# Data Structures and Algorithms

# INFO 6205

# Homework 3

# Due: February 21, 2022

Put all your java, compiled class files and documentation files into a zip file Homework3.zip

and submit it via drop box on Canvas before the END of due date. Put your name on all

.java files.

1. What is the time complexity of the following code, and why?

public makeSentence ( String[] words) {

String sentence=“”;

for (String w:words) {

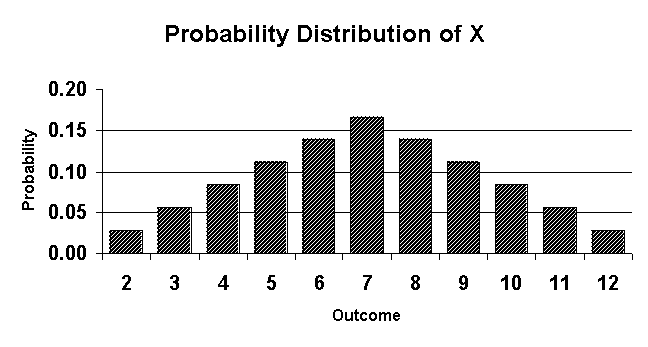
sentence+=w;

}

return sentence;

}

2. Suppose the customers enter a Bank has the following histogram:



a) What is Random variable?

b) What are the probabilities for this distribution throwing two dices?

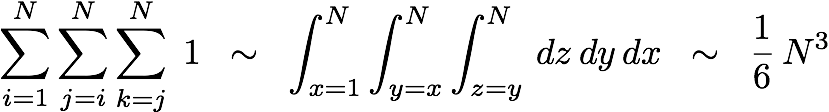
c) Calculate the Mean and Standard Deviation of this Probability distribution?

d) Explain the observed statistics for a Bank system.

3. Write code that results to the following running time. The 3-Sum Triple loop has

the following running time estimate.

A) Do Not prove Math. Just want explaining the math. What does the math does

represent and why the result is 1/6 N^3

B) If you have 2-Sum loop, what change would you need to make to Math in (a)

4. What is Stack data structure, and stack operations? Explain

5. Consider String “It was the best of time”. Start with the first word, design a Stack such that when you read back the words, the order of string does not change. Provide code for all necessary operations of Stack. Compile and run the code.

6. The Recursive operations for Factorial and Fibonacci sequence was discussed in class.

A) For Fibonacci sequence with n=7, the following diagram shows its Tree Structure

a) Is this diagram iterative or recursive?

b) What data structure is used to implement recursion?

c) Provide Tree Structure for n=5 step-by-step. What differences do you see in

diagrams between n=6 and n=7?

d) What are Pros and Cons between iterative and recursive Algorithms?

e) Write recursive Java code for both n=6 and n=7

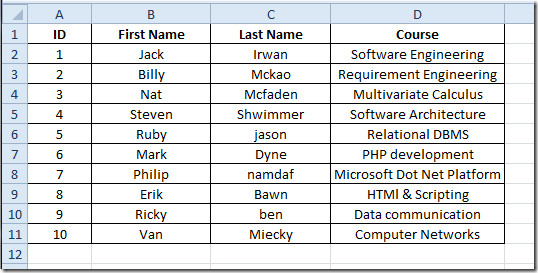
f) Compile and Run your programs, report resultspasted-image.tiff

B) For factorial 8!a) Show recursive stack operations, provide details step-by-step, b**)** Walk through your stack operations and provide the result. c**)** Write Java code with input factorial 6! d**)** Compile and run your program, what is the running time of your algorithm?

7. Consider following data to build Stack with:

A) LinkedList implementation

B) Array implementation



a) Create file “Input.txt” with this data

b) Read input.data into an ArrayList

c) Create Stack with LinkedList implementation

d) Write Node data structure of your input data

e) Stack must support all operations of stack: push, pop. is-empty, is-full

f) Write a Test program to test your linked implementation of Stack:

—push 4 elements into stack

—pop 5 elements from stack

—push all elements into stack

—push

11 john henry “software development”

12 justin morgan “engineering statistics”

—pop all elements from stack

—push 8 elements into stack

—pop 9 elements from stack

—push all elements into stack

—pop all elements from stack

—Print stack with the goal:

i) reverse order ii) original order as was first read into array list

g) Compile and Run your program

h) what is Stack LinkedList time-complexity?

i) Repeat (a)—(h) with Stack fixed Array Implementation

j) What are the consequences of oversizing or undersizing fixed array size?

