Namra Joshi

Professor Mortezaie

CS 146-04

September 15, 2019

Homework 1

1) Using the algorithm convertToPostfix given in the lecture, convert each of the following infix expressions to postfix expressions. Use the stack technique but check your work using pencil and paper technique as well as stack technique.

a. $a * b / (c - d)$
b. $(a - b * c) / (d * e * f + g)$
c. $a / b * (c + (d - e))$

Answer:

a) a*b/(c-d)

- First, " ***a*** " is added to the output string, Next operator " ***** " is pushed into the stack. " ***b*** " is now added to the output string. The next operator " / " is of equal priority with the top of the stack that is " ***** "**.** So the top is added to the output string. Output string now will be **ab***

- “ / ” is now pushed into the stack. The opening parenthesis is pushed over “ / ” into the stack. The next character c is added to the output string. When “ **-** ” is encountered, and as the top of the stack is “ **(** ” , the operator is pushed into the stack. Stack contents now are “ **/(-** ”
- Next character “**d**” is added to the output string, which becomes **ab*cd**. When **)** is encountered, all operators in the stack are popped and appended to the output string.
- Thus the output string will be: ***a b * c d - /***

b) (a-b * c) / (d*e* f+g)

- “ (” is pushed into the stack. **a** is added to the output string. “ **-** ” is pushed into the stack. **b** is added to the output which now becomes, **ab**. When “ ***** ” is encountered, as “ **-** ” of lower priority is as the top of the stack, “ ***** ” is pushed into the stack. The stack contains **(-*** and **c** is added to the output which becomes, **abc**.
- When “ **)** ” is encountered, all the operators in the stack up to are popped and appended to output string which now becomes, **abc*-** and the stack becomes empty. Next, “ / ” is pushed into the stack. “ **(** ” is pushed into the stack, which becomes “ **/(** ”.
- **d** is now added to the output which becomes **abc*-d**. When “ ***** ” comes, as we have “ **(** ” as stack top, push it into the stack, the stack becomes **/(***.
- **e** is now added to the output which becomes **abc*-de**. When “ ***** ” comes, as it has equal priority with the stack top that is “ ***** ”, the “ ***** ” from the stack is popped and added to the output which becomes **abc*-de***. The new “ ***** ” is pushed into stack which becomes **/(***.

- **f** is now added to the output which becomes **abc*-de*f**. When " + " comes, as it has lower priority with the stack top that is " * "**,** the " * " from the stack is popped and added to the output which becomes **abc*-de*f***. The new " + " is pushed into the stack.
- **g** is now added to the output which becomes **abc*-de*f*g**. When " **)** " is encountered, all the operators in the stack are popped and appended to output string up to " **)",** which now becomes, **abc*-de*f*g+** and the stack is left with " / ".
- At the end of the input string, all the operators left in the stack are appended to the output which finally becomes **ab c*- de* f*g+/**

c) a / b * (c + (d-e))

- First, **a** is added to the output string. " / " is pushed into the stack. **b** is added to the output string which becomes, **ab**. When " * " is encountered, as it has equal priority to the top of the stack that is " / ", stack top is popped and appended to the string which becomes, **ab/**.
- **" * "** is pushed into the stack. " **(** " is pushed into the stack whose contents now are " ***(** ". **c** is added to the output which now becomes **ab / c**. " + " is pushed into the stack which now contains " *(+ ". "**(** " is pushed into the stack whose contents now are " ***(+(** ".
- **d** is added to the output, which now becomes **ab/cd**. When " **-** " is encountered, as stack top is " **(** ", it is pushed into the stack which now becomes " ***(+(-** ". **e** is added to the output, which now becomes **ab / cde**.
- When " **)** " is encountered, all the operators in the stack up to (are popped and appended to output string which now becomes, **ab / cde-** and the stack is left with " ***(+** " .

- When the second " **)** " is encountered, all the operators in the stack up to " **(** " are popped and appended to output string which now becomes, **ab / cde-**+ and the stack is left with " * " which is also added to the output.
- The final output string is **a b / c de - +***.

