Data Structures and Algorithms
INFO 6205
Homework 3
Due: September 26, 2019

Name: Qian Cai
NU ID: 001389278

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

A) For factorial 6! 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?

B) For Fibonacci sequence with n=5, a) Show recursive stack operations, provide details step-by-step, b) Walk through your stack operations, provide the result. c) Provide Iterative algorithm for Fibonacci function, d) Write Java code for both recursive and iterative algorithms. e) Compile and Run your program.

C) For Towers of Hanoi problem with n=5 discs, how does the algorithm work? What data structures would you use? provide step by step operations. Write Java code, compile and run program.

A) b)

Factorial(6) **720**
↓ ↑
6*Factorial(5) **return 6*(5*4*3*2*1)**
↓ ↑
5*Factorial(4) **return 5*(4*3*2*1)**
↓ ↑
4* Factorial(3) **return 4*(3*2*1)**
↓ ↑
3* Factorial(2) **return 3*(2*1);**
↓ ↑
2*Factorial(1) **return 2*1**
↓ ↑
1 **return 1**

Stack	**Operation**	**Description**
F(6)		
F(6), F(5)		
F(6), F(5), F(4)		
F(6), F(5), F(4), F(3)		

F(6), F(5), F(4), F(3), F(2)		
F(6), F(5), F(4), F(3), F(2), F(1)		
F(6), F(5), F(4), F(3), F(2)	**Return F(1)=1**	**Pop and manage F(1)**
F(6), F(5), F(4), F(3)	**Return F(2)*1=2**	**Pop and manage F(2)**
F(6), F(5), F(4)	**Return F(3)*2=6**	**Pop and manage F(3)**
F(6), F(5)	**Return F(4)*6=24**	**Pop and manage F(4)**
F(6)	**Return F(5)*24=120**	**Pop and manage F(5)**
	Return F(6)*120=720	**Pop and manage F(6)**

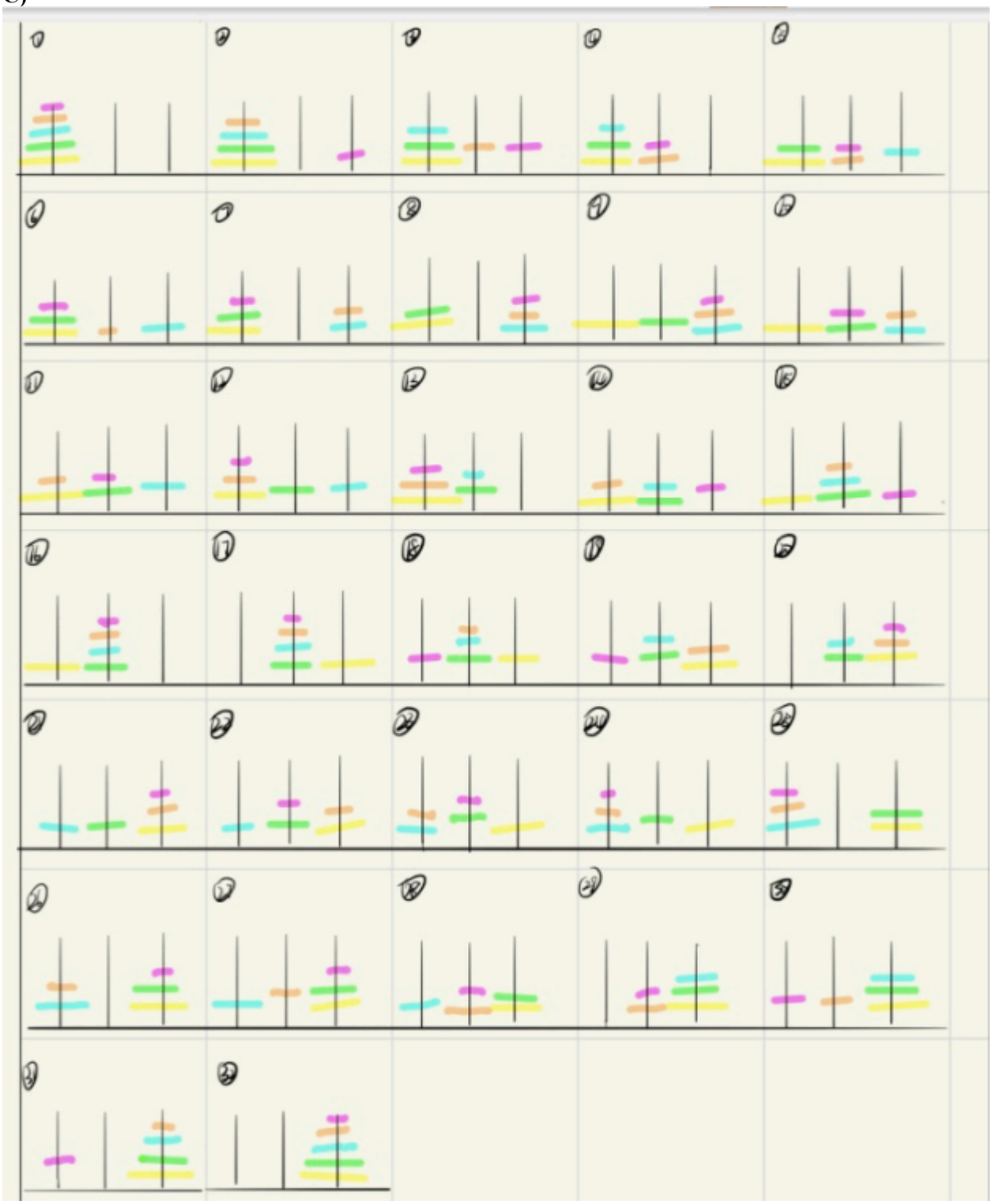
d) The running time of my algorithm is O(n)

