Test 4

The fourth test will be 75 minutes, will consist of two parts and will take place this week.

The programming part will be about Chapter 2-6, excluding Section 2.6, 4.5, and 6.8.8. You will be asked to implement one class. We will already provide you with a skeleton which includes the javadoc. This part will be worth 70% of the marks. If your code does not compile, you get a 50% penalty (that is, your score for the programming part will be divided by 2 if your code does not compile).

The "written" part will be about Chapter 2-6, excluding Section 2.6, 4.5, and 6.8.8. This part will consist of three questions (two multiple choice/short answer questions and one longer answer question). This part will be worth the remaining 30% of the marks.

During the test, you will have access to the textbook. You may bring a blank piece of paper to the test.

Chapter 7: Recursion

EECS 1030

moodle.yorku.ca

```
/**
 * Returns 3 raised to the given power.
 *
 * @param n a number.
 * @pre. n >= 0
 */
public static BigInteger pow3(int n)
```

Question

Why is the return type of `pow3` the class `BigInteger`?

BigInteger

Question

Why is the return type of `pow3` the class `BigInteger`?

Answer

Because $3^{\texttt{Integer.MAX\_VALUE}}$ cannot be represented by `int` or `long`.

Powers of 3: Iteration

```
BigInteger power = BigInteger.ONE;
for (int i = 0; i < n; i++)
{
   power = power.multiply(THREE);
}
```

Powers of 3: Recursion – version 1

```
BigInteger power;
if (n == 0)
{
   power = BigInteger.ONE;
}
else
{
   power = pow3(n - 1).multiply(THREE);
}
```

Powers of 3: Recursion – version 2

```
BigInteger power;
if (n == 0)
{
   power = BigInteger.ONE;
}
else if (n % 2 == 1)
{
   power = pow3(n - 1).multiply(THREE);
}
else
{
   power = pow3(n / 2).multiply(pow3(n / 2));
}
```

Powers of 3: Recursion – version 3

```
BigInteger power;
if (n == 0)
{
   power = BigInteger.ONE;
}
else if (n % 2 == 1)
{
   power = pow3(n - 1).multiply(THREE);
}
else
{
   BigInteger temp = power(n / 2);
   power = temp.multiply(temp);
}
```

Efficiency

Question

How do we determine which implementation is the most efficient one?

Efficiency

Question

How do we determine which implementation is the most efficient one?

Answer

Run the implementations and measure their execution times.

Errors

Question

Why may the following approach give rise relatively large errors in the measurement of the execution time?

```
for n = 0, 1, 2, ...
   measure the execution time of pow3(n)
```

Errors

Question

Why may the following approach give rise relatively large errors in the measurement of the execution time?

```
for n = 0, 1, 2, ...
   measure the execution time of pow3(n)
```

Answer

Because the execution time of `pow3(n)` is small (for example, `pow3(16)` takes less than 500 nanoseconds), other processes that run on the computer (even if they take a small amount of time) may have a relatively big impact on the error of the measurement.

Question

How can we reduce the error?

Error

Question

How can we reduce the error?

Answer

Execute `pow3` not once but, say, 1,000,000 times.

Confidence

Question

```
for n = 0, 1, 2, ...
   measure the execution time of
      for (int i = 0; i < 1000000; i++)
         pow3(n)
```

How can we improve our confidence in the measured execution time?

Confidence

Question

```
for n = 0, 1, 2, ...
   measure the execution time of
      for (int i = 0; i < 1000000; i++)
         pow3(n)
```

How can we improve our confidence in the measured execution time?

Answer

Run the experiment 100 times (compute the average and standard deviation).

