Q1

a)

```
push(c)
if stack is empty:
        insert c
        make max equal to c
if stack size is not zero:
        if c less than or equal to max:
                insert c
        if c larger than max:
                insert (2c – max)
                max = c
pop():
        if stack is empty:
                print “stack size is zero”
        if stack is not empty:
                remove element from top
                set removed element as e
                if e less than or equal to max:
                        do nothing
                if e larger than max:
                        set max = (2* max - e)
max():
        get max
```

b) The big(o) complexity of both method is O(1), since those methods function regardless of input sizes. They changed int max everytime push or pop method is used, and max method Is just getting the data of max from push and

c) Yes… the pseudocode is the same that I have given before.

```
        push(c)
    if stack is empty:
        insert c
        make max equal to c
    if stack size is not zero:
        if c less than or equal to max:
            insert c
        if c larger than max:
            insert 2c - max
            max = c
    pop():
        if stack is empty:
            print “stack size is zero”
        if stack is not empty:
            remove element from top
            set removed element as e
            if e less than or equal to max:
                do nothing
            if e larger than max:
                set max = (2* max - e)
    max():
        get max
```

q2)

The following solution is space and time efficient. It makes for a constant complexity and prevents an overflow by checking the space between the top elements of the two stacks. The code is made in such a way that the two stacks start from the two extremities of arr[]. One stack starts

from LHS pushing it's first element at index 0; and the other starts from RHS; pushing it's first element at index (n-1). Both of which grow and shrink dynamically in the opposite direction.

b. Code to implement Two stacks in single array.

```
class twoStacks{
        initialize int size, Top1, Top2, arr[];
        two Stacks constructor with parameter int n{
                arr=new int[n];
                size=n;
                Top1 =-1;
                Top2=size;
        }

make an int method firstPush(int x){
        if Top1 <Top2-1:
                Top1++;
                push element x into stack one (arr[Top1]=x)
        else:
                print  "Stack overflow"
                exit the program;
        }
make an int method secondPush(int x){
        if Top1 <Top2-1:
                Top2--;
        push element x into stack two (arr[Top2]=x);
        else:
```

```
                print: "Stack overflow";
                exit the program;
        }
make and int method firstPop(){
        if Top1 >= 0:
                int x = arr[Top1];
                -- Top1;
                return x;
        else
                print: "Stack underflow";
                exit the program;
        return 0;
        }
Function int secondPop(){ //to pop an element from stack2
        if Top2 < size:
                int x = arr[Top2];
                ++ Top2;
                return x;
        else
                print: "Stack underflow";
                exit the program;
        return 0;
    }
Function boolean isFull1(){
    if Top1<Top2-1:
            return false;
    return true;
```

```
}
Function boolean isFull2(){
        return isFull1();
    }
Function boolean isEmpty1(){
    if Top1==0:{
        return true;
    }return false;
}
Function boolean isEmpty2(){
    if Top2==size:{
        return true;
    }return false;
    }
}
```

c. Big-O complexity of of my methods push() ,pop(), isEmpty(), isFull() is O(1). Overall, it has a complexity of O(1)

d. Big-Ω complexity time complexities of static array and a simples linked list and operations push() ,pop(), isEmpty(), isFull() is Ω(1); this code has a time complexity of Ω(1)

e. We could, but it would not be time efficient although it can be more space efficient:

- initialize two stacks of the array at the beginning that grows in opposite directions.
- Initialize third stack as starting in the middle that grows in any direction.
- Making an adjustment to the push() operation, for the operation to overwrite the other stack, then shift the whole middle stack in the opposite direction before using the push() operation.

Q3

i) $f(n) = \log^3 n$; $g(n) = \sqrt{n} \log n$.

Solution:

Compare f(n) ; g(n)

log on both sides:

3logn ; (1⁄2)log(n^2)

Eliminating the constants:

log(n) <= log(n^2)

f(n)=O(g(n))

ii) $f(n) = n\sqrt{n} + \log n$; $g(n)=\log n^4$

Solution:

$n\sqrt{n}$ = n^(1+1⁄2)= n^(3/2)

Log on both sides:

(3/2)log(n) ; 4log(n)

Eliminating the constants:

log(n) = log(n)

f(n) is θ(g(n))

iii) $f(n) = 2n$; $g(n) = \log^2 n$.

