ASSIGNMENT 4

Jaspreet Kaur

11529746

1. Consider the following:

   a) For Power (x, n) with n=7, a) write Iterative algorithm step-by-step, and b) write Recursive algorithm, step-by-step, c) Write Java code for a and b, compile and run

   b) For Towers of Hanoi problem with n=8 discs, how does the algorithm work? What data structures would you use? provide step by step operations. Write Java code, compile and run program. https://introcs.cs.princeton.edu/java/23recursion/

Answer –

(a)

a) Iterative Algorithm –
   1. Assign result = 1.
   2. For loop from 1 to n –
      - Update result for each iteration as -> result *= x;
   3. Return the value of result

```
public int getPower(int x, int n){
        int result = 1;
        for(int i=1;i<=n;i++){
                result *= x;
        }
        return result;
}
```

Step 1 – result = result * 2 = 1*2 = 2
Step 2 – result = result * 2 = 2*2 = 4
Step 3 – result = result * 2 = 4*2 = 8
Step 4 – result = result * 2 = 8*2 = 16
Step 5 – result = result * 2 = 16*2 = 32
Step 6 – result = result * 2 = 32*2 = 64
Step 7 – result = result * 2 = 64*2 = 128

b) Recursive Algorithm –
   1. The base condition is n=0 => return 1.
   2. If n>0 => recursive call -> return x * getPower(x, n-1);

```
public int getPower(int x, int n){
        if(n == 0){
                return 1;
        }
        else{
```

```
                return x * getPower(x, n-1);
            }
        }
```

2 * getPower(2,6)
2 * getPower(2,5)
2 * getPower(2,4)
2 * getPower(2,3)
2 * getPower(2,2)
2 * getPower(2,1)
2 * getPower(2,0) => return 1

Backtracking -
2 * 1 = 2
2 * 2 = 4
2 * 4 = 8
2 * 8 = 16
2 * 16 = 32
2 * 32 = 64
2 * 64 = 128

c) **JavaFiles** => Qs1_Power_Iterative, Qs1_Power_Recursive

(b)

1 (b) Tower of Hanoi has following 2 rules

1) Only one disk can be removed at a time

2) Disks are in ~~according order~~. lower (top) to upper (base)

For $n=8 \Rightarrow$ there will be $2^8 - 1 \Rightarrow 256 - 1 = \boxed{255}$

Let A be source peg, B be auxillary peg and C be destination peg

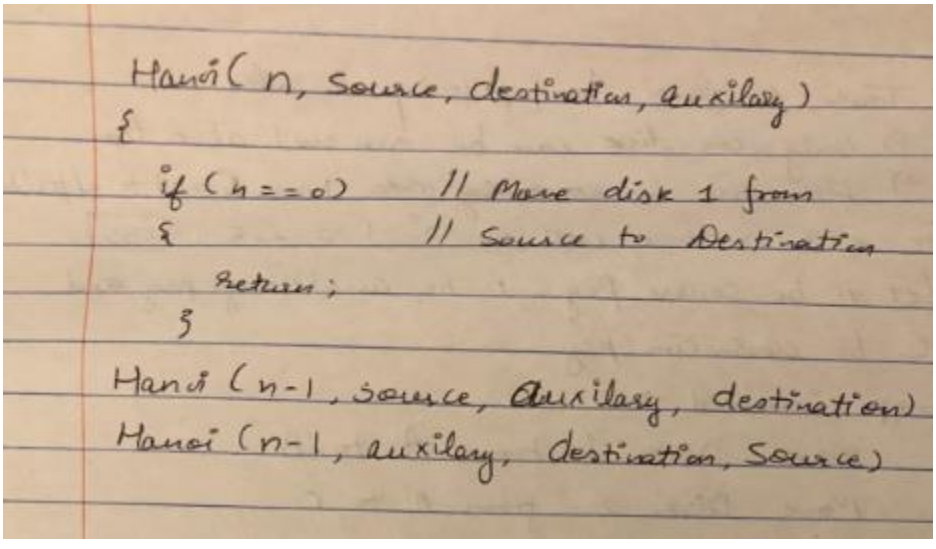
Steps involved →

Move Disk 1 from A to B
Move Disk 2 from A to C
Move Disk 1 from B to C
Move Disk 3 from A to B
Move Disk 1 from C to A
Move Disk 2 from C to B
Move Disk 1 from A to B
Move Disk 4 from A to C
Move Disk 1 from B to C
Move Disk 2 from B to A
Move Disk 1 from C to A
Move Disk 3 from B to C
Move Disk 1 from A to B
Move Disk 2 from A to C
Move Disk 1 from B to C.

and so on

Data Structure →

The Tower of Hanoi Problem is solved using stack, as well as Priority Queue But, stack is used most widely.