B)

Stack	**Operation**	**Description**
Fib(5)	**Fib(4)+Fib(3)**	**Pop and manage Fib(5)**
Fib(4)	**Fib(4)+Fib(2)+Fib(1)**	**Push Fib(4) into stack and manage Fib(3) firstly.**
Fib(4)+Fib(2)	**Fib(4)+Fib(2)+1**	**Push Fib(2) into stack and manage Fib(1).**
Fib(4)	**Fib(4)+Fib(1)+Fib(0)+1**	**Pop and manage Fib(2)**
Fib(4)	**Fib(4) +2**	**Mange Fib(1) and Fib(0).**
	Fib(3)+Fib(2)+2	**Pop and manage Fib(4)**
Fib(3)	**Fib(3)+Fib(1)+Fib(0)+2**	**Push Fib(3) into stack and manage Fib(2).**
Fib(3)	**Fib(3)+3**	**Mange Fib(1) and Fib(0).**
	Fib(2)+Fib(1)+3	**Pop and manage Fib(3)**
Fib(2)	**Fib(2)+4**	**Push Fib(2) into stack and manage Fib(1).**
	Fib(1)+Fib(0)+4	**Pop and manage Fib(2)**
	Result=5	**Mange Fib(1) and Fib(0).**

Algorithm:
case:
N<=1 → return n
N>1 → return f(n-1)+f(n+1);

C)



The data structures I used is Stack.
Algorithm:

1. **Move the (n-1)disc from left to the middle.**
2. **Move the n disc from left to the right.**
3. **Move the (n-1)disc from middle to the right.**

2. For the LinkedList implementation of Queue example discussed in class, write a TestLinkedListQueue class to test enqueue, dequeue, isEmpty and other operations as needed.

3. Describe the Array Implementation of Queue with one example. You need to provide a sample data and walk through the enqueue and dequeue, and other operations as necessary and manage the head and tail pointers. Note: I have already provided you with one example.
The sample data: Java is the best programming language.

enqueue(): add new item at q[tail].
dequeue(): remove item from q[head].