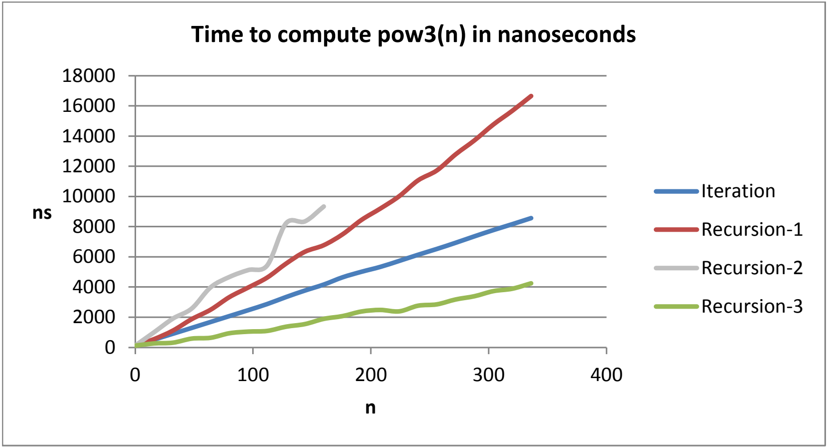
Experimental set up

When writing and running the code to measure the execution time

- ensure that `pow3` is executed,
- invoke the garbage collector regularly,
- run the code in server mode,
- use the 64-bit version of the JVM,
- ignore the first couple of runs to eliminate the effects of JIT compilation,
- ...

The details, although interesting, are outside the scope of this course.

Comparison


Time to compute pow3(n) in nanoseconds
ns
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
100
200
300
400
n
Iteration
Recursion-1
Recursion-2
Recursion-3

The experimental approach

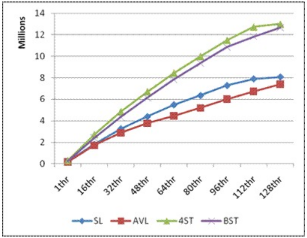
Advantages

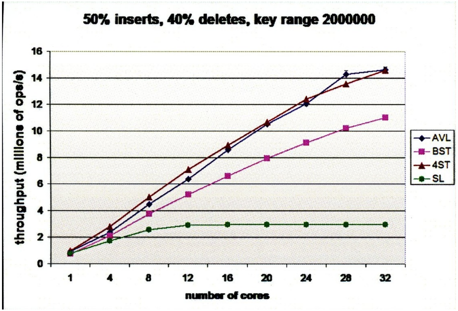
- The graph gives us an answer to our question "which implementation is the most efficient one?"
- It is relatively simple.

Disadvantages

- It only answers the question for the computer on which the code is run.
- It is time consuming (running the experiment for $n = 0, 16, 32, \ldots, 336$ took more than 24 hours).

Different computers, different graphs





Research by undergraduate student Trevor Brown and graduate student Joanna Helga.

Alternative

Can we analyze the running time, independent of any computer, that takes considerably less time?

Alternative

Can we analyze the running time, independent of any computer, that takes considerably less time?

Answer

Yes.

Running time analysis

Basic idea

Given a value $\texttt{n} \geq 0$, estimate the number of elementary instructions that are executed during the invocation of `pow3(n)`.

Running time analysis

Basic idea

Given a value `n` ≥ 0, estimate the number of elementary instructions that are executed during the invocation of `pow3(n)`.

Question

What is an elementary instruction?

Running time analysis

Basic idea

Given a value $n \geq 0$, estimate the number of elementary instructions that are executed during the invocation of `pow3(n)`.

Question

What is an elementary instruction?

Answer

Since we estimate, a precise definition of an elementary instruction is not needed. For example, each bytecode instruction or each machine instruction can be considered elementary.

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
int x = 1;
```

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
int x = 1;
```

Answer

1 or 2 or even 10 are good estimates.

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
x = x + 1;
```

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
x = x + 1;
```

Answer

Something like 3.

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
for (int i = 0; i < n; i++) { }
```

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
for (int i = 0; i < n; i++) { }
```

Answer

Something like $6n + 5$, but not 10

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
for (int i = 0; i < n; i++)
{
   for (int j = 0; j < n; j++) {}
}
```

Estimate

Question

Estimate the number of elementary instructions of the following code snippet.

```
for (int i = 0; i < n; i++)
{
   for (int j = 0; j < n; j++) {}
}
```

Answer

Something like $6n^2 + 11n + 5$

Question

Estimate the number of elementary instructions[a] of the following code snippet.