Code -> **JavaFiles** /Qs1(b)_Tower

Qs2. For the following Algorithm

a) Why would you use Grey Binary?

b) Convert Binary numbers to Grey numbers

11011
11011001101
11001100110

c) Write Java code for the following Algorithm to convert Binary to Grey number:

```
binary_to_grey(n)
    if n == 0
        grey = 0;
    else if last two bits are opposite  to each other
        grey = 1 + 10 * binary_to_gray(n/10))
    else if last two bits are same
        grey = 10 * binary_to_gray(n/10))
```

d) Write Algorithm to convert Grey to Binary Number

e) Write Java code to your Algorithm in (d)

f) Write step-by-step Algorithm to generate n-bit Gray code

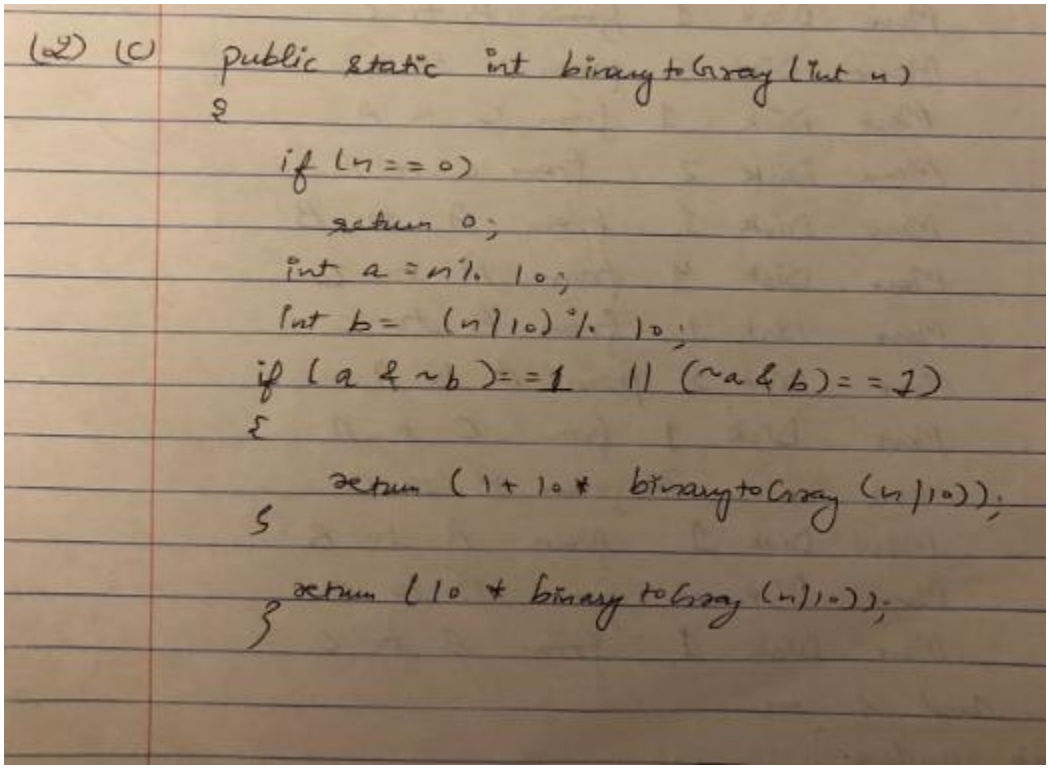
g) Apply algorithm to generate 3-bits Gray code

a) Gray code is an ordering of the binary numeral system such that two successive values differ in only 1 bit. Gray code is useful in the normal sequence of binary numbers generated by h/w that may cause an error during transition from one number to the next. Gray code solves this because only one bit changes its value when any transition occurs. Eg - Gray code is used in optical encoders.

b) 11011 -> 10110

11011001101 -> 10110101011
11001100110 -> 10101010101

c)



```
(2) (c)  public static int binaryToGray(int n)
{
    if (n == 0)
        return 0;
    int a = n % 10;
    int b = (n/10) % 10;
    if (a & ~b) == 1 || (~a & b) == 1)
    {
        return (1 + 10 * binaryToGray(n/10));
    }
    return (10 * binaryToGray(n/10));
}
```

d) Algorithm for Gray -> Binary

1. The MSB of the binary code is always equal to MSB of given gray code.
2. Other bits of the output binary code can be obtained by checking gray code bit at that index. If current gray code bit is 0, then copy previous binary code bit, else copy invert of previous binary code bit.