Solution:

log on both sides:

log(n) ; 2log(n)

eliminating the constants :

log(n) = log(n)

f(n) = Ω(g(n))

iv) f(n) = $\sqrt{}$ n; g(n) = 2$\sqrt{}$ logn.

Solution:

log on both sides:

(1⁄2)log n ; ((log n)^(1⁄2))*log2

Eliminating the constant:

log(n) >= ((log n)^(1⁄2))

Therefore:

f(n) = Ω(g(n))

v) f(n) = 2^n ; g(n) = n^n

Solution:

log on both sides:

n log(2) ; n log(n)

Eliminate constants:

n <= n log(n)

f(n) = O(g(n))

f(n) is θ(g(n))

vi) f(n) = 50; g(n) = log 60

These are both constants, therefore:

f(n)=g(n)

f(n) = Ω(g(n))

Question 4

Develop a well-documented pseudo code that accepts an array of integers, A, of any size, then finds and removes all

duplicate values in the array. For instance, given an array A as shown below:

[22, 61,-10, 61, 10, 9, 9, 21, 35, 22,-10, 19, 5, 77, 5, 92, 85, 21, 35, 12, 9, 61]

your code should find and remove all duplicate values, resulting in the array looking as follows:

[22, 61,-10, 10, 9, 21, 35, 19, 5, 77, 92, 85, 35, 12]

Notice that this is just an example and your solution should be applied to any given array.

Additionally, you are only allowed to use Stacks, Queues, or Double-ended Queues (DQ) to design your solution, if

needed.

a. Briefly justify the motive(s) behind your solution design (choice of the algorithm and the data structure).

b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.

c. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.

d. What is the Big-O space complexity of your solution?

Solution:

```
public class Main{
        removeDuplicates(int arr[]): static int array method{
                unrepeated: initialize an integer stack to store unrepeated elements
                int numbers =0;
                for every int i in the arr{
                        if arr[i] exists in unrepeated.search:
                        unrepeated push arr[i]
                        unrepeated ++
                }
        }
        int result[]: a new array created for the unrepeated numbers resulted from
removeDuplicates
```

```
        for every int i in reverse order from the last index to index 0{
                pop()element from the stack and insert it in the array result
                set result[i]=unrepeated.pop()
        }
        return the resulting arr (return result)
}
main() method{
        define the array of integers int arr[] = {22, 61, -10, 61, 9, 9, 21, 35, 22, -10, 19, 5, 77, 5, 92, 85, 21, 35, 12, 9, 61};
        call the previously defined method removeDuplicates to remove the duplicates
        int result[]=removeDuplicates(arr))
        Print the original array
                for every int i in the array:
                        print (arr[i]+" ")
                        print the resulting array
                for every int i in the result:
                        Print (result[i]+" ")
}
```

a)

This code has a complexity of O(N) and it eliminates all the duplicated numbers by using stack and avoiding nested loops.

b)

A: The Big-O complexity of this solution is O(N) It is composed of single array & if-else statements of O(1) and the for loops dependent on input size which makes it O(N); O(1)*O(N)=O(N). Therefore, the Big-O complexity of the code is O(N).

C) What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.

A: The Big-Ω complexity is Ω(n) as well because it is the best case.

D) What is the Big-O space complexity of your solution?

A: The space complexity is O(n) because it creates an extra array of data based on input sizes.