8. Consider following Algorithm to “Evaluate Infix Expressions” with Two arrays:

Test data:

A \* B / C + (D + E - (F \* (G / H)))

(1 + 3 + ( ( 4 / 2 ) \* ( 8 \* 4 ) ))

(4 + 8) \* (6 - 5)/((3 - 2) \* (2 + 2))

A) Step through algorithm to develop a **Stack Table** for for each Infix expression

B) Write Java code to test each Infix Expression

C) Compile and Run

**Algorithm:**

Iterate through given expression, one character at a time

1. If the character is an operand, push it to the operand stack.

2. If the character is an operator,

1. If the operator stack is empty then push it to the operator stack.

2. Else If the operator stack is not empty,

• If the character’s precedence is greater than or equal to the precedence of the stack top of the operator stack, then push the character to the operator stack.

• If the character’s precedence is less than the precedence of the stack top of the operator stack then do Process (as explained above) until character’s precedence is less or stack is not empty.

3. If the character is “(“, then push it onto the operator stack.

4. If the character is “)”, then do *Process* (as explained above) until the corresponding “(” is encountered in operator stack. Now just pop out the “(“.

Once the expression iteration is completed and the operator stack is not empty,

do *Process* until the operator stack is empty.  The values left in the operand stack

is our final result.

**9.** Consider the following Algorithm to convert Infix expression to Postfix.

A) Infix expression example: (A + B) \* C + D / (E + F \* G) - H

B) Apply Algorithm to Infix example, show step-by-step

C) Write Java code for the algorithm to convert Infix to Postfix expression

Algorithm:

while there are more symbols to read

read the next symbol

case:

operand --> output it.

’(’ --> push it on the stack.

’)’ --> pop operators from the stack to output

until a ’(’ is popped; do not output either of

the parentheses.

operator --> pop higher- or equal-precedence operators

from the stack to the output; stop before

popping a lower-precedence operator or

a ’(’. Push the operator on the stack.

end case

end while

pop the remaining operators from the stack to the output

10. Consider this Algorithm to “Evaluate Postfix Expression”: **10 2 8 \* + 3 -**

Algorithm: Maintain a stack and scan the postfix expression from left to right – When we get a number, output it – When we get an operator, pop the top element in the stack until there is no operator having higher priority than this operator, and then push (operator) into the stack – When the expression is ended, pop all the operators remain in the stack:

A) Show Stack step-by-step

B) Write Java code to compute postfix expression

11. Consider the following code with Array Stack implementation

A) Explain this code

B) Why would an application need such a code, Explain

C) What code change would you make to this code to correct over-sizing?

public ResizingArrayStackOfStrings()

{ s = new String[1]; }

public void push(String item)

{

if (N == s.length) resize(2 \* s.length);

s[N++] = item;

}

private void resize(int capacity)  
 {   
 String[] copy = new String[capacity];  
 for (int i = 0; i < N; i++)  
 copy[i] = s[i];  
 s = copy;  
 }

A) This code is used for resizing the length of the array. If the length hits the current length [ N == s.length ] => then, a copy of array of string is created with capacity double the size of current length and current array is assigned to copied array with new capacity.

B) This code is helpful to prevent undersizing of an array by doubling the current capacity.

C) Preventing oversizing of an array is crucial from the perspective of memory utilization. To check oversizing – While deleting an element from array, we will check whether the array size is 4 times the current number of elements and if it is true, we will resize the array by resizing it to half size.

Public String pop(){

String item = s[--N];

s[N] = null;

if (N > 0 && N == s.length/4)

resize(s.length/2); return item;

}