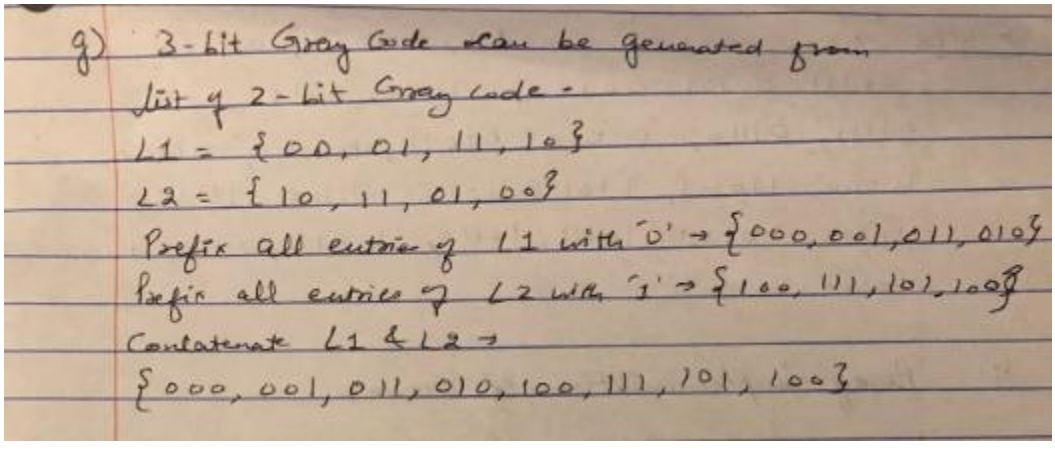
```
// MSB of binary code is same as Gray Code
Binary += gray.chartAt(0);
For(int i=0;i<gray.length();i++){
    // if current bit is 0
    If(gray.charAt(i) == '0'){
            Binary += binary.chartAt(i-1);
    }
    // concatenate invert of bit
    Else{
            Binary += flip(binary.charAt(i-1);
    }
}
```

e) **JavaFiles** -> Qs2(d)_GrayToBinary

f) **n-bit Gray Codes can be generated from list of (n-1)-bit Gray codes using following steps.**

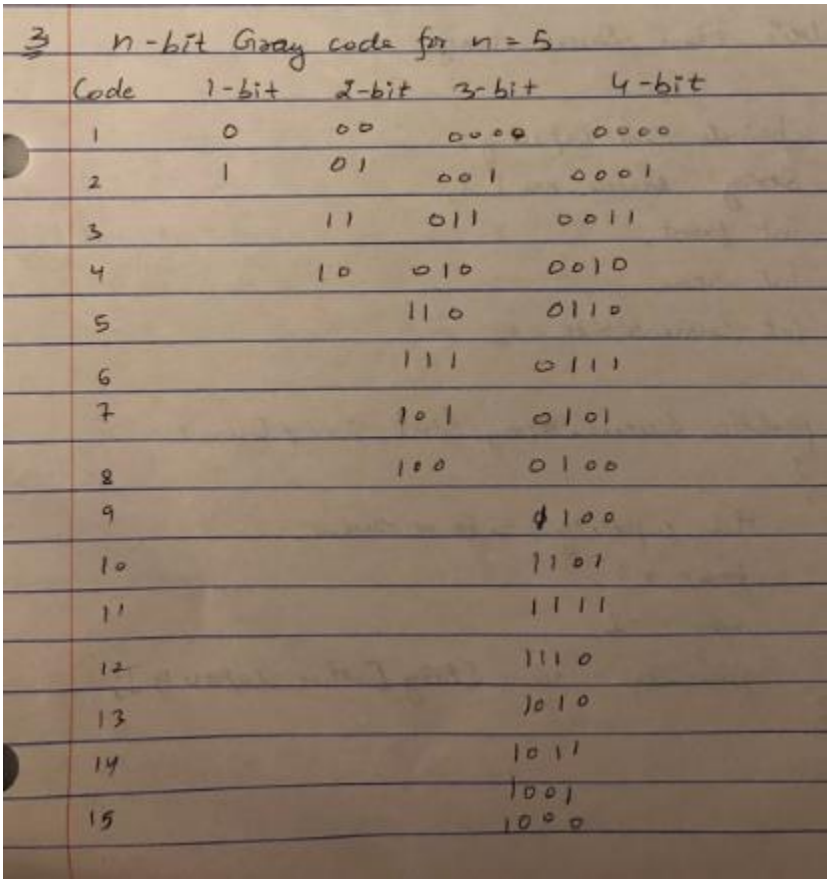
1. Let the list of (n-1)-bit Gray codes be L1. Create another list L2 which is reverse of L1.
2. Modify the list L1 by prefixing a '0' in all codes of L1.
3. Modify the list L2 by prefixing a '1' in all codes of L2.
4. Concatenate L1 and L2. The concatenated list is required list of n-bit Gray codes

g)