Programming

Pseudo Code part 1

```
public static void doOp()
  {

    x = valStk.pop
    y = valStk.pop()
    op = opStk.pop().toString();
    int answer = EvalExp(x,y,op);

    valStk.push(answer);

  }

  public static int EvalExp(int y, int x, String op){
    int answer =0;
    switch(op){
      case "+":

        return x+y;
      case "^":
```

```
    return power of x and y
case "*":
    return x*y;
case "/":
    return x/y;
case "-":
    return x-y;
case ">":
    if(x>y){
        return 1;
    }
    return 0;
case ">=":
    if(x>=y){
        return 1;
    }
    return 0;
case "<":
    if(x<y){
        return 1;
    }
    return 0;
case "<=":
    if(x<=y){
        return 1;
    }
    return 0;
```

```
        case "≥":
            if(x>=y){
                return 1;
            }
            return 0;
        case "≤":
            if(x<=y){
                return 1;
            }
            return 0;
        case "==":
            if(x==y){
                return 1;
            }
            return 0;
        case "!=":
            if(x!=y){
                return 1;
            }
            return 0;
        default:
            return 0;
    }
}

public static void repeatOps (String refOp)
{
```

```
   while (valStk.size() > 1 && (prec(refOp) <= prec(opStk.peek().toString())))
   {
            doOp();
   }
}

public static int prec(String op)
{
   switch(op)
   {
      case "(":
      case ")":
         return 1;
      case "!":
         return 2;
      case "^":
         return 3;
      case "*":
      case "/":
         return 4;
      case "+":
      case "-":
         return 5;
      case ">":
      case ">=":
      case "<":
      case "<=":
```

```
        case "≥":
        case "≤":
            return 6;
        case "==":
        case "!=":
            return 7;
        case "$":
            return 8;
        default:
            throw new IllegalArgumentException("Invalid Operator!");
    }
}
public static void main(String[] args) {
    try
    {
            Initialize scanner and fileinputStream to the file that you want to read, sc
            Initialize printwriter to the file you want to output, pw
            while(!line.equals("$"))
            {
                while(line.trim().equals("")){
                    sc.nextLine();
                }
                pw.println("Expression: " + line);
                line = line replace all spaces
                initialize array of char tokens to line.toCharArray();
                for(int i = 0; i < tokens.length; i++)
                {
```

```
if(tokens[i]=='-' && i==0){
   initialize sb as StringBuilder();
   sb.append(tokens[i++]);
   while(i <tokens.length && Character.isDigit(tokens[i])){
      sb.append(tokens[i]);
      i++;
   }
   valStk.push(Integer.parseInt(sb.toString()));
 }

if(i>0){
   if(!Character.isDigit(tokens[i-1]) && tokens[i]=='-'){
      sb = new StringBuilder();

      sb.append(tokens[i]);
      System.out.println(tokens[i]);
      i++;
      while(i <tokens.length && Character.isDigit(tokens[i])){
         System.out.println(tokens[i]);
         sb.append(Character.toString(tokens[i]));
         i++;
         System.out.println("Found Negative Number");
      }
      System.out.println(sb.toString());
      valStk.push(Integer.parseInt(sb.toString()));
   }
```

```
}
if( tokens[i] is digit)
{

   sb = new StringBuilder();

      while(i <tokens.length && Character.isDigit(tokens[i])){

         sb.append(Character.toString(tokens[i]));
         i++;

      }
   valStk.push(Integer.parseInt(sb.toString()));

}



if(tokens[i]=='(')
{

   opStk.push(Character.toString(tokens[i]));
}
else if(tokens[i]==')')
{

   while(!opStk.peek().toString().equals("(")){
      doOp();
```

```
    }
    opStk.pop();
}
else if(tokens[i]=='!')
{
    if(i+1>tokens.length){
        int num =((Number) valStk.pop()).intValue();;
        valStk.push(factorial(num));
    }else{
        if(tokens[i+1]=='='){
            sb = new StringBuilder();
            sb.append(tokens[i]);
            sb.append(line.charAt(i + 1));
            i++;
            repeatOps(sb.toString());
            opStk.push(Character.toString(tokens[i]));
        }else{
            int num =((Number) valStk.pop()).intValue();;
            valStk.push(factorial(num));
        }
    }

}
else
{
```

```
                    if((tokens[i] == '>' || tokens[i] == '<'  || tokens[i] == '=') && (line.charAt(i
+ 1) == '='))
                    {

                       sb = new StringBuilder();
                       sb.append(tokens[i]);
                       sb.append(line.charAt(i + 1));
                       i++;
                       repeatOps(sb.toString());
                       opStk.push(Character.toString(tokens[i]));

                    }
                    else
                    {
                       //if the non digit number reach and yet it is the first character


                       System.out.println("Found Operation");
                       repeatOps(Character.toString(tokens[i]));
                       opStk.push(Character.toString(tokens[i]));




                    }
                 }
```

```
                }
                While opStk  !empty{
                   doOp();
                }
                pw.println(valStk.peek());
                line = sc.nextLine();
                opStk.removeAllElements();
                valStk.removeAllElements();
             }
             pw.close();
      }

   }
      static int factorial(int n)
{

      if (n == 0)
         return 1;
      else
         return(n * factorial(n-1));
      }
}
```

