Week 7 worksheet: Complexity Analysis and Generics

Total points: 10

Out: 2024 October 14 (Monday)

Due: 2024 October 16 (Thursday end of day [2359 CDT according to D2L])

## What to submit?

Upload only one Word or PDF file to the designated D2L folder.

## Running time assumptions

* Assume that basic arithmetic operations (+, -, \*, /, %) run in constant time.
* Assume that the System.out.println( … ) statement runs in constant time.

## Generic Node <E> assumptions

* Assume that generic class E has an equals method.

# Exercise 1: Detailed Runtime Complexity Analysis

[2 pts] In the following code, assume that arithmetic and the println statement run in constant time.

**int count = 10;**

**for ( int j = 0; j < 5; j++ ) {**

**System.out.println( j \* count );**

**}**

For each line of this code, how many times is the line executed, and how many operations are executed whenever the line is executed? *[0.7]*

|  |  |  |
| --- | --- | --- |
| **Line of code** | **Ops per execution** | **Number of times executed** |
| **int count = 10;** | 2 | 1 |
| **for ( int j = 0; j < 5; j++ )** | 3 | 6 |
| **System.out.println( j \* count );** | 2 | 5 |
| **TOTAL NUMBER OF STEPS** | | 30 – 2 (int j = 0 only happens once) 28 |

Now consider the following code snippet:

**int count = 10;**

**for ( int j = 0; j < n; j++ ) {**

**System.out.println( j \* count );**

**}**

For each line of this code, how many times is the line executed, and how many operations are executed whenever the line is executed? *[0.7]*

|  |  |  |
| --- | --- | --- |
| **Line of code** | **Ops per execution** | **Number of times executed** |
| **int count = 10;** | 1 | 1 |
| **for ( int j = 0; j < n; j++ )** | 3 | n + 1 |
| **System.out.println( j \* count );** | 2 | n + 1 |
| **TOTAL NUMBER OF STEPS** | | 1 + (5 \* (n + 1)) |

What is the big-O running time of this code snippet? *[0.6]* O(N)

# Exercise 2: Another runtime analysis

[3 pts] What are the big-O running times of the following methods as a function of the input parameter(s)?

public void method1( int n ) {

for ( int j = 1; j <= n; j++ ) {

for ( int k = 1; k < n; k++ ) {

System.out.println ( j \* k ) ;

}

}

}

Big-O running time of method1 O(N^2)

public void method2( int m, int n ) {

for ( int j = 1; j <= m; j++ ) {

for ( int k = 1; k < n; k++ ) {

System.out.println ( j \* k ) ;

}

}

}

Big-O running time of method2 O(N^2)

public void method3( int n ) {

for ( int j = 1; j <= n; j \*= 2 ) {

System.out.println ( j \* k ) ;

}

}

Big-O running time of method3 O(n log n)

# Exercise 3: Implement removeFirst

[5 pts] Suppose that you have a linked list of Nodes of generic objects:

public class Node<E> {

E data;

Node<E> link;

public E getData() { return data; }

public Node<E> getLink() { return link; }

public void setData( E data ) { this.data = data; }

public void setLink( Node<E> link ) { this.link = link; }

}

Suppose further you have a linked list with a head Node<E> named “head”.

Suppose further that class E has an equals method.

Write a method removeFirst that removes the first node of the linked list that contains a value “target” of type E. Of course, your method must then reconnect the list. If there is no such element in the list, return null. *[4 pts]* The signature of this method is public E removeFirst( E target, Node<E> head )

A screen shot of a computer program

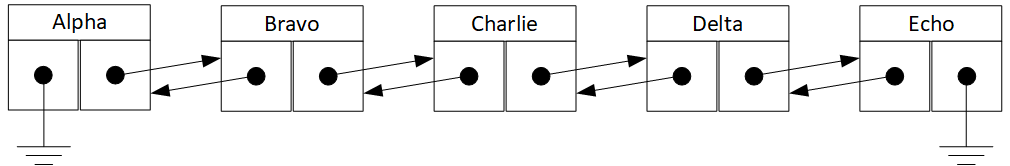
Description automatically generated

If the list has n nodes, what is the big-O running time of this method in terms of n? *[1 pt]* O(N)

# Exercise 4: Doubly linked list removeFirst

[2 pts] Suppose that you have a doubly linked list of Nodes of generic objects. Each Node<E> has two references, one to the previous element in the list and one to the next element. It has two named nodes, “head” and “tail”. “head” has a null “prev” reference, and “tail” has a null “next” reference.

A sample such list with Strings would look like this:



public class Node<E> {

E data;

Node<E> prev;

Node<E> next;

public E getData() { return data; }

public Node<E> getPrev() { return prev; }

public Node<E> getNext() { return next; }

public void setData( E data ) { this.data = data; }

public void setPrev ( Node<E> prev ) { this.prev = prev; }

public void setNext ( Node<E> next ) { this.prev = next; }

}

Write a method removeLast that removes the last node of the linked list that contains a value “target” of type E, and then reconnects the list. If there is no such element in the list, return null. The signature of this method is public E removeLast( E target, Node<E> head, Node<E> tail )

A screen shot of a computer program

Description automatically generated

# Exercise 4: reflect on your learning this week

1. What is the most important thing you learned in this week’s lecture? Write 2-3 sentences to explain your answer.

Generic classes can be used to handle multiple different kinds of objects however, they all must be the same type per use.

1. What is the muddiest (most-unclear) point(s) in this week’s lecture? Explain why?

Doubly linked lists, kind of as expected. I just don’t have enough practice working through it yet.