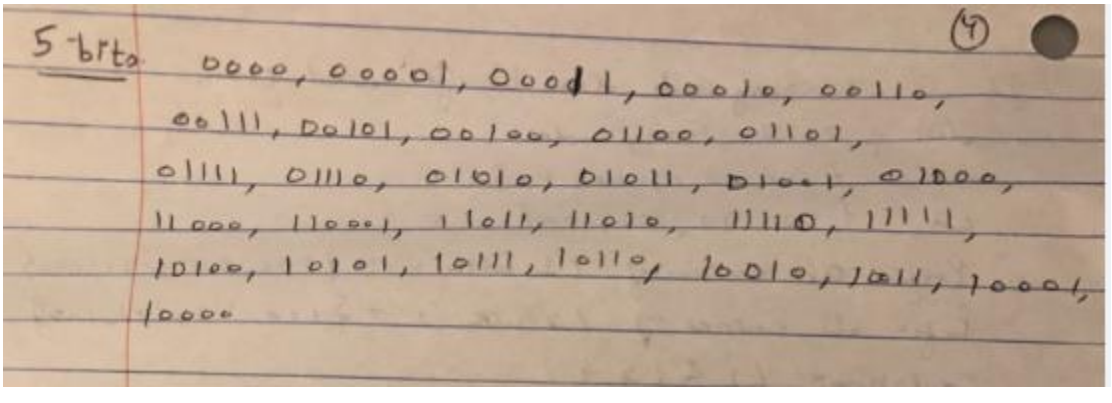
Qs3. . An *n*-bit Gray code is a list of the 2^n different *n*-bit binary numbers such that each entry in the list differs in precisely one bit from its predecessor. The *n* bit binary reflected Gray code is defined recursively. How does algorithm works for n=5, describe step-by-step. Write Java code, compile and run program. https://introcs.cs.princeton.edu/java/23recursion/

The algorithm for the n-bit gray code generation requires us to generate and store the entire (n-1)-bit Gray code sequence prior to generating any of the codes in the n-bit Gray code sequence, and hence the recursive approach is used. So for that the below table will be show how to generate and append prefix to the strings. So here the previous bits are copied and again copies in reverse manner and both of them are joined with prefix 0 and 1 respectively. For example, here I have attached generated 5-bit gray code.



3 n-bit Gray code for n = 5

Code	1-bit	2-bit	3-bit	4-bit
1	0	00	000	0000
2	1	01	001	0001
3		11	011	0011
4		10	010	0010
5			110	0110
6			111	0111
7			101	0101
8			100	0100
9				1100
10				1101
11				1111
12				1110
13				1010
14				1011
15				1001
				1000



(4)

5-bits 00000, 00001, 00011, 00010, 00110,
00111, 00101, 00100, 01100, 01101,
01111, 01110, 01010, 01011, 01001, 01000,
11000, 11001, 11011, 11010, 11110, 11111,
10100, 10101, 10111, 10110, 10010, 10011, 10001,
10000

Qs4. Describe the Array Implementation of Queue with “It was the best of times” example discussed in class. You need to walk through the enqueue and dequeue, and other operations and to manage the front and last pointers. The example shows queue B and queue C, what is the difference, explain.

4. Array Implementation of Queue -

```
public class Queue_Array
{
    private int capacity;
    String Queue_arr[];
    int front;
    int rear;
    int current size = 0;

    public Queue_Array (int size of Queue)
    {
        this.capacity = size of Queue;
        front = 0;
        rear = -1
        queue_arr = new String[this.capacity];
    }
```

```
public void enqueue (String data)
{
  if (isFull())
  { // Queue is full }
  else {
    rear++;
  if (rear == capacity-1)
  {
    rear = 0;
  }
  queue_arr[rear] = data;
  currentsize++;
  }
}

public void dequeue()
{
  if (isEmpty())
  { // Queue is Empty; }
  else {
    front++;
    if (front == capacity-1)
    {
      SOP(queue_arr[front-1] is removed);
    front = 0;
  }
```

```
else {
    sout(queue_arr[front-1] is removed);
  }
current_size --;
}
}

public boolean isFull()
{
    if (current_size == capacity)
    {
        return true;
    }
    return false;
}

public boolean isEmpty()
{
    if (current_size == 0) { return true; }
    else return false;
}
```

```
public static void main(String a[])
{
  String word = "It was the best of times";
  String[] split = word.split(" ");
  int n = split.length;

Queue ~~Array~~ ob = new QueueArray(n);
  for (String i : split)
  {
    Queue.enqueue(i);
  }
for (int i=0; i<n; i++)
  {
    sol(queue.dequeue() + " ");
  }
}
```

split → | It | was | the | best | of | times |

Queue →
It
It was
It was the
It was the best of
It was the best of times

Dequeue
It was the best of times
was the best of times
the best of times
of times