Part 2

```
public static void main(String[]args){

      System.out.println(evaluate("((1+-2+3!-23)+2)!=23"));



         Initialize scanner inputStream to
         PrintWriter outputStream=new PrintWriter(new FileOutputStream("out2.txt"));
         While inputStream.hasNextLine{
            Initialize String str to inputStream.nextLine() and replaceAll spaces to no space
            Print str on outputStream
            outputStream print evaluate(str);
            inputStream.nextLine();
         }



   }


   public static int evaluate(String s) {


      if (!s.contains("+") && !s.contains("-") && !s.contains("*") && !s.contains("/")
&& !s.contains("!")&& !s.contains("==")&& !s.contains("<")&& !s.contains("≥")&& !s.conta
ins("≤")
            && !s.contains(")")&& !s.contains("(")) {
         return Integer.parseInt(s);
      }
      if(s.charAt(0)=='-' ){
         for(int index=1; index<s.length();index++){
            if(!Character.isDigit(s.charAt(index))){
```

```
            break;
        }
        if(index==s.length()-1){
            return Integer.parseInt(s);
        }
      }

  }

int i, countF, result=0, b=0;
initialize StringBuilder sb, rebuilder;


  for(i=0;i<s.length();i++)
  {
     If s.charAt(i) equals to '('{
        sb=new StringBuilder();
        rebuilder= new StringBuilder(s);
        int j =i;
        rebuilder.deleteCharAt(j);
        i++;
        b++;
        while(s.charAt(i)!=')' || b>0){
           rebuilder.deleteCharAt(j);
           if(s.charAt(i)=='('){
              System.out.println(s.charAt(i));
              b++;
```

```
        }
        sb.append(s.charAt(i++));
        if s.charAt(i) equals to ')' {
            b--;
        }
    }
    Initialize String results to evaluate(sb)
    Rebuilder delete each character that were going to be evaluate
    Set String s=results+rebuilder.toString();
    Set i to 0 so it can Iterate over again;

}
if(s.charAt(i) == '!'){
    if(i+1>s.length()-1){
        initialize rebuilder = new StringBuilder(s);
        rebuilder.deleteCharAt(i);
        i--;
        sb=new StringBuilder();
        while s.charAt(i) is digit{
            rebuilder.deleteCharAt(i);
            sb.append(s.charAt(i--));
            if(i<=0){
                break;
            }
        }
        Initialize String temp to factorial of sb to int
        if(rebuilder.toString().equals("")){
```

```
        return Integer.parseInt(temp);
    }
    s= rebuilder.insert(0, temp).toString();
    i=0;
}else{
    if(s.charAt(i+1)!='='){
        rebuilder = new StringBuilder(s);
        System.out.println("rebuilder original "+ rebuilder);
        rebuilder.deleteCharAt(i);
        i--;
        sb=new StringBuilder();
        while s.charAt[i] is digit{
            rebuilder.deleteCharAt(i);
            sb.append(s.charAt(i--));
            if(i<=0){
                break;
            }
        }
        Initialize String temp to
        System.out.println("Factorial: "+temp);
        if(rebuilder.toString().equals("")){
            return Integer.parseInt(temp);
        }
        s= rebuilder.insert(0, temp).toString();
        System.out.println("S after factorial: "+ s);
        i=0;
        }else{
```

```
            rebuilder= new StringBuilder(s);
            rebuilder.deleteCharAt(i+1);
            s=rebuilder.toString();
            break;
          }
    }
}
if(s.charAt(i) == '^'){
    break;
}
if(s.charAt(i) == '*' || s.charAt(i) == '/'){
    break;
}
if(s.charAt(i) == '+' || s.charAt(i) == '-'){
    if(i-1<0 && s.charAt(i)=='-'){

    }else{
        break;
    }
}
if(s.charAt(i) == '>'|| s.charAt(i) == '<' || s.charAt(i)=='≤' || s.charAt(i)=='≥'){
    if(s.charAt(i+1)=='=' && s.charAt(i)=='<')
    {
        rebuilder = new StringBuilder(s);
        rebuilder.deleteCharAt(i);
        rebuilder.deleteCharAt(i);
        rebuilder.insert(i, '≤');
```

```
            s=rebuilder.toString();

        }else if(s.charAt(i+1)=='=' && s.charAt(i)=='>'){
            rebuilder = new StringBuilder(s);
            rebuilder.deleteCharAt(i);
            rebuilder.deleteCharAt(i);
            insert  '≥' at index I to rebuilder
            s=rebuilder.toString();
        }

        break;




    }
    if(s.charAt(i) == '='){
        rebuilder= new StringBuilder(s);
        rebuilder.deleteCharAt(i);
        s=rebuilder.toString();
        break;
    }



  }
  System.out.println(s);
String r1 = s.substring(0, i);
```

```
System.out.println("ri: "+r1);
System.out.println(i);
String r2 = s.substring(i + 1, s.length());
System.out.println(r2);



switch (s.charAt(i)) {
   case '+':
      result = evaluate(r1) + evaluate(r2);
      break;
   case '-':
      result = evaluate(r1) - evaluate(r2);
      break;
   case '*':
      result = evaluate(r1) * evaluate(r2);
      break;
   case '/':
      int right = evaluate(r2);
      if (right == 0) //if denominator is zero
      {
         System.out.println("Invalid divisor");
         System.exit(1);
      } else {
         result = (evaluate(r1) / right);
      }
      break;
   case '^':
```

```
    result =(int) Math.pow(evaluate(r1), evaluate(r2));
    break;
case '=':
    if(evaluate(r1)==evaluate(r2)){
        result =1;
    }else{
        result =0;
    }
    break;
case '≥':
    if(evaluate(r1)>=evaluate(r2)){
        result =1;
    }else{
        result =0;
    }
    break;
case '≤':
    if(evaluate(r1)<=evaluate(r2)){
        result =1;
    }else{
        result =0;
    }
    break;
case '>':
    if(evaluate(r1)>=evaluate(r2)){
        result =1;
    }else{
```

```
            result =0;
        }
        break;
    case '<':
        if(evaluate(r1)<evaluate(r2)){
            result =1;
        }else{
            result =0;
        }
        break;
    case '!':
        if(evaluate(r1)!=evaluate(r2)){
            result =1;
        }else{
            result =0;
        }
        break;
  }
  return result;

}

  static int factorial(int n)
    {
    if (n == 0)
        return 1;
    else
```

```
            return(n * factorial(n-1));
        }
    }
```

