Analysis of Algorithms

Outline

Algorithmic Complexity

Algorithmic Complexity

Time complexity of a program is how long it will take to solve a problem

Algorithmic Complexity

Time complexity of a program is how long it will take to solve a problem

Space complexity of a program is how much memory it will need to solve a problem

Time Complexity

Time Complexity

Program: `TripleSum.java`

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs
$ _
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs
$ cat ../data/1Kints.txt
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ _
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ /usr/bin/time -f "%es" java TripleSum ../data/1Kints.txt
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ /usr/bin/time -f "%es" java TripleSum ../data/1Kints.txt
70
0.28s
$ _
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ /usr/bin/time -f "%es" java TripleSum ../data/1Kints.txt
70
0.28s
$ /usr/bin/time -f "%es" java TripleSum ../data/2Kints.txt
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ /usr/bin/time -f "%es" java TripleSum ../data/1Kints.txt
70
0.28s
$ /usr/bin/time -f "%es" java TripleSum ../data/2Kints.txt
528
1.80s
$ _
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ /usr/bin/time -f "%es" java TripleSum ../data/1Kints.txt
70
0.28s
$ /usr/bin/time -f "%es" java TripleSum ../data/2Kints.txt
528
1.80s
$ /usr/bin/time -f "%es" java TripleSum ../data/4Kints.txt
```

Time Complexity

Program: `TripleSum.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ cat ../data/1Kints.txt
 324110
-442472
...
 745942
$ /usr/bin/time -f "%es" java TripleSum ../data/1Kints.txt
70
0.28s
$ /usr/bin/time -f "%es" java TripleSum ../data/2Kints.txt
528
1.80s
$ /usr/bin/time -f "%es" java TripleSum ../data/4Kints.txt
4039
14.06s
$ _
```

Time Complexity

Time Complexity

TripleSum.java

```java
import stdlib.In;
import stdlib.StdOut;

public class TripleSum {
    public static void main(String[] args) {
        In in = new In(args[0]);
        int[] a = in.readAllInts();
        StdOut.println(count(a));
    }

    private 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;
    }
}
```

Time Complexity

Time Complexity

Experimental analysis

n	$f(n)$
1K	0.28s
2K	1.8s
4K	14.06s
8K	111.83s
16K	892.19s

Time Complexity

Experimental analysis

n	$f(n)$
1K	0.28s
2K	1.8s
4K	14.06s
8K	111.83s
16K	892.19s

$$f(n) = 0.2273121n^3 + 0.007625303n^2 + 0.006868505n + 0.01817256$$

Time Complexity

Time Complexity

The function $g(n)$ is called the tilde approximation of the function $f(n)$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 1$$

Time Complexity

The function $g(n)$ is called the tilde approximation of the function $f(n)$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 1$$

For example, if $f(n) = 31n^2 + 78n + 42$, then $g(n) = 31n^2$

Time Complexity

The function $g(n)$ is called the tilde approximation of the function $f(n)$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 1$$

For example, if $f(n) = 31n^2 + 78n + 42$, then $g(n) = 31n^2$

We often work with tilde approximations of the form $g(n) = an^b(\log n)^c$, where a, b, and c are constants

Time Complexity

The function $g(n)$ is called the tilde approximation of the function $f(n)$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 1$$

For example, if $f(n) = 31n^2 + 78n + 42$, then $g(n) = 31n^2$

We often work with tilde approximations of the form $g(n) = an^b(\log n)^c$, where a, b, and c are constants

We refer to the function $T(n) = n^b(\log n)^c$ as the running time

Time Complexity

The function $g(n)$ is called the tilde approximation of the function $f(n)$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 1$$

For example, if $f(n) = 31n^2 + 78n + 42$, then $g(n) = 31n^2$

We often work with tilde approximations of the form $g(n) = an^b(\log n)^c$, where a, b, and c are constants

We refer to the function $T(n) = n^b(\log n)^c$ as the running time

For example, if $g(n) = 31n^2$, then $T(n) = n^2$

Time Complexity

The function $g(n)$ is called the tilde approximation of the function $f(n)$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 1$$