2) Write a java code to implement the infix to postfix algorithm as described below. Test the three infix expressions given in problem-1 and check your work for problem-1.

Answer:

Code:

```
import java.io.IOException;
public class IntoPost {
public String convertTopostfix(String infix) {
Stack operatorStack;
String postfix = "";
int Size = infix.length();
operatorStack = new Stack(Size);
int i=0;
while (i < Size) {
char character = infix.charAt(i);
switch (character) {
case '^':
operatorStack.push(character);
break;
case '+':
```

```
case '-':
case '*':
case '/':
while (!operatorStack.isEmpty()) {
char top = operatorStack.pop();
int precedence1= precedenceLevel(character);
if (top == '(') {
operatorStack.push(top);
break;
} else {
int precedence2;

if (top == '+' || top == '-')
precedence2 =0 ;
else
precedence2 = 1;
if (precedence2 < precedence1) {
operatorStack.push(top);
break;
}
else postfix = postfix + top;
}
}
operatorStack.push(character);
break;
```

```
case '(':
operatorStack.push(character);
break;
case ')':
char topCharacter = operatorStack.pop();
while (!operatorStack.isEmpty() && topCharacter!='(') {
postfix = postfix + topCharacter;
topCharacter = operatorStack.pop();
}
break;
default:
postfix = postfix + character;
break;

}
i++;
}
while (!operatorStack.isEmpty()) {
char topCharacter=operatorStack.pop();
postfix = postfix + topCharacter;
}

return postfix;
}
//Algorithm code ends here
```

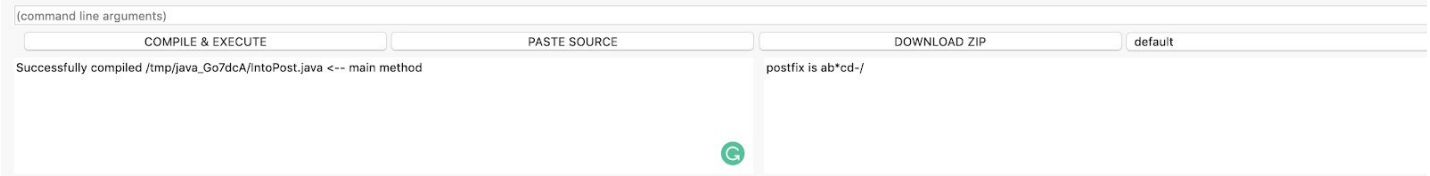
```
//This function is used to give the precedence to operators..
int precedenceLevel(char operator) {
switch (operator) {
case '+':
case '-':
return 0;
case '*':
case '/':
return 1;
default:
throw new IllegalArgumentException("Operator unknown: " + operator);
}
}
public static void main(String[] args) throws IOException {
String input = "a*b/(c-d)"; //Testing the Question 1
String postfix;
IntoPost theTrans = new IntoPost();
postfix = theTrans.convertTopostfix(input);
System.out.println("postfix is " + postfix + '\n');



}
class Stack {
private int maxSize;
private char[] stackArray;
```

```
private int top;


public Stack(int max) {

maxSize = max;

stackArray = new char[maxSize];

top = -1;

}

public void push(char j) {

stackArray[++top] = j;

}

public char pop() {

return stackArray[top--];

}

public char peek() {

return stackArray[top];

}

public boolean isEmpty() {

return (top == -1);

}

}
```

