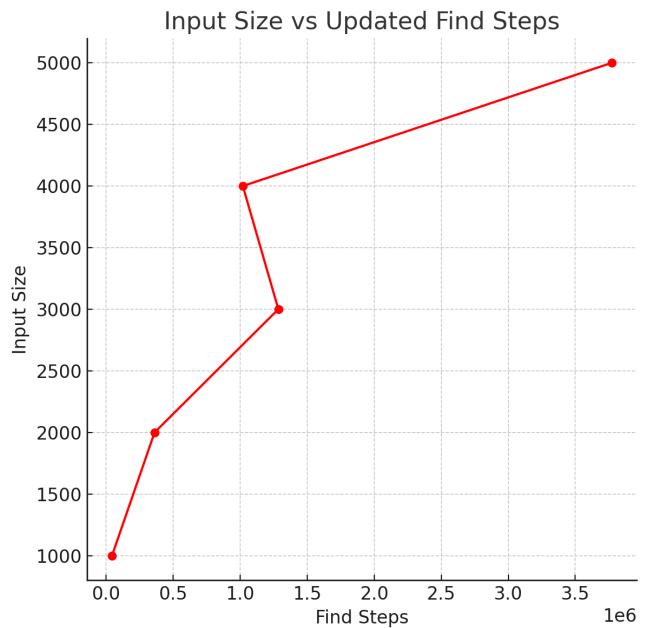
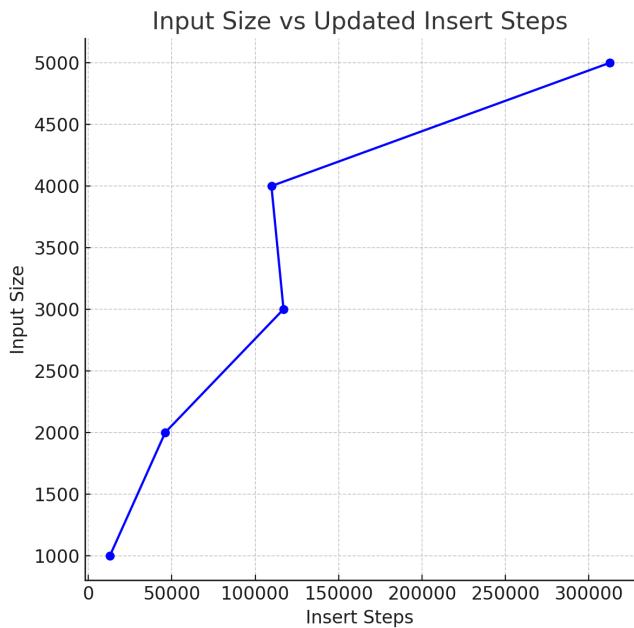
Hashtable size	Input size	insert steps	find steps	total steps	Average Steps
1100	1000	4856	8072	12928	6,464
2100	2000	10533	28363	38896	9,724
3100	3000	17927	67475	85402	14,23366666666
4100	4000	24532	108811	133343	16,667875
5100	5000	35463	163305	198768	19,8768