a) Briefly explain the time and memory complexity for both versions of your calculator.

Time complexity for first version of calculator that uses stacks has time complexity of $O(n^2)$. This is because it has to iterate through the input and expecting the worst-case scenario where every expression involves using factorial which has time complexity of O(n) which ultimately leads up to $O(n^2)$. While the space complexity is $O(n^2)$ as well since one stack store operand and one stack store operator.

In second version, the time complexity is exponential which is $O(2^n)$ This is because the method evaluates whole string depending on the number of operators. If there are four operators then we will evaluate more than four times and each time we evaluate the result plus the operands and operators. While each evaluate has to pass through the for loop which means the time complexity to be T(n)*T(n-1)…. So on and so forth. The space complexity is O(n).

b) For the second version of your calculator describe the type of recursion used in your implementation.

This is direct recursion. Indirect recursion might have much higher complexity and therefore is better to use direct recursion.

c) Provide test logs for at least 20 different and sufficiently complex arithmetic expressions that use all types of operators (including parentheses) in varying combinations.

Expression: 1+2

3

Expression: 2-1

1

Expression: 4*4

16

Expression: 5/3

1

Expression: 3+(3*2)

9

Expression: -3+2

-1

Expression: 5^2

25

Expression: 21+23

44

Expression: -20+5

-15

Expression: 15/5

3

Expression: 9!

9

Expression: 3+2 + 3!

8

Expression: 3+(3-4)

2

Expression: (2+1)!

3

Expression: 4-3

1

Expression: 3!=3

0

Expression: (5*5)== 2

0

Expression: (12+12)== (5/5)

0

Expression: 5>2

1

Expression: (2+4+-2)<2

0