For example, if $f(n) = 31n^2 + 78n + 42$, then $g(n) = 31n^2$

We often work with tilde approximations of the form $g(n) = an^b(\log n)^c$, where a, b, and c are constants

We refer to the function $T(n) = n^b(\log n)^c$ as the running time

For example, if $g(n) = 31n^2$, then $T(n) = n^2$

For the Three Sum problem, $T(n) = n^3$

Time Complexity

Time Complexity

Mathematically, we compute a function $f(n)$ from:

Time Complexity

Mathematically, we compute a function $f(n)$ from:

1. The cost of executing each statement (property of the computer)

Time Complexity

Mathematically, we compute a function $f(n)$ from:

1. The cost of executing each statement (property of the computer)
2. The frequency of execution of each statement (property of the program and the input)

Time Complexity

Time Complexity

```
private static int count(int[] a) {
    int n = a.length;                                   [A]
    int count = 0;
    for (int i = 0; i < n; i++) {                       [B]
        for (int j = i + 1; j < n; j++) {               [C]
            for (int k = j + 1; k < n; k++) {           [D]
                if (a[i] + a[j] + a[k] == 0) {
                    count++;                            [E]
                }
            }
        }
    }
    return count;
}
```

Time Complexity

```
private static int count(int[] a) {
    int n = a.length;                                   [A]
    int count = 0;
    for (int i = 0; i < n; i++) {                       [B]
        for (int j = i + 1; j < n; j++) {               [C]
            for (int k = j + 1; k < n; k++) {           [D]
                if (a[i] + a[j] + a[k] == 0) {
                    count++;                            [E]
                }
            }
        }
    }
    return count;
}
```

Statement Block	Time	Frequency	Total Time
$[A]$	t_4	1	t_4
$[B]$	t_3	n	$t_3 n$
$[C]$	t_2	$\binom{n}{2} = n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
$[D]$	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
$[E]$	t_0	x (depends on input)	$t_0 x$

Time Complexity

```
private static int count(int[] a) {
    int n = a.length;                                   [A]
    int count = 0;
    for (int i = 0; i < n; i++) {                       [B]
        for (int j = i + 1; j < n; j++) {               [C]
            for (int k = j + 1; k < n; k++) {           [D]
                if (a[i] + a[j] + a[k] == 0) {
                    count++;                            [E]
                }
            }
        }
    }
    return count;
}
```

Statement Block	Time	Frequency	Total Time
$[A]$	t_4	1	t_4
$[B]$	t_3	n	$t_3 n$
$[C]$	t_2	$\binom{n}{2} = n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
$[D]$	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
$[E]$	t_0	x (depends on input)	$t_0 x$

$$f(n) = (t_1/6)n^3 + (t_2/2 - t_1/2)n^2 + (t_1/3 - t_2/2 + t_3)n + t_4 + t_0 x$$

Time Complexity

```java
private static int count(int[] a) {
    int n = a.length;                                   [A]
    int count = 0;
    for (int i = 0; i < n; i++) {                       [B]
        for (int j = i + 1; j < n; j++) {               [C]
            for (int k = j + 1; k < n; k++) {           [D]
                if (a[i] + a[j] + a[k] == 0) {
                    count++;                            [E]
                }
            }
        }
    }
    return count;
}
```

Statement Block	Time	Frequency	Total Time
$[A]$	t_4	1	t_4
$[B]$	t_3	n	$t_3 n$
$[C]$	t_2	$\binom{n}{2} = n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
$[D]$	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
$[E]$	t_0	x (depends on input)	$t_0 x$

$$f(n) = (t_1/6)n^3 + (t_2/2 - t_1/2)n^2 + (t_1/3 - t_2/2 + t_3)n + t_4 + t_0 x$$

$$g(n) = (t_1/6)n^3$$

Time Complexity

```java
private static int count(int[] a) {
    int n = a.length;                                   [A]
    int count = 0;
    for (int i = 0; i < n; i++) {                       [B]
        for (int j = i + 1; j < n; j++) {               [C]
            for (int k = j + 1; k < n; k++) {           [D]
                if (a[i] + a[j] + a[k] == 0) {
                    count++;                            [E]
                }
            }
        }
    }
    return count;
}
```