Print
It
It was the
It was the best

Dequeue
times

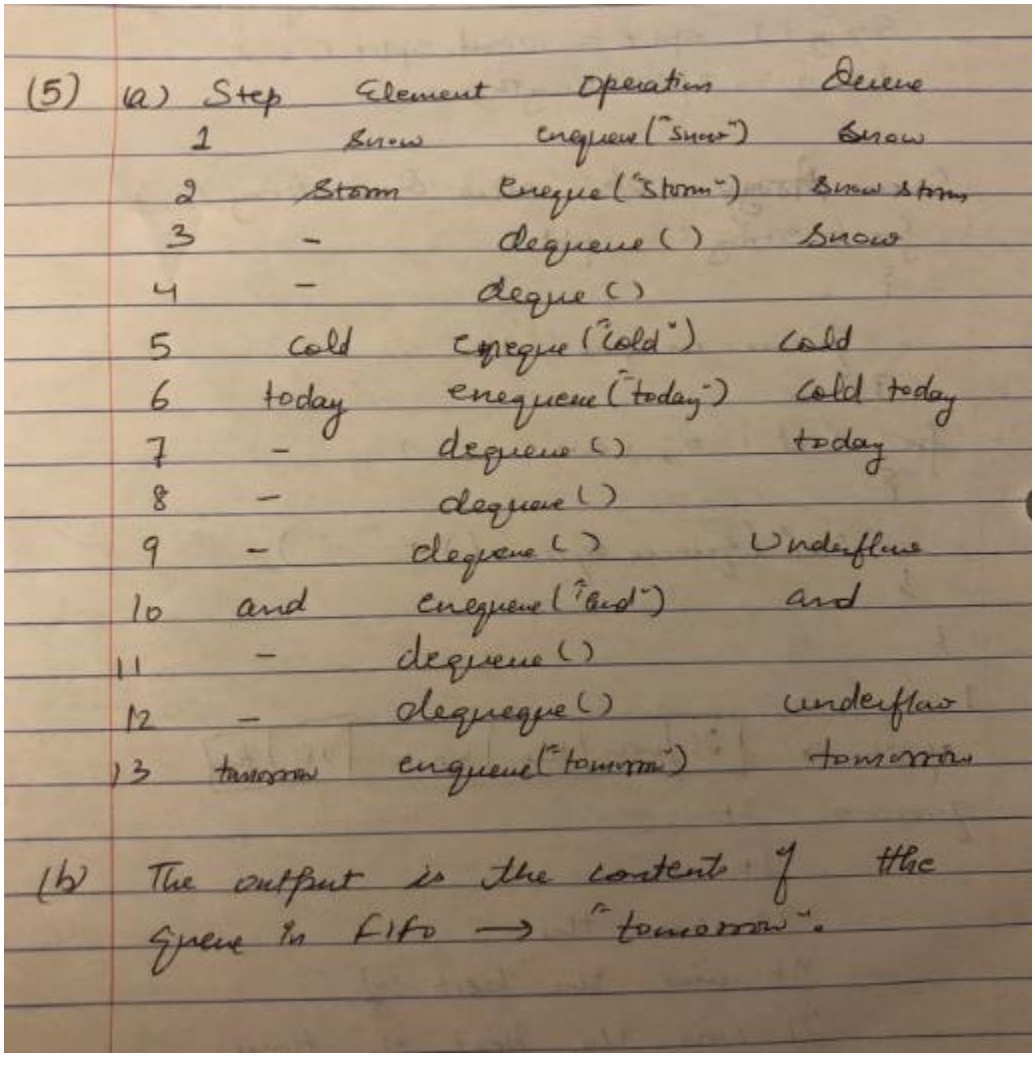
Print
It was the best of
It was the best of times