```
BigInteger power = BigInteger.ONE;
for (int i = 0; i < n; i++)
{
   power = power.multiply(THREE);
}
```

[a]To simplify matters a little, we assume that the methods of the BigInteger class take a constant number of elementary instructions, no matter how big the numbers are.

Running time analysis

Strategy

- For each line of code, estimate its number of elementary instructions.
- For each line of code, determine how often it is executed.
- Determine the total number of elementary instructions.

Question

```
BigInteger power = BigInteger.ONE;
for (int i = 0; i < n; i++)
  power = power.multiply(THREE);
```

For each line of code, estimate its number of elementary instructions.

Estimate

Question

```
1. BigInteger power = BigInteger.ONE;
2. for (int i = 0; i < n; i++)
3.    power = power.multiply(THREE);
```

For each line of code, estimate its number of elementary instructions.

Answer

line 1: 2 instructions
line 2: 8 instructions
line 3: 4 instructions

Estimate

Question

```
BigInteger power = BigInteger.ONE;
for (int i = 0; i < n; i++)
    power = power.multiply(THREE);
```

For each line of code, determine how often it is executed.

Estimate

Question

```
1. BigInteger power = BigInteger.ONE;
2. for (int i = 0; i < n; i++)
3.     power = power.multiply(THREE);
```

For each line of code, determine how often it is executed.

Answer

line 1: once
line 2: n times
line 3: n times

Estimate

Question

```
1. BigInteger power = BigInteger.ONE;
2. for (int i = 0; i < n; i++)
3.     power = power.multiply(THREE);
```

Determine the total number of elementary instructions.

Estimate

Question

```java
BigInteger power = BigInteger.ONE;
for (int i = 0; i < n; i++)
    power = power.multiply(THREE);
```

Determine the total number of elementary instructions.

Answer

$2 + 8n + 4n = 12n + 2$

Functions

Each estimate can be viewed as a function from natural numbers (nonnegative integers) to natural numbers.

$$I \quad : \quad \mathbb{N} \quad \rightarrow \quad \mathbb{N}$$

Functions

Each estimate can be viewed as a function from natural numbers (nonnegative integers) to natural numbers.

$$I \quad : \quad \mathbb{N} \quad \longrightarrow \quad \mathbb{N}$$

$\uparrow$

name of function

Functions

Each estimate can be viewed as a function from natural numbers (nonnegative integers) to natural numbers.

$$I \quad : \quad \mathbb{N} \quad \rightarrow \quad \mathbb{N}$$

$$\uparrow$$

domain of function

Functions

Each estimate can be viewed as a function from natural numbers (nonnegative integers) to natural numbers.

$$I \quad : \quad \mathbb{N} \quad \rightarrow \quad \mathbb{N}$$

$$\uparrow$$

range of function

Functions

Each estimate can be viewed as a function from natural numbers (nonnegative integers) to natural numbers.

$$I \;:\; \mathbb{N} \;\rightarrow\; \mathbb{N}$$

I(n) : estimate of the number of elementary instructions executed in pow3(n)

Functions

Each estimate can be viewed as a function from natural numbers (nonnegative integers) to natural numbers.

$$I \quad : \quad \mathbb{N} \quad \rightarrow \quad \mathbb{N}$$

$$I(n) = 12n + 2$$

Estimate

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else
    power = pow3(n - 1).multiply(THREE);
```

For each line of code, estimate its number of elementary instructions.

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else
5.     power = pow3(n - 1).multiply(THREE);
```

For each line of code, estimate its number of elementary instructions.

Answer

line 1: 1 instruction
line 2: 3 instructions
line 3: 2 instructions
line 5: $R_1(n-1)+2$ instructions

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else
5.     power = pow3(n - 1).multiply(THREE);
```

For each line of code, determine how often it is executed.

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else
5.     power = pow3(n - 1).multiply(THREE);
```

For each line of code, determine how often it is executed.

Answer

line 1: once
line 2: once
line 3: once or not at all
line 5: once or not at all

Estimate

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else
    power = pow3(n - 1).multiply(THREE);
```

Determine the total number of elementary instructions.