Statement Block	Time	Frequency	Total Time
$[A]$	t_4	1	t_4
$[B]$	t_3	n	$t_3 n$
$[C]$	t_2	$\binom{n}{2} = n^2/2 - n/2$	$t_2(n^2/2 - n/2)$
$[D]$	t_1	$\binom{n}{3} = n^3/6 - n^2/2 + n/3$	$t_1(n^3/6 - n^2/2 + n/3)$
$[E]$	t_0	x (depends on input)	$t_0 x$

$$f(n) = (t_1/6)n^3 + (t_2/2 - t_1/2)n^2 + (t_1/3 - t_2/2 + t_3)n + t_4 + t_0 x$$

$$g(n) = (t_1/6)n^3$$

$$T(n) = n^3$$

Time Complexity

Time Complexity

Running time classifications

Name	$T(n)$	Code Description	Example
constant	1	statement	increment the ith element in an array
logarithmic	$\log n$	divide and discard	binary search
linear	n	loop	find the maximum
linearithmic	$n \log n$	divide and conquer	merge sort
quadratic	n^2	double loop	check all ordered pairs
cubic	n^3	triple loop	check all ordered triples
exponential	2^n	exhaustive search	check all subsets

Time Complexity

Time Complexity

dsa.LinearSearch	
`static int indexOf(Object[] a, Object key)`	returns the index of `key` in the array `a`, or -1
`static int indexOf(int[] a, int key)`	returns the index of `key` in the array `a`, or -1
`static int indexOf(double[] a, double key)`	returns the index of `key` in the array `a`, or -1

Time Complexity

Time Complexity

Program: `LinearSearch.java`

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ _
```

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs
$ cat ../data/tinyW.txt
```

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ _
```

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
```

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
23
50
...
68
$ _
```

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
23
50
...
68
$ java dsa.LinearSearch ../data/tinyW.txt < ../data/tinyT.txt
```

Time Complexity

Program: `LinearSearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
23
50
...
68
$ java dsa.LinearSearch ../data/tinyW.txt < ../data/tinyT.txt
50
99
13
$ _
```

Time Complexity

Time Complexity

LinearSearch.java

```java
package dsa;

import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

public class LinearSearch {
    public static int indexOf(Object[] a, Object key) {
        for (int i = 0; i < a.length; i++) {
            if (a[i].equals(key)) {
                return i;
            }
        }
        return -1;
    }

    public static int indexOf(int[] a, int key) {
        for (int i = 0; i < a.length; i++) {
            if (a[i] == key) {
                return i;
            }
        }
        return -1;
    }

    public static int indexOf(double[] a, double key) {
        for (int i = 0; i < a.length; i++) {
            if (a[i] == key) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
```

Time Complexity

LinearSearch.java

```
        In inStream = new In(args[0]);
        int[] whiteList = inStream.readAllInts();
        while (!StdIn.isEmpty()) {
            int key = StdIn.readInt();
            if (indexOf(whiteList, key) == -1) {
                StdOut.println(key);
            }
        }
    }
}
```

Time Complexity

Time Complexity

dsa.BinarySearch	
`static int indexOf(Comparable[] a, Comparable key)`	returns the index of `key` in the sorted array `a`, or -1
`static int indexOf(int[] a, int key)`	returns the index of `key` in the sorted array `a`, or -1
`static int indexOf(double[] a, double key)`	returns the index of `key` in the sorted array `a`, or -1

Time Complexity

Time Complexity

Program: `BinarySearch.java`

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ _
```

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
```

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ _
```

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
```

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
23
50
...
68
$ _
```

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
23
50
...
68
$ java dsa.BinarySearch ../data/tinyW.txt < ../data/tinyT.txt
```

Time Complexity

Program: `BinarySearch.java`

- Command-line input: a filename (String)
- Standard input: a sequence of integers
- Standard output: the integers from standard input that are not in the file