Qs5 Consider the following QueueOfStrings code to manage queue. The input to this method is String “Snow storm - - cold today - - - and - - tomorrow”

a) Show step-by-step of queue execution

b) What is the output

```
public static void main(String[] args) {
   QueueOfStrings q = new QueueOfStrings();
   while (!StdIn.isEmpty()) {
     String s = StdIn.readString();
     if (s.equals("-")) StdOut.print(q.dequeue());
     else               q.enqueue(s);
   }
}
```



(5) (a)

Step	Element	Operation	Queue
1	Snow	enqueue("Snow")	Snow
2	Storm	enqueue("Storm")	Snow Storm
3	-	dequeue()	Storm
4	-	deque()	
5	cold	enqueue("cold")	cold
6	today	enqueue("today")	cold today
7	-	dequeue()	today
8	-	deque()	
9	-	dequeue()	Underflow
10	and	enqueue("and")	and
11	-	dequeue()	
12	-	dequeue()	underflow
13	tomorrow	enqueue("tomorrow")	tomorrow

(b) The output is the content of the queue is FIFO → "tomorrow".

Qs6. Consider the following data:

	A	B	C	D
1	**ID**	**First Name**	**Last Name**	**Course**
2	1	Jack	Irwan	Software Engineering
3	2	Billy	Mckao	Requirement Engineering
4	3	Nat	Mcfaden	Multivariate Calculus
5	4	Steven	Shwimmer	Software Architecture
6	5	Ruby	jason	Relational DBMS
7	6	Mark	Dyne	PHP development
8	7	Philip	namdaf	Microsoft Dot Net Platform
9	8	Erik	Bawn	HTMl & Scripting
10	9	Ricky	ben	Data communication
11	10	Van	Miecky	Computer Networks
12				

Build **Queue** with LinkedList implementation and Array implementation:

a) Create file "input.txt" with this data
b) Read input.data into an an ArrayList.
c) Create Queue with LinkedList implementation
d) Write Node data structure of your input data
e) Queue must support all operations of queue: enqueue, dequeue, isEmpty, isFull
f) Write a Test program to test your linked implementation of Queue:
—enqueue all elements into queue
—dequeue 4 elements from queue
—enqueue all elements into queue
—dequeue all elements from queue
—dequeue 2 element
—enqueue all elements into queue
—enqueue this element into the queue:

11	John	Henry	"software development"
12	Raj	Manish	"Statistician"
13	Justin	Morgan	"engineering statistics"

—Print queue 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 Queue LinkedList time-complexity?
i) Repeat (a)—(g) with Queue fixed Array Implementation
j) what is Queue Fixed Array time-complexity?
k) What are the consequences of oversizing or undersizing fixed array size?'

Ans – (a) to (g) -> Java Files/Question6

h) For the operations of enqueue and dequeue the time complexity of LinkedList will be O(1). And for print and reverse print operation O(n). where n is number of elements in the Queue.

j) For fixed array time-complexity for enqueue and dequeue will be O(1). And for operation of print and reverse print it's time complexity will be O(n) where n is number of elements in Queue.

k) In case of undersizing, the array will be resized by double of it's capacity and in case of oversizing the array will be minimize to it's half capacity.

Oversizing – _We need to copy all the items to a new array. It will take time to copy the number of items from old array to new array and corrosponds to time-complexity of O(n).

```
public void push(Item item)
{
        // Add item to top of stack.
        if (N == a.length) resize(2*a.length);
        a[N++] = item;
}
private void resize(int max)
{
        // Move stack to a new array of size max.
        Item[] temp = (Item[]) new Object[max];
        for (int i = 0; i < N; i++)
                temp[i] = a[i];
                a = temp;
}
```

Undersizing – _Undersizing also requires O(n) time-complexity.

```
public Item pop()
{
        // Remove item from top of stack.                    i
                                                             f
        Item item = a[--N];
        a[N] = null;                                         (
                                                             N
```

```
        > 0 && N == a.length/4)
                resize(a.length/2);
        return item;
}
```

