Building Java Programs

Chapter 13
binary search and complexity

reading: 13.1-13.2
10/07/2018

Dogs spotted this weekend

Chunk on the street

Dazzling smile

Waiting patiently

Little weenie, too far away

Now Playing: nanatee commune
Road Map

CS Concepts
- Client/Implementer
- Efficiency
- Recursion
- Regular Expressions
- Grammars
- Sorting
- Backtracking
- Hashing
- Huffman Compression

Java Language
- Exceptions
- Interfaces
- References
- Comparable
- Generics
- Inheritance/Polymorphism
- Abstract Classes

Data Structures
- Lists
- Stacks
- Queues
- Sets
- Maps
- Priority Queues

Java Collections
- Arrays
- ArrayList
- LinkedList
- Stack
- TreeSet / TreeMap
- HashSet / HashMap
- PriorityQueue
• Let’s write a method to calculate the sum from 1 to some n

```java
public static int sum1(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

• Gauss also has a way of solving this

```java
public static int sum2(int n) {
    return n * (n + 1) / 2;
}
```

• Which one is more efficient?
Runtime Efficiency (13.2)

- **efficiency**: measure of computing resources used by code.
  - can be relative to speed (time), memory (space), etc.
  - most commonly refers to run time
- We want to be able to compare different algorithms to see which is more efficient
Efficiency Try 1

- Let’s time the methods!

<table>
<thead>
<tr>
<th>n</th>
<th>sum1 took</th>
<th>sum2 took</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 1</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 5</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 10</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 100</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 1,000</td>
<td>1ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 10,000,000</td>
<td>18ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 100,000,000</td>
<td>127ms</td>
<td>0ms</td>
</tr>
<tr>
<td>n = 2,147,483,647</td>
<td>1570ms</td>
<td>0ms</td>
</tr>
</tbody>
</table>

- Downsides
  - Different computers give different run times
  - The same computer gives different results!!! D:<
Efficiency – Try 2

- Count number of “simple steps” our algorithm takes to run
- Assume the following:
  - Any single Java statement takes same amount of time to run.
    - `int x = 5;`
    - `boolean b = (5 + 1 * 2) < 15 + 3;`
    - `System.out.println("Hello");`
  - A loop's runtime, if the loop repeats N times, is N times the runtime of the statements in its body.
  - A method call's runtime is measured by the total runtime of the statements inside the method's body.
public static void method1(int N) {
    statement1;
    statement2;
    statement3;

    for (int i = 1; i <= N; i++) {
        statement4;
    }

    for (int i = 1; i <= N; i++) {
        statement5;
        statement6;
        statement7;
    }
}

Efficiency examples

- 3
- N
- 4N + 3
- 3N
public static void method2(int N) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            statement1;
        }
    }
}

for (int i = 1; i <= N; i++) {
    statement2;
    statement3;
    statement4;
    statement5;
}

- How many statements will execute if N = 10? If N = 1000? 
  \( \sim 140 \quad \sim 10,000 \)
Sum this up for me

- Let’s write a method to calculate the sum from 1 to some n
  
  ```java
  public static int sum1(int n) {
      int sum = 0;
      for (int i = 1; i <= n; i++) {
          sum += i;
      }
      return sum;
  }
  ```

- Gauss also has a way of solving this
  
  ```java
  public static int sum2(int n) {
      return n * (n + 1) / 2;
  }
  ```

- Which one is more efficient?
Visualizing Difference

Comparing sum1 and sum2

Number of steps vs. n
Algorithm growth rates (13.2)

- We measure runtime in proportion to the input data size, N.
  - **growth rate**: Change in runtime as N changes.

- Say an algorithm runs \(0.4N^3 + 25N^2 + 8N + 17\) statements.
  - Consider the runtime when N is extremely large.
  - We ignore constants like 25 because they are tiny next to N.
  - The highest-order term (\(N^3\)) dominates the overall runtime.

- We say that this algorithm runs "on the order of" \(N^3\).
- or \(O(N^3)\) for short ("Big-Oh of N cubed")

  If you double input, will take about 8x as long
Suppose our list had the contents