```
>_ ~/workspace/dsaj/programs

$ cat ../data/tinyW.txt
84
48
...
29
$ cat ../data/tinyT.txt
23
50
...
68
$ java dsa.BinarySearch ../data/tinyW.txt < ../data/tinyT.txt
50
99
13
$ _
```

Time Complexity

Time Complexity

Successful binary search for the key 23 (returns 5)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Successful binary search for the key 23 (returns 5)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	7	14	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Successful binary search for the key 23 (returns 5)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	3	6	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Successful binary search for the key 23 (returns 5)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	5	6	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Successful binary search for the key 23 (returns 5)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	5	6	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	7	14	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	11	14	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	9	10	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	8	8	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9		8	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Unsuccessful binary search for the key 50 (returns -1)

			a[]														
lo	mid	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9		8	10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Time Complexity

Time Complexity

BinarySearch.java

```java
package dsa;

import java.util.Arrays;

import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

public class BinarySearch {
    public static int indexOf(Comparable[] a, Comparable key) {
        int lo = 0;
        int hi = a.length - 1;
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            int cmp = key.compareTo(a[mid]);
            if (cmp < 0) {
                hi = mid - 1;
            } else if (cmp > 0) {
                lo = mid + 1;
            } else {
                return mid;
            }
        }
        return -1;
    }

    public static int indexOf(int[] a, int key) {
        int lo = 0;
        int hi = a.length - 1;
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            if (key < a[mid]) {
                hi = mid - 1;
            } else if (key > a[mid]) {
                lo = mid + 1;
```

Time Complexity

BinarySearch.java

```
            } else {
                return mid;
            }
        }
        return -1;
    }

    public static int indexOf(double[] a, double key) {
        int lo = 0;
        int hi = a.length - 1;
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            if (key < a[mid]) {
                hi = mid - 1;
            } else if (key > a[mid]) {
                lo = mid + 1;
            } else {
                return mid;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        In inStream = new In(args[0]);
        int[] whiteList = inStream.readAllInts();
        Arrays.sort(whiteList);
        while (!StdIn.isEmpty()) {
            Integer key = StdIn.readInt();
            if (indexOf(whiteList, key) == -1) {
                StdOut.println(key);
            }
        }
    }
}
```

Time Complexity

Time Complexity

The running time of a single linear search on an array of size n is

$$T(n) = n$$

Time Complexity

The running time of a single linear search on an array of size n is

$$T(n) = n$$

The running time of a single binary search on an array of size n is

$$T(n) = n \log n \text{ (sorting cost) } + \log n \text{ (searching cost)}$$

Time Complexity

The running time of a single linear search on an array of size n is

$$T(n) = n$$

The running time of a single binary search on an array of size n is

$$T(n) = n \log n \text{ (sorting cost) } + \log n \text{ (searching cost)}$$

The running time of m linear searches on an array of size n is

$$T(n) = mn$$

Time Complexity

The running time of a single linear search on an array of size n is

$$T(n) = n$$

The running time of a single binary search on an array of size n is

$$T(n) = n \log n \text{ (sorting cost) } + \log n \text{ (searching cost)}$$

The running time of m linear searches on an array of size n is

$$T(n) = mn$$

The running time of m binary searches on an array of size n is

$$T(n) = n \log n \text{ (sorting cost) } + m \log n \text{ (searching cost)}$$

Time Complexity

Time Complexity

Program: `TripleSumFast.java`

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ _
```

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/1Kints.txt
```

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ /usr/bin/time -f "%es" java ThreeSumFast ../data/1Kints.txt
70
0.10s
$ _
```

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ /usr/bin/time -f "%es" java ThreeSumFast ../data/1Kints.txt
70
0.10s
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/2Kints.txt
```

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ /usr/bin/time -f "%es" java ThreeSumFast ../data/1Kints.txt
70
0.10s
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/2Kints.txt
528
0.17s
$ _
```

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ /usr/bin/time -f "%es" java ThreeSumFast ../data/1Kints.txt
70
0.10s
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/2Kints.txt
528
0.17s
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/4Kints.txt
```

Time Complexity

Program: `TripleSumFast.java`

- Command-line input: a filename (String)
- Standard output: the number of unordered triples (x, y, z) in the file such that $x + y + z = 0$