Head ↓ **Tail** ↓

Java	is	the	best	programming	null	null	null	null	null
0	1	2	3	4	5	6	7	8	9

Enqueue():add new item at q[tail].
New item=language.
Tail=5

Head ↓ **Tail** ↓

Java	is	the	best	programming	language	null	null	null	null
0	1	2	3	4	5	6	7	8	9

dequeue(): remove item from q[head].
Head=0.
Item=Java

Head ↓ **Tail** ↓

null	is	the	best	programming	language	null	null	null	null
0	1	2	3	4	5	6	7	8	9

				item[]					
Input	**Out**	**head**	**tail**	**0**	**1**	**2**	**3**	**4**	**5**

		0	0	null	null	null	null	null	null
Java		0	1	Java	null	null	null	null	null
is		0	2	Java	is	null	null	null	null
the		0	3	Java	is	the	null	null	null
best		0	4	Java	is	the	best	null	null
program-ming		0	5	Java	is	the	best	program-ming	null
language		0	6	Java	is	the	best	program-ming	lan-guage
	Java	1	6	null	is	the	best	program-ming	lan-guage
	is	2	6	null	null	the	best	program-ming	lan-guage
	the	3	6	null	null	null	best	program-ming	lan-guage
	best	4	6	null	null	null	null	program-ming	lan-guage
	program-ming	5	6	null	null	null	null	null	lan-guage
	language	6	6	null	null	null	null	null	null

Step	Operation	Descirption
1	Initialize the size of array.	The capacity of array is 100
2	Enqueue “Java”	Assess the situation. The situation of the queque is empty. So set head=0 and put “Java” into s[head]. Then set tail=1
3	Enqueue “is”	Assess the situation. The situation of the queque is not empty. So tail=1 and put “is” into s[tail]. Then let tail++.
4	Enqueue “the”	Assess the situation. The situation of the queque is not empty. So tail=2 and put “the” into s[tail]. Then let tail++.
5	Enqueue “best”	Assess the situation. The situation of the queque is not empty. So tail=3 and put “best” into s[tail]. Then let tail++.
6	Enqueue “programming”	Assess the situation. The situation of the queque is not empty. So tail=4 and put “programming” into s[tail]. Then let tail++.

7	**Enqueue “language”**	**Assess the situation. The situation of the queque is not empty. So tail=5 and put “language” into s[tail]. Then let tail++.**
8	**Dequeue. The result is “Java”**	**Assess the situation. The situation of the queque is not empty. The head=0 and item=s[head]. Let head ++. Then return item and show the result is “Java”**
9	**Dequeue. The result is “is”**	**Assess the situation. The situation of the queque is not empty. The head=1 and item=s[head]. Let head ++. Then return item and show the result is “is”**
10	**Dequeue. The result is “the”**	**Assess the situation. The situation of the queque is not empty. The head=2 and item=s[head]. Let head ++. Then return item and show the result is “the”**
11	**Dequeue. The result is “best”**	**Assess the situation. The situation of the queque is not empty. The head=3 and item=s[head]. Let head ++. Then return item and show the result is “best”**
12	**Dequeue. The result is “programming”**	**Assess the situation. The situation is the queque is not empty. The head=4 and item=s[head]. Let head ++. Then return item and show the result is “programming”**
13	**Dequeue. The result is “language”**	**Assess the situation. The situation is the queque is not empty. The head=5 and item=s[head]. Let head ++. Then return item and show the result is “language”**

The code of implementation：

```
/**
 *Name: Qian Cai
 * NU ID:001389278
 * Created by Qian Cai on 2019/9/20.
 */
public class H3_3 {
    private String[]s;

    private int head;
    private int tail;

    public H3_3(int capacity)
    { s = new String[capacity]; }

    public boolean isEmpty()
    { if(tail==head)
          return true;
    else return false;
    }

    public void enqueue(String item)
    {
        if(isEmpty())
        {
            head=0;
            s[head]=item;
            tail=1;
        }
        else{
            s[tail]=item;
            tail++;
        }
    }
```

```
public String dequeue()
{
    if(isEmpty())
        return null;
    else{
       String item=s[head];
       head++;
       return item;
    }


}
public static void main(String[] args)
{
    H3_3 h=new H3_3( capacity: 100);
    h.enqueue( item: "Java");
    h.enqueue( item: "is");
    h.enqueue( item: "the");
    h.enqueue( item: "best");
    h.enqueue( item: "programming");
    h.enqueue( item: "language");
    System.out.print(h.dequeue()+" "+h.dequeue()+" "+h.dequeue()+" "+h.dequeue()+" "+h.dequeue()+" "+h.dequeue());
```

4. Java Generics allow you to build collections with unique data type. To perform uniqueness, comparisons of object types need to be made:

A) Using compareTo() method from Comparable interface, the equals, and hashcode. Explain the differences?

