Data Structures and Algorithms in Java

Algorithms and Data Structures: Analysis of Algorithms

Outline

1. Algorithmic Complexity
2. Time Complexity
3. Space Complexity

Algorithmic Complexity

Algorithmic Complexity

Given a program $\mathcal{P}$ to solve a problem of size n

- $\mathcal{P}$'s time complexity (aka running time), denoted $T(n)$, measures how long $\mathcal{P}$ will take to solve the problem
- $\mathcal{P}$'s space complexity, denoted $S(n)$, measures how much memory $\mathcal{P}$ will need to solve the problem

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

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

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

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

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

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

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

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

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

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

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 obtain $T(n)$ and $S(n)$ from some $g(n)$ by dropping the coefficient a

TripleSum.java

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

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

```
$ _
```

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

```
$ cat data/1Kints.txt
```

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

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

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

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

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

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

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

```
$ 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
```

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

```
$ 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
$ _
```

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

```
$ 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
```

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

```
$ 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
$ _
```

Algorithmic Complexity · Example (Triple Sum)

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;
    }
}
```

Given a program $\mathcal{P}$, collect execution time data for it, fit a function $f(n)$ to the data, and from it, derive $g(n)$ and $T(n)$

Given a program $\mathcal{P}$, collect execution time data for it, fit a function $f(n)$ to the data, and from it, derive $g(n)$ and $T(n)$

Example (Triple Sum)

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

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

$$\therefore g(n) = 0.2273121n^3$$

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

Given a program $\mathcal{P}$, compute a function $f(n)$ from

- The cost of executing each statement (property of the computer running $\mathcal{P}$)
- The frequency of execution of each statement (property of $\mathcal{P}$)

Given a program $\mathcal{P}$, compute a function $f(n)$ from

- The cost of executing each statement (property of the computer running $\mathcal{P}$)
- The frequency of execution of each statement (property of $\mathcal{P}$)

From $f(n)$, derive $g(n)$ and $T(n)$

Time Complexity · Mathematical Approach

Example (Triple Sum)

```java
1  private static int count(int[] a) {
2      int n = a.length;
3      int count = 0;
4      for (int i = 0; i < n; i++) {
5          for (int j = i + 1; j < n; j++) {
6              for (int k = j + 1; k < n; k++) {
7                  if (a[i] + a[j] + a[k] == 0) {
8                      count++;
9                  }
10             }
11         }
12     }
13     return count;
14 }
```

Line #	Time	Frequency	Total Time
2	t_1	1	t_1
3	t_2	1	t_2
4	t_3	n	$t_3 n$
5	t_4	${}^nC_2 = n^2/2 - n/2$	$t_4(n^2/2 - n/2)$
6	t_5	${}^nC_3 = n^3/6 - n^2/2 + n/3$	$t_5(n^3/6 - n^2/2 + n/3)$
7	t_6	${}^nC_3 = n^3/6 - n^2/2 + n/3$	$t_6(n^3/6 - n^2/2 + n/3)$
8	t_7	x (depends on input)	$t_7 x$
13	t_8	1	t_8

$$f(n) = \frac{t_5 + t_6}{6} n^3 + \frac{t_4 - t_5 - t_6}{2} n^2 + \frac{2t_5 + 2t_6 - 3t_4}{6} n + (t_1 + t_2 + t_7 x + t_8)$$

$$\therefore\ g(n) = \frac{t_5 + t_6}{6} n^3$$

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

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	enumerate all ordered pairs
cubic	n^3	triple loop	enumerate all ordered triples
exponential	2^n	exhaustive search	enumerate all subsets

Search for an item `item` in a collection of items `a`

Search for an item `item` in a collection of items `a`

`LinearSearch`	
`static int indexOf(Object[] a, Object item)`	returns the index of `item` in the array `a`, or -1
`static <T> int indexOf(T[] a, T item, Comparator<T> c)`	returns the index of `item` in the array `a`, or -1 (comparisons are made using the comparator `c`)
`static int indexOf(int[] a, int item)`	returns the index of `item` in the array `a`, or -1
`static int indexOf(double[] a, double item)`	returns the index of `item` in the array `a`, or -1

Search for an item `item` in a collection of items `a`

LinearSearch

`static int indexOf(Object[] a, Object item)`	returns the index of `item` in the array `a`, or -1
`static <T> int indexOf(T[] a, T item, Comparator<T> c)`	returns the index of `item` in the array `a`, or -1 (comparisons are made using the comparator `c`)
`static int indexOf(int[] a, int item)`	returns the index of `item` in the array `a`, or -1
`static int indexOf(double[] a, double item)`	returns the index of `item` in the array `a`, or -1

BinarySearch

`static <T extends Comparable<T>> int indexOf(T[] a, T item)`	returns the index of `item` in the sorted array `a`, or -1
`static <T> int indexOf(T[] a, T item, Comparator<T> c)`	returns the index of `item` in the array `a`, or -1 (comparisons are made using the comparator `c`)
`static int indexOf(int[] a, int item)`	returns the index of `item` in the sorted array `a`, or -1
`static int indexOf(double[] a, double item)`	returns the index of `item` in the sorted array `a`, or -1

TripleSumFast.java

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

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