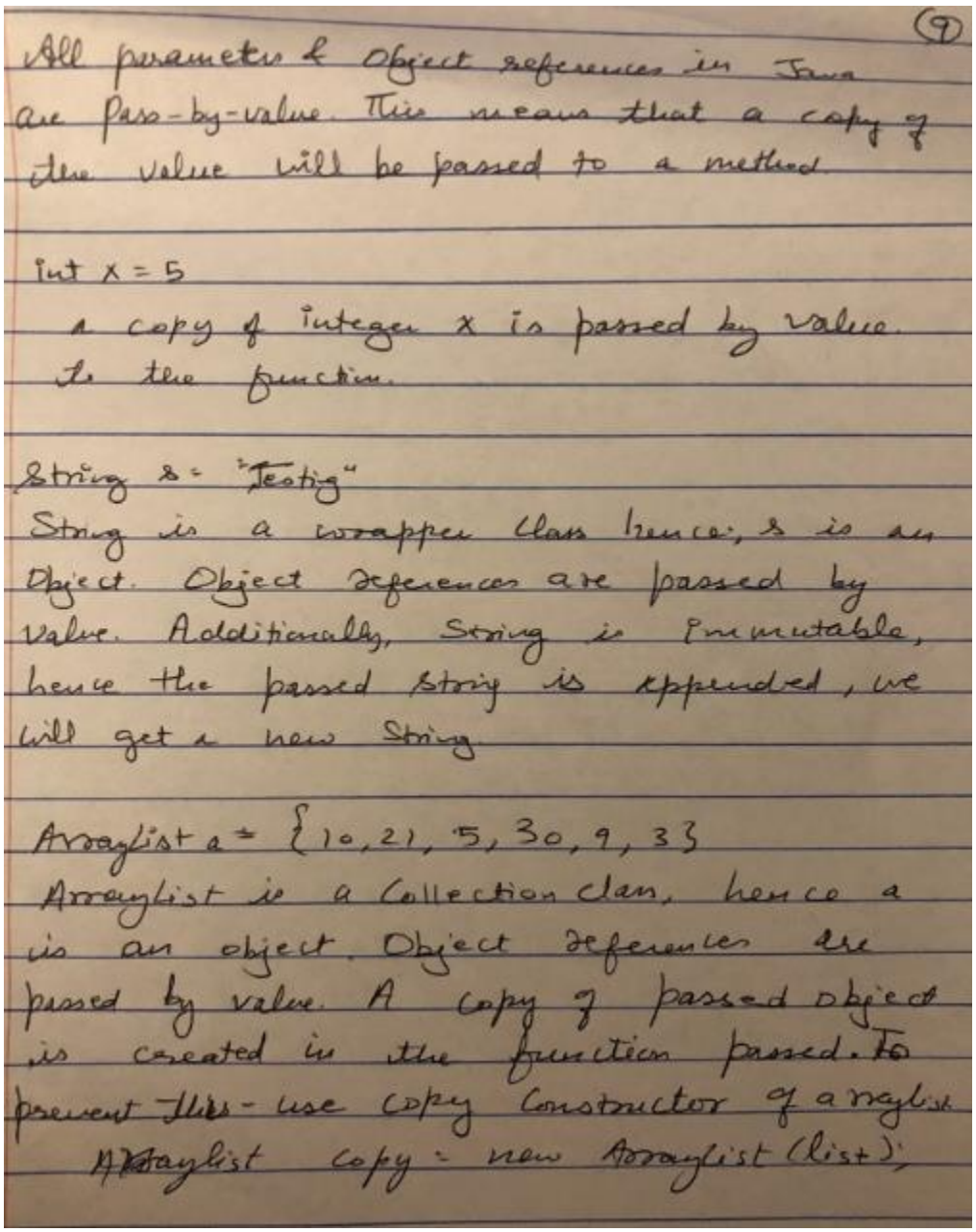
7. Consider signed byte X, and unsigned byte Y. What are the possible values for both X and Y?

Ans.

In JAVA, a signed byte takes 8-bit / 1 byte & so does unsigned, but the unsigned byte can store double the size of signed byte as the MSB is used to store value instead of sign. Range of signed byte is -128 to +127 and range of unsigned byte is 0 to 255

Qs8. Java is Pass-by-Value, what does that mean? How does it work with examples,

int X=5; String s="Testing"; ArrayList = {10, 21, 5, 30, 9, 3}



(9)

All parameters & Object references in Java are pass-by-value. This means that a copy of the value will be passed to a method.

int x = 5

a copy of integer x is passed by value to the function.

String s = "Testing"

String is a wrapper class hence, s is an Object. Object references are passed by value. Additionally, String is Immutable, hence the passed String is appended, we will get a new String.