```java
public void method(int n) {
    int value = 0;
    for (int i = 0; i < 7; i++) {
        for (int j = 0; j < n; j++) {
            value += j;
        }
    }
    return value + n / 2;
}
```

What is the Big-O efficiency for this function?

- O(1)
- O(n) \(\checkmark\)
- O(7n) \(\times\)
- O(7n + 4); \(\times\)
- O(n^2)
- O(n^3)
### Complexity classes

- **complexity class**: A category of algorithm efficiency based on the algorithm's relationship to the input size $N$.

<table>
<thead>
<tr>
<th>Class</th>
<th>Big-Oh</th>
<th>If you double $N$, ...</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>$O(1)$</td>
<td>unchanged</td>
<td>10ms</td>
</tr>
<tr>
<td>logarithmic</td>
<td>$O(\log_2 N)$</td>
<td>increases slightly</td>
<td>175ms</td>
</tr>
<tr>
<td>linear</td>
<td>$O(N)$</td>
<td>doubles</td>
<td>3.2 sec</td>
</tr>
<tr>
<td>log-linear</td>
<td>$O(N \log_2 N)$</td>
<td>slightly more than doubles</td>
<td>6 sec</td>
</tr>
<tr>
<td>quadratic</td>
<td>$O(N^2)$</td>
<td>quadruples</td>
<td>1 min 42 sec</td>
</tr>
<tr>
<td>cubic</td>
<td>$O(N^3)$</td>
<td>multiplies by 8</td>
<td>55 min</td>
</tr>
<tr>
<td>exponential</td>
<td>$O(2^N)$</td>
<td>multiplies drastically</td>
<td>$5 \times 10^{61}$ years</td>
</tr>
</tbody>
</table>
Complexity classes

Sequential search

- **sequential search**: Locates a target value in an array/list by examining each element from start to finish. Used in `indexOf`.
  - How many elements will it need to examine?
  - Example: Searching the array below for the value 42:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

i
Sequential search

- What is its complexity class?

```java
public int indexOf(int value) {
    for (int i = 0; i < size; i++) {
        if (elementData[i] == value) {
            return i;
        }
    }
    return -1;  // not found
}
```

- On average, "only" N/2 elements are visited
  - 1/2 is a constant that can be ignored

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>
Binary search (13.1)

- **binary search**: Locates a target value in a *sorted* array or list by successively eliminating half of the array from consideration.

  - How many elements will it need to examine?
  - Example: Searching the array below for the value **42**:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

  - min
  - mid
  - max
Binary search

- **binary search** successively eliminates half of the elements.

  - **Algorithm:** Examine the middle element of the array.
    - If it is too big, eliminate the right half of the array and repeat.
    - If it is too small, eliminate the left half of the array and repeat.
    - Else it is the value we're searching for, so stop.

- Which indexes does the algorithm examine to find value **42**?
- What is the runtime complexity class of binary search?

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

min | mid | max
Binary search runtime

- For an array of size N, it eliminates \( \frac{1}{2} \) until 1 element remains.
  - \( N, \frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \ldots, 4, 2, 1 \)
  - How many divisions does it take?

- Think of it from the other direction:
  - How many times do I have to multiply by 2 to reach \( N \)?
    - \( 1, 2, 4, 8, \ldots, \frac{N}{4}, \frac{N}{2}, N \)
  - Call this number of multiplications "\( x \)".

\[
2^x = N \\
x = \log_2 N
\]

- Binary search is in the **logarithmic** complexity class.
Complexity classes

**Collection efficiency**

- Efficiency of our Java's `ArrayList` and `LinkedList` methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>ArrayList</th>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>O(1)*</td>
<td>O(1)</td>
</tr>
<tr>
<td><code>add(index, value)</code></td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td><code>indexOf</code></td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td><code>get</code></td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td><code>remove</code></td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td><code>set</code></td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td><code>size</code></td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

* Will cover Wed

* Most of the time!
Throw Back: Unique words

- Recall two weeks ago when we counted the number of unique words in a file. Our first attempt

```java
public static int uniqueWords(Scanner input) {
    List<String> words = new LinkedList<String>();
    while (input.hasNext()) {
        String word = input.next();
        if (!words.contains(word)) {
            words.add(word);
        }
    }
    return words.size();
}
```
Throw Back: Unique words

- Recall two weeks ago when we counted the number of unique words in a file. Our second attempt
- We saw briefly that operations on HashSet are $O(1)$

```java
public static int uniqueWords(Scanner input) {
    Set<String> words = new HashSet<String>();
    while (input.hasNext()) {
        String word = input.next();
        words.add(word);
    }
    return words.size();
}
```