```
$ _
```

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

```
$ /usr/bin/time -f "%es" java TripleSumFast data/1Kints.txt
```

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

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

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

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

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

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

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

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

Time Complexity · Example (Triple Sum Fast)

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

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

Time Complexity · Example (Triple Sum Fast)

Time Complexity · Example (Triple Sum Fast)

TripleSumFast.java

```java
import dsa.BinarySearch;
import dsa.Quick;
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;
        Quick.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 · Example (Triple Sum Fast)

Time Complexity · Example (Triple Sum Fast)

```java
1  private static int count(int[] a) {
2      int n = a.length;
3      Quick.sort(a);
4      int count = 0;
5      for (int i = 0; i < n; i++) {
6          for (int j = i + 1; j < n; j++) {
7              int k = BinarySearch.indexOf(a, -(a[i] + a[j]));
8              if (k > j) {
9                  count++;
10             }
11         }
12     }
13     return count;
14 }
```

Line #	Time	Frequency	Total Time
2	t_1	1	t_1
3	$t_2 n \log n$	1	$t_2 n \log n$
4	t_3	1	t_3
5	t_4	n	$t_4 n$
6	t_5	${}^nC_2 = n^2/2 - n/2$	$t_5(n^2/2 - n/2)$
7	$t_6 \log n$	${}^nC_2 = n^2/2 - n/2$	$t_6(n^2/2 - n/2) \log n$
8	t_7	${}^nC_2 = n^2/2 - n/2$	$t_7(n^2/2 - n/2)$
9	t_8	x (depends on input)	$t_8 x$
13	t_9	1	t_9

$$f(n) = \frac{t_6}{2} n^2 \log n + \frac{t_5 + t_6 + t_7}{2} n^2 + \frac{2t_2 - t_6}{2} n \log n + \frac{2t_4 - t_5 - t_7}{2} n + (t_1 + t_3 + t_8 x + t_9)$$

$$\therefore\ g(n) = \frac{t_6}{2} n^2 \log n$$

$$\therefore\ T(n) = n^2 \log n$$

Time Complexity · Triple Sum Versus Triple Sum Fast

n	T(n) (Triple Sum)	T(n) (Triple Sum Fast)
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

Time Complexity · Linear Search

Time Complexity · Linear Search

</> LinearSearch.java

```java
package dsa;

import java.util.Comparator;
import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

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

    public static <T> int indexOf(T[] a, T item, Comparator<T> c) {
        for (int i = 0; i < a.length; i++) {
            if (c.compare(a[i], item) == 0) {
                return i;
            }
        }
        return -1;
    }

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

```

Time Complexity · Linear Search

LinearSearch.java


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

    public static void main(String[] args) {
        // Unit tests the library
    }
}
```

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

`item = 23`

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

item = 23

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

item = 23

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

item = 23

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

item = 23

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

`item = 23`

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

item = 23

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

`item = 50`

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

item = 50

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

item = 50

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

item = 50

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

`item = 50`

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

`item = 50`

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

item = 50

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

item = 50

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

`item = 50`

Time Complexity · Binary Search

BinarySearch.java 1/3