```
>_ ~/workspace/dsaj/programs

$ /usr/bin/time -f "%es" java ThreeSumFast ../data/1Kints.txt
70
0.10s
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/2Kints.txt
528
0.17s
$ /usr/bin/time -f "%es" java ThreeSumFast ../data/4Kints.txt
4039
0.47s
$ _
```

Time Complexity

Time Complexity

TripleSumFast.java

```java
import java.util.Arrays;

import dsa.BinarySearch;
import stdlib.In;
import stdlib.StdOut;

public class TripleSumFast {
    public static void main(String[] args) {
        In in = new In(args[0]);
        int[] a = in.readAllInts();
        StdOut.println(count(a));
    }

    private static int count(int[] a) {
        int n = a.length;
        Arrays.sort(a);
        int count = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                int k = BinarySearch.indexOf(a, -(a[i] + a[j]));
                if (k > j) {
                    count++;
                }
            }
        }
        return count;
    }
}
```

Time Complexity

Time Complexity

n	Triple Sum $T(n)$	Fast Triple Sum $T(n)$
1K	0.28s	0.1s
2K	1.8s	0.17s
4K	14.06s	0.47s
8K	111.83s	1.58s
16K	892.19s	6.09s

Space Complexity

Space Complexity

Memory requirements for primitive types

Type	Bytes
`boolean`	1
`byte`	1
`char`	2
`short`	2
`int`	4
`float`	4
`long`	8
`double`	8

Space Complexity

Memory requirements for primitive types

Type	Bytes
`boolean`	1
`byte`	1
`char`	2
`short`	2
`int`	4
`float`	4
`long`	8
`double`	8

To determine the memory usage of an object, we add the amount of memory used by each instance variable

Space Complexity

Memory requirements for primitive types

Type	Bytes
`boolean`	1
`byte`	1
`char`	2
`short`	2
`int`	4
`float`	4
`long`	8
`double`	8

To determine the memory usage of an object, we add the amount of memory used by each instance variable

For example, a `Counter` object uses 12 bytes: 8 bytes for `id` (a reference) and 4 bytes for `count`

Space Complexity

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

For example, an array of n `int` values uses $4n$ bytes

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

For example, an array of n `int` values uses $4n$ bytes

An array of objects is an array of references to objects, so we need to add the space for the references to the space required for the objects

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

For example, an array of n `int` values uses $4n$ bytes

An array of objects is an array of references to objects, so we need to add the space for the references to the space required for the objects

For example, an array of n `Counter` objects uses $8n$ bytes for references plus 12 bytes for each `Counter` object, for a grand total of $20n$ bytes

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

For example, an array of n `int` values uses $4n$ bytes

An array of objects is an array of references to objects, so we need to add the space for the references to the space required for the objects

For example, an array of n `Counter` objects uses $8n$ bytes for references plus 12 bytes for each `Counter` object, for a grand total of $20n$ bytes

A 2D array is an array of arrays (each array is an object)

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

For example, an array of n `int` values uses $4n$ bytes

An array of objects is an array of references to objects, so we need to add the space for the references to the space required for the objects

For example, an array of n `Counter` objects uses $8n$ bytes for references plus 12 bytes for each `Counter` object, for a grand total of $20n$ bytes

A 2D array is an array of arrays (each array is an object)

For example, an $m \times n$ array of `double` values uses $8m$ bytes for references plus 8 bytes for each of the mn `double` values, for a grand total of $8mn + 8m$ bytes

Space Complexity

The memory requirement for an array of primitive-type values is the memory needed to store the values

For example, an array of n `int` values uses $4n$ bytes

An array of objects is an array of references to objects, so we need to add the space for the references to the space required for the objects

For example, an array of n `Counter` objects uses $8n$ bytes for references plus 12 bytes for each `Counter` object, for a grand total of $20n$ bytes

A 2D array is an array of arrays (each array is an object)

For example, an $m \times n$ array of `double` values uses $8m$ bytes for references plus 8 bytes for each of the mn `double` values, for a grand total of $8mn + 8m$ bytes

A `String` of length n uses $2n$ bytes