# Deliverable 6:

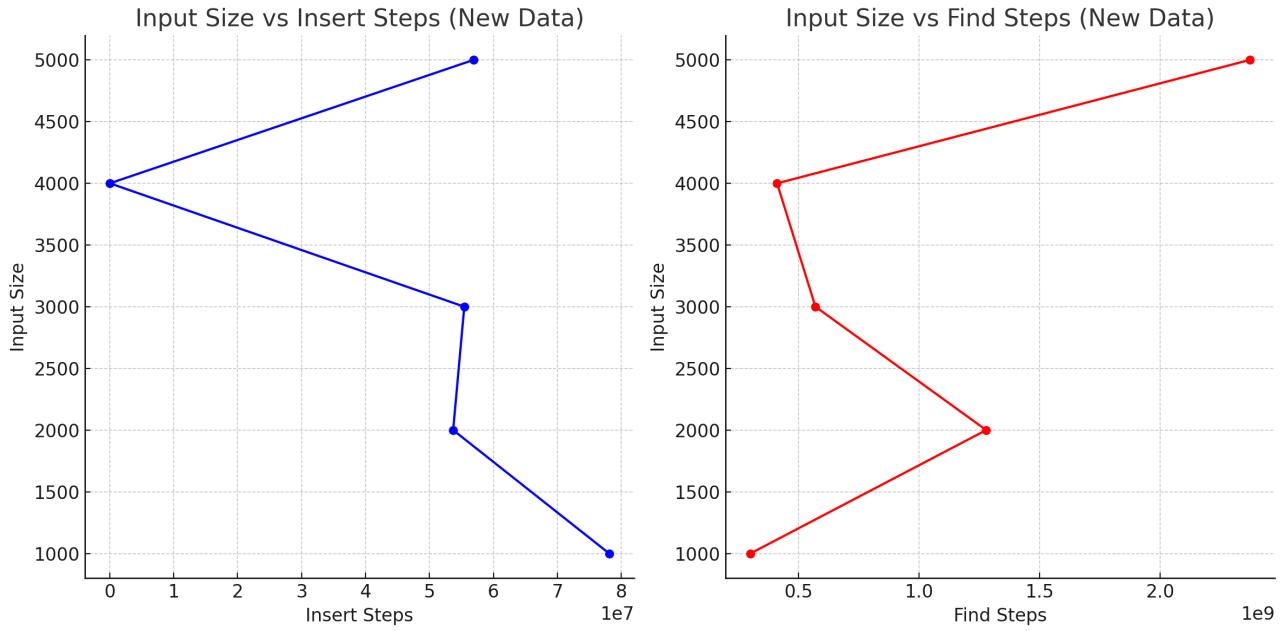
## Chaining



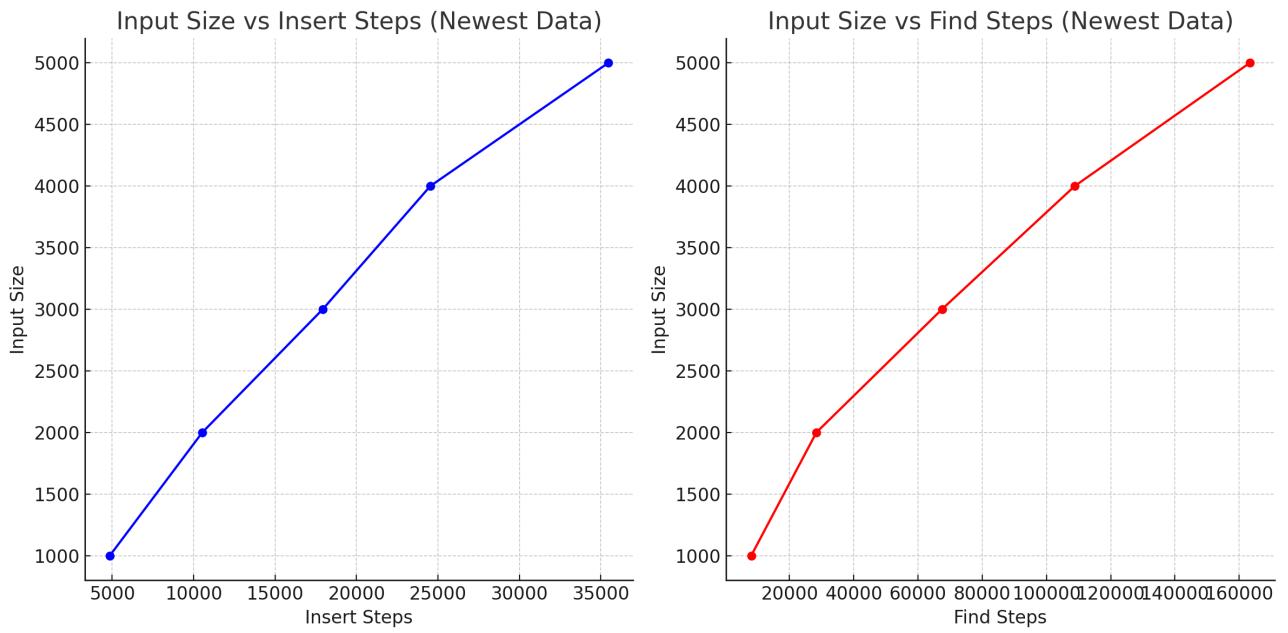
## Linear Probing



## Double Hashing



## Quadratic Probing



- a) Which hash table algorithm requires the least number of steps to store the Student objects  
Comment why that is the case.  
Chaining, because it just uses the right index and then appends it at the linked list. There is no loop.
- b) Which hash table algorithm requires the most number of steps to store the Student objects?  
Comment why that is the case.  
Double Hashing, probably because my second hashing function was poorly designed.
- c) Compare the performance of the hash tables when the input size was 1000 compared to 5000.  
For most of them the performance got worse
- d) Why does the insert algorithm require more steps to store the objects when the input size is 5000 compared to the input size 1000?  
Because we have more elements to place. We also increased the size of the table, but it was always 100 bigger than the input size, which is 10% for 1000 but only 2% for 5000. Therefore, we have less space.