```java
package dsa;

import java.util.Comparator;
import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

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

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

BinarySearch.java 2/3

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

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

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

BinarySearch.java 3/3

```
        }
        return -1;
    }

    public static void main(String[] args) {
        // Unit tests the library
    }
}
```

Single linear search on an array of size n

$$T(n) = n$$

Single linear search on an array of size n

$$T(n) = n$$

Single binary search on an array of size n

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

Single linear search on an array of size n

$$T(n) = n$$

Single binary search on an array of size n

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

m linear searches on an array of size n

$$T(n) = mn$$

Single linear search on an array of size n

$$T(n) = n$$

Single binary search on an array of size n

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

m linear searches on an array of size n

$$T(n) = mn$$

m binary searches on an array of size n

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

Space Complexity

Space Complexity

Memory used by primitive type values

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

Space Complexity

Memory used by primitive type values

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

Reference (pointer) to an object uses 8 bytes

Space Complexity

Memory used by primitive type values

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

Reference (pointer) to an object uses 8 bytes

For an object, add the memory used by each instance variable

Space Complexity

Memory used by primitive type values

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

Reference (pointer) to an object uses 8 bytes

For an object, add the memory used by each instance variable

Example: a `Counter` object uses 12 bytes — 8 bytes for `id` (reference to a `String` object) plus 4 bytes for `count` (an `int`)

Space Complexity

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Example: an array of n `int` values uses $4n$ bytes

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Example: an array of n `int` values uses $4n$ bytes

For an array of objects, add the memory used by references to the objects to the memory used by the objects

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Example: an array of n `int` values uses $4n$ bytes

For an array of objects, add the memory used by references to the objects to the memory used by the objects

Example: an array of n `Counter` objects uses $20n$ bytes — $8n$ bytes for references plus $12n$ bytes for the objects

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Example: an array of n `int` values uses $4n$ bytes

For an array of objects, add the memory used by references to the objects to the memory used by the objects

Example: an array of n `Counter` objects uses $20n$ bytes — $8n$ bytes for references plus $12n$ bytes for the objects

A 2D array is an array of arrays

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Example: an array of n `int` values uses $4n$ bytes

For an array of objects, add the memory used by references to the objects to the memory used by the objects

Example: an array of n `Counter` objects uses $20n$ bytes — $8n$ bytes for references plus $12n$ bytes for the objects

A 2D array is an array of arrays

Example: an $m \times n$ array of `double` values uses $8mn + 8m$ bytes — $8m$ bytes for references to m arrays plus $8mn$ bytes for the mn `double` values

Space Complexity

Memory used by an array of primitive-type values is the memory needed to store the values

Example: an array of n `int` values uses $4n$ bytes

For an array of objects, add the memory used by references to the objects to the memory used by the objects

Example: an array of n `Counter` objects uses $20n$ bytes — $8n$ bytes for references plus $12n$ bytes for the objects

A 2D array is an array of arrays

Example: an $m \times n$ array of `double` values uses $8mn + 8m$ bytes — $8m$ bytes for references to m arrays plus $8mn$ bytes for the mn `double` values

A string of length n uses $2n$ bytes

Space Complexity

Space Complexity

Example (Triple Sum)

```
1  private static int count(int[] a) {
2      int n = a.length;
3      int count = 0;
4      for (int i = 0; i < n; i++) {
5          for (int j = i + 1; j < n; j++) {
6              for (int k = j + 1; k < n; k++) {
7                  if (a[i] + a[j] + a[k] == 0) {
8                      count++;
9                  }
10             }
11         }
12     }
13     return count;
14 }
```

Line #	Type	Bytes
1	`int[]`	$4n$
2	`int`	4
3	`int`	4
4	`int`	4
5	`int`	4
6	`int`	4

$$f(n) = 4n + 20$$

$$\therefore g(n) = 4n$$

$$\therefore S(n) = n$$