Arraylist a = {10, 21, 5, 30, 9, 3}

Arraylist is a Collection class, hence a is an object. Object references are passed by value. A copy of passed object is created in the function passed. To prevent this - use copy Constructor of arraylist

Arraylist copy = new Arraylist (list);

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:

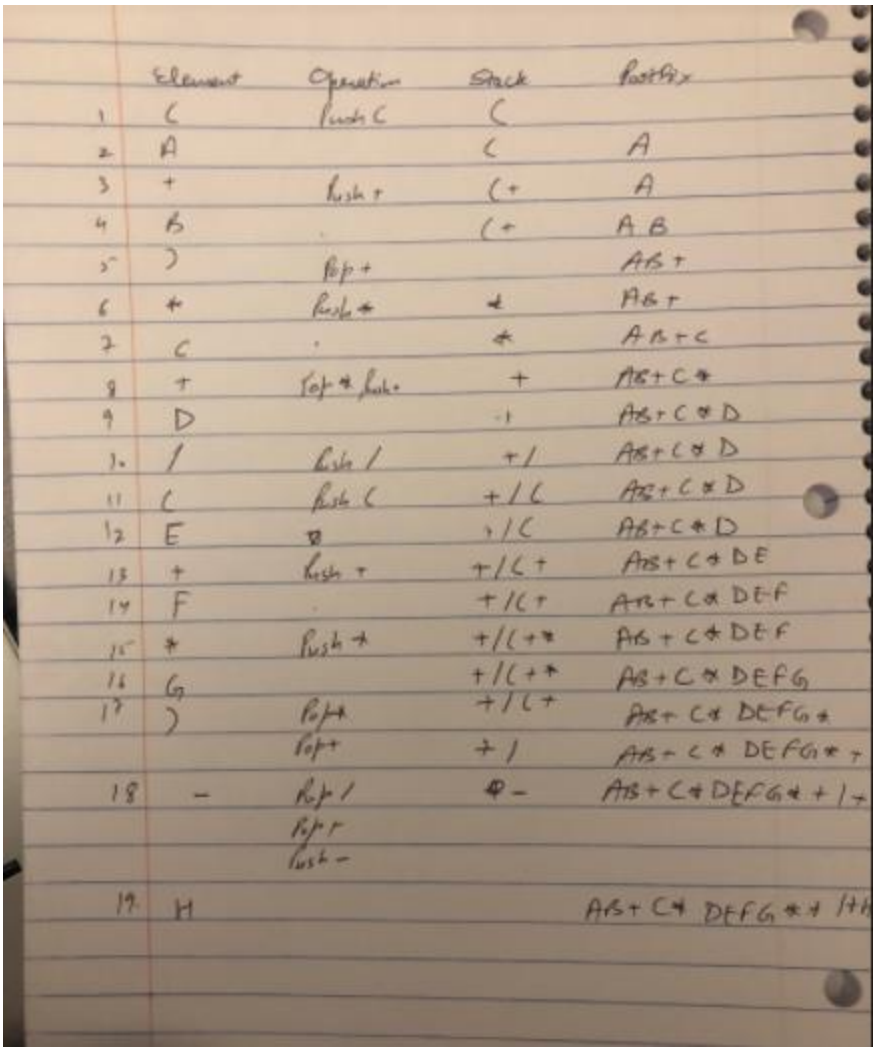
```
while there are more symbols to read
        read the next symbol
        case:
                operand -->  output it.
                   '('   -->  push it on the stack.
                   ')'   -->  pop operators from the stack to output
                              until a '(' is popped; do 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
```

Ans - JavaFiles/Question9

10. Consider this 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 then this operator, and then push (operator) into the stack – When the expression is ended, pop all the operators remain in the stack

A) Write Java code to transform this Infix expression to Postfix: (1 + 3 + ((4 / 2) * (8 * 4)))
B) Write Java code to Evaluate Postfix expression.

Ans – JavaFiles/Question10



	Element	Operation	Stack	Postfix
1	(	Push (	(	
2	A		(	A
3	+	Push +	(+	A
4	B		(+	A B
5	)	Pop +		A B +
6	*	Push *	*	A B +
7	C		*	A B + C
8	+	Pop * Push +	+	A B + C *
9	D		+1	A B + C * D
10	/	Push /	+/	A B + C * D
11	(	Push (	+/(	A B + C * D
12	E		+/(	A B + C * D
13	+	Push +	+/(+	A B + C * D E
14	F		+/(+	A B + C * D E F
15	*	Push *	+/(+*	A B + C * D E F
16	G		+/(+*	A B + C * D E F G
17	)	Pop *	+/(+	A B + C * D E F G *
		Pop +	+/	A B + C * D E F G * +
18	-	Pop /	-	A B + C * D E F G * + /
		Pop +		
		Push -		
19	H			A B + C * D E F G * * / H

10. Consider this 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 then this operator, and then push (operator) into the stack – When the expression is ended, pop all the operators remain in the stack

A) Write Java code to transform this Infix expression to Postfix: (1 + 3 + ((4 / 2) * (8 * 4)))
B) Write Java code to Evaluate Postfix expression.

Ans. JavaFiles/Question10