CompareTo(): The comparison of CompareTo() is to compare strings on the basis of Unicode value of each character in the strings. It returns positive number, negative number or 0.
Equals: The shallow comparison of equals method is checking if two Object references refer to the same Object. The deep comparison of equals method is compare the data members of objects. It returns true or false.
Hashcode: The comparison of hashcode is compare the hashcode value of objects. It returns the hashcode value as an Integer.

B) Java String class object hashcode is described as following:

$$h(s) = \sum_{i=0}^{n-1} s[i] \cdot 31^{n-1-i}$$

What is the hashcode 32-bit integer number for string =“Hello to the World“,
a) mathematically by hand, b) Write Java code

CHAR	(space)	H	e	l	o	t	h	W	r	d
ASCII	32	72	101	108	111	116	104	87	114	100

Hashcode=$72 * 31^{17}+101 * 31^{16}+108 * 31^{15}+108 * 31^{14}+111 * 31^{13}+32 * 31^{12}+116 * 31^{11}+111 * 31^{10}+32 * 31^{9}+116 * 31^{8}+104 * 31^{7}+101 * 31^{6}+32 * 31^{5}+87 * 31^{4}+111 * 31^{3}+114 * 31^{2}+108 * 31^{1}+100 * 31^{0}$

5. Write a Iterative method to `sumDigits` that has one integer parameter and returns the sum of the digits in the integer specified. The method should throw `IllegalArgumentException` if the integer specified is negative. Remember, your method should not use. For example, if the integer is 26497, then this method should return 28.

6. Write a recursive method `countStringBinary` that has one integer parameter n and returns the number of binary strings of length n that do not have two consecutive 0's. For example, for n = 4, the number of binary strings of length 4 that do not contain two consecutive 1's is 8: 1111, 1110, 1101, 1011, 1010, 0111, 0110, 0101

7. 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 the 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

Symbol	Scanned	STACK	Postfix Expression	Description
1		**(**		**Start**
2	**A**	**(**	**A**	**Output it**
3	*****	**(***	**A**	**Push the operator on the stack.**
4	**B**	**(***	**AB**	**Output it**

5	**/**	**(/**	**AB***	**Pop equal-precedence operators from the stack to the output**
6	**C**	**(/**	**AB*C**	**Output it**
7	**+**	**(+**	**AB*C/**	**Pop higher-precedence operators from the stack to the output**
8	**(**	**(+(**	**AB*C/**	**Push it on the stack.**
9	**D**	**(+(**	**AB*C/D**	**Output it**
10	**+**	**(+(+**	**AB*C/D**	**Push the operator on the stack.**
11	**E**	**(+(+**	**AB*C/DE**	**Output it**
12	**-**	**(+(-**	**AB*C/DE+**	**Pop equal-precedence operators from the stack to the output**
13	**(**	**(+(-(**	**AB*C/DE+**	**Push it on the stack.**
14	**F**	**(+(-(**	**AB*C/DE+F**	**Output it**
15	*****	**(+(-(***	**AB*C/DE+F**	**Push the operator on the stack.**
16	**(**	**(+(-(*(**	**AB*C/DE+F**	**Push it on the stack.**
17	**G**	**(+(-(*(**	**AB*C/DE+FG**	**Output it**
18	**/**	**(+(-(*(/**	**AB*C/DE+FG**	**Push the operator on the stack.**
19	**H**	**(+(-(*(/**	**AB*C/DE+FGH**	**Output it**
20	**)**	**(+(-(***	**AB*C/DE+FGH/**	**Pop operators from the stack to the output until a '(' is popped; do not output either of the parentheses.**
21	**)**	**(+(-**	**AB*C/DE+FGH/***	**Pop operators from the stack to the output until a '(' is popped; do not output either of the parentheses.**
22	**)**	**(+**	**AB*C/DE+FGH/*-**	**Pop operators from the stack to the output until a '(' is popped; do not output either of the parentheses.**
23	**)**	**Empty**	**AB*C/DE+FGH/*-+**	**Pop operators from the stack to the output until a '(' is**

				popped; do not output either of the parentheses.