Test Results:

a) a*b/(c-d)

```
public static void main(String[] args) throws IOException {
String input = "a*b/(c-d)";
String postfix;
IntoPost theTrans = new IntoPost();
postfix = theTrans.convertTopostfix(input);
System.out.println("postfix is " + postfix + '\n');
}
class Stack {
private int maxSize;
private char[] stackArray;
private int top;

public Stack(int max) {
maxSize = max;
stackArray = new char[maxSize];
top = -1;
}
public void push(char j) {
stackArray[++top] = j;
}
public char pop() {
return stackArray[top--];
}
public char peek() {
return stackArray[top];
}
public boolean isEmpty() {
return (top == -1);
}
}
}
```



b) (a-b * c) / (d*e* f+g)

```
public static void main(String[] args) throws IOException {
String input = "(a-b * c) / (d*e* f+g)";
String postfix;
IntoPost theTrans = new IntoPost();
postfix = theTrans.convertTopostfix(input);
System.out.println("postfix is " + postfix + '\n');
}
class Stack {
private int maxSize;
private char[] stackArray;
private int top;

public Stack(int max) {
maxSize = max;
stackArray = new char[maxSize];
top = -1;
}
public void push(char j) {
stackArray[++top] = j;
}
public char pop() {
return stackArray[top--];
}
public char peek() {
return stackArray[top];
}
public boolean isEmpty() {
return (top == -1);
}
}
}
```

(command line arguments)

COMPILE & EXECUTE | PASTE SOURCE | DOWNLOAD ZIP | default

Successfully compiled /tmp/java_nNRr1S/IntoPost.java <-- main method

postfix is ab c*- de* f*g+/

c) a / b * (c + (d-e))

```
public static void main(String[] args) throws IOException {
String input = "a / b * ( c +  ( d-e ) )";
String postfix;
IntoPost theTrans = new IntoPost();
postfix = theTrans.convertTopostfix(input);
System.out.println("postfix is " + postfix + '\n');
}
class Stack {
private int maxSize;
private char[] stackArray;
private int top;

public Stack(int max) {
maxSize = max;
stackArray = new char[maxSize];
top = -1;
}
public void push(char j) {
stackArray[++top] = j;
}
public char pop() {
return stackArray[top--];
}
public char peek() {
return stackArray[top];
}
public boolean isEmpty() {
return (top == -1);
}
}
}
```

(command line arguments)

COMPILE & EXECUTE | PASTE SOURCE | DOWNLOAD ZIP | default

Successfully compiled /tmp/java_KE7kxQ/IntoPost.java <-- main method

postfix is a b / c de - +*

3) Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that k >= 1, $\in$ > 0, and c>1 are constants. Your answer should be in the form of the table with "Yes" or "No" written in each box.

Answer:

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ϵ					
b.	n^k	c^n					
c.	$\sqrt{n}$	$n^{\sin n}$					
d.	2^n	$2^{n/2}$					
e.	$n^{\lg c}$	$c^{\lg n}$					
f.	$\lg(n!)$	$\lg(n^n)$					

	O	*o*	Ω	*ω*	Θ
a	YES	YES	NO	NO	NO
b	YES	YES	NO	NO	NO
c	NO	NO	NO	NO	NO
d	NO	NO	YES	YES	NO
e	YES	NO	YES	NO	YES
f	YES	NO	YES	NO	YES

4) Using Big Oh notation, indicate the time requirement of each of the following tasks in the worst case.

Answer:

a. Display all the integers in an array of integers.

- The time required for displaying all the integers in an array in the worst case is O(n).

b. Display all the integers in a chain of linked nodes.

- The time required for displaying all the integers in an array in the worst case is O(n).

c. Display the nth integer in an array of integers.

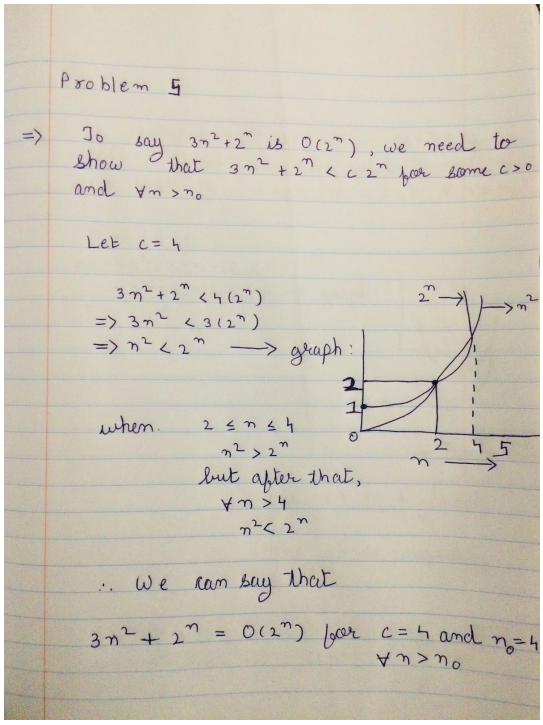
- The time required for displaying all the integers in an array in the worst case is O(1).
- Just compute a (base address + n(memory size)) so we do not need a ‘for’ loop.