Estimate

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else
    power = pow3(n - 1).multiply(THREE);
```

Determine the total number of elementary instructions.

Answer

$R_1(0) = 6$
$R_1(n) = R_1(n-1) + 6$

The above is known as a recurrence relation.

Solving recurrence relation

Claim

For all $n \geq 0$, $R_1(n) = 6n + 6$.

Solving recurrence relation

Claim

For all $n \geq 0$, $R_1(n) = 6n + 6$.

Proof

We prove the claim by induction on n.

- Base case: n $= 0$. By definition, $R_1(0) = 6$. Also, $R_1(0) = 6 \times 0 + 6 = 6$.
- Inductive case: $n > 0$. Assume that $R_1(n-1) = 6(n-1) + 6$. (This is the induction hypothesis.) By definition, $R_1(n) = R_1(n-1) + 6$. By the induction hypothesis, $R_1(n) = 6(n-1) + 6 + 6 = 6n + 6$.

Estimate

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    power = pow3(n / 2).multiply(pow3(n / 2));
```

For each line of code, estimate its number of elementary instructions.

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else if (n % 2 == 1)
5.     power = pow3(n - 1).multiply(THREE);
6. else
7.     power = pow3(n / 2).multiply(pow3(n / 2));
```

For each line of code, estimate its number of elementary instructions.

Answer

line 1: 1 instruction
line 2: 3 instructions
line 3: 2 instructions
line 4: 5 instructions
line 5: $R_2(n-1)+2$ instructions
line 7: $2R_2(n/2)+2$ instructions

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    power = pow3(n / 2).multiply(pow3(n / 2));
```

For each line of code, determine how often it is executed.

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else if (n % 2 == 1)
5.     power = pow3(n - 1).multiply(THREE);
6. else
7.     power = pow3(n / 2).multiply(pow3(n / 2));
```

For each line of code, determine how often it is executed.

Answer

line 1: once
line 2: once
line 3: once or not at all
line 4: once or not at all
line 5: once or not at all
line 7: once or not at all

Powers of 3

Recurrence relation

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    power = pow3(n / 2).multiply(pow3(n / 2));
```

Determine the recurrence relation.

Powers of 3

Recurrence relation

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    power = pow3(n / 2).multiply(pow3(n / 2));
```

Determine the recurrence relation.

Answer

$R_2(0) = 6$

$$R_2(n) = \begin{cases} R_2(n-1) + 11 & \text{if } n \text{ is odd} \\ 2R_2(n/2) + 11 & \text{if } n \text{ is even} \end{cases}$$

Estimate

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    BigInteger temp = power(n / 2);
    power = temp.multiply(temp);
```

For each line of code, estimate its number of elementary instructions.

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else if (n % 2 == 1)
5.     power = pow3(n - 1).multiply(THREE);
6. else
7.     BigInteger temp = power(n / 2);
8.     power = temp.multiply(temp);
```

For each line of code, estimate its number of elementary instructions.

Answer

line 1: 1 instruction
line 2: 3 instructions
line 3: 2 instructions
line 4: 5 instructions
line 5: $R_3(n-1)+2$ instructions
line 7: $R_3(n/2)+4$ instructions
line 8: 4 instructions

Estimate

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    BigInteger temp = power(n / 2);
    power = temp.multiply(temp);
```

For each line of code, determine how often it is executed.

Estimate

Question

```
1. BigInteger power;
2. if (n == 0)
3.     power = BigInteger.ONE;
4. else if (n % 2 == 1)
5.     power = pow3(n - 1).multiply(THREE);
6. else
7.     BigInteger temp = power(n / 2);
8.     power = temp.multiply(temp);
```

For each line of code, determine how often it is executed.

Answer

line 1: once
line 2: once
line 3: once or not at all
line 4: once or not at all
line 5: once or not at all
line 7: once or not at all
line 8: once or not at all

Recurrence relation

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    BigInteger temp = power(n / 2);
    power = temp.multiply(temp);
```

Determine the total number of elementary instructions.

Recurrence relation

Question

```
BigInteger power;
if (n == 0)
    power = BigInteger.ONE;
else if (n % 2 == 1)
    power = pow3(n - 1).multiply(THREE);
else
    BigInteger temp = power(n / 2);
    power = temp.multiply(temp);
```

Determine the total number of elementary instructions.

Answer

$R_3(0) = 6$

$$R_3(n) = \begin{cases} R_2(n-1) + 11 & \text{if } n \text{ is odd} \\ R_2(n/2) + 11 & \text{if } n \text{ is even} \end{cases}$$