d. Compute the sum of the first n even integers in an array of integers.

- The time required for displaying all the integers in an array in the worst case is O(n).
- It occurs when all elements in an array are even.

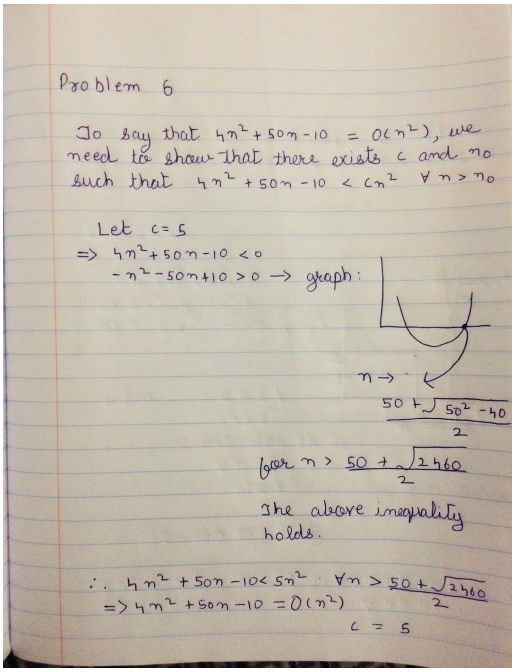
5) Show that $3n^2 + 2^n$ *is* $O(2^n)$. What Values of c and n did you use?

Answer:



6) Show that $4n^2 + 50n - 10$ *is* $O(n^2)$. What Values of c and n did you use?

Answer:

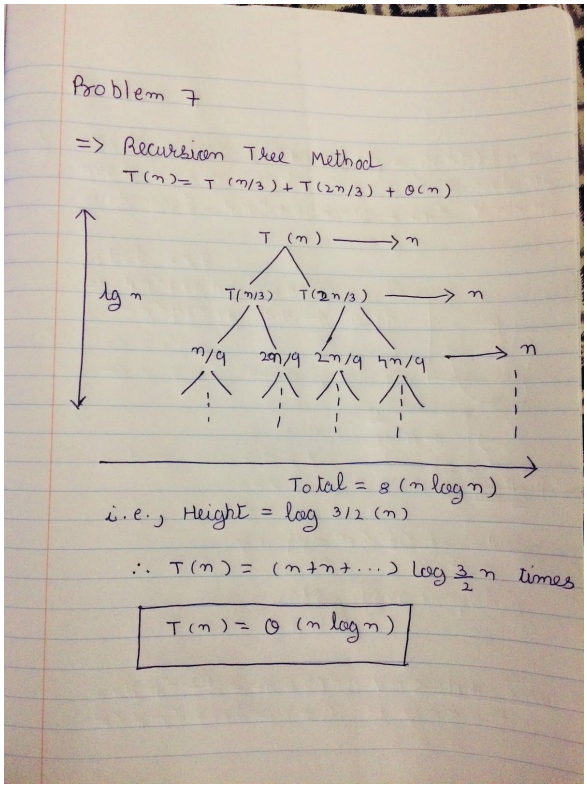


7) Prove the running time of the following T(n) using a recursion tree to come up with a guess and use the substitution to prove.

$T(n) = T(n/3) + T(2n/3) + \Theta(n)$.

Answer:

Recursion Tree Method:



Substitution Method: