Lab-7

Name :- Vegad Rutvik

Student Id :- 202201143

Course :- Software Engineering(IT314)

- **PROGRAM INSPECTION:**

1. How many errors are there in the program? Mention the errors you have identified.
    a. Data Reference Errors:
        i. Uninitialized Variables:

```
T* allocate() {

T* tmp = mHead; if (!tmp) {
tmp = performAllocation();

}

mHead = *
reinterpret_cast_no_cast_align_
warning
<T**>(tmp); return tmp;
}
```

mHead and mListForFree (Line 418): These variables are set to nullptr but are not consistently reset after deallocating memory, which could result in dangling pointers or accessing uninitialized memory.

ii. **Array Bound Violations:**

shiftUp and shiftDown operations: There are no validations to confirm that the index stays within the valid range of the array, which could result in out-of-bounds access.

i. **Dangling Pointers:**
In the BulkPoolAllocator, the reset() method releases memory but fails to set the pointer to nullptr, which may result in dangling pointer issues.

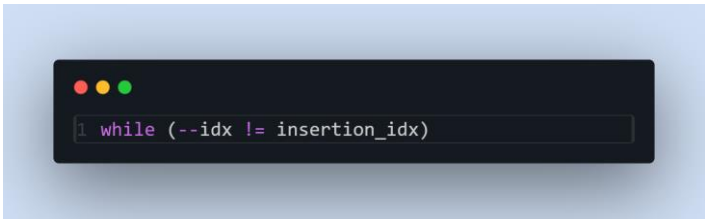
b. Data-Declaration Errors:
   i. **Potential Data Type Mismatches:**
   In hash_bytes, there are several type conversions during hashing operations. If there are discrepancies in size or attributes between the data types, it could lead to unpredictable behavior.

   ii. **Similar Variable Names:**
   Variables such as mHead, mListForFree, and mKeyVals have names that are quite similar, potentially causing confusion during code modifications or debugging efforts.

c. Computation Errors:
   i. **Integer Overflow:**
   In hash_bytes, hash computations involve several shifts and multiplications on large integers. If the values grow beyond the allowable range, this could lead to overflow issues.

   ii. **Off-by-One Errors**:

The loop conditions in shiftUp and shiftDown can cause off-by-one mistakes, particularly when the size of the data structure is not handled correctly.



```
while (--idx != insertion_idx)
```

d. Comparison Errors:

   i. Incorrect Boolean Comparisons:

   In findIdx, the combination of multiple logical operations may not be handled correctly, particularly with && and ||, which could result in inaccurate evaluations.

```
if (info == mInfo[idx] &&

ROBIN_HOOD_LIKELY(WKeyEqual::operator()(key,
mKeyVals[idx].getFirst())) {

return idx;

}
```

   ii. Mixed Comparisons:

   There are instances where different types, such as signed and unsigned integers, are compared. This may produce incorrect results based on the system or compiler being used.

e. Control-Flow Errors:

   i. Potential Infinite Loop:

   Unterminated Loops: In functions such as shiftUp and shiftDown, there is a risk that the loop may not terminate properly if the termination condition is never satisfied.

f. Interface Errors:

   i. Mismatched Parameter Attributes:

   Function Calls: There is a possibility of parameter mismatches in functions like insert_move, where the arguments provided may not align with the expected attributes (e.g., data type or size).

```
insert_move(std::move(oldKeyVals[i]));
```

ii. Global Variables:
Usage of Global Variables Across Functions: When the same global variable is accessed in various functions or procedures, it is crucial to ensure that it is used consistently and initialized correctly. While this may not be immediately evident, it could become a source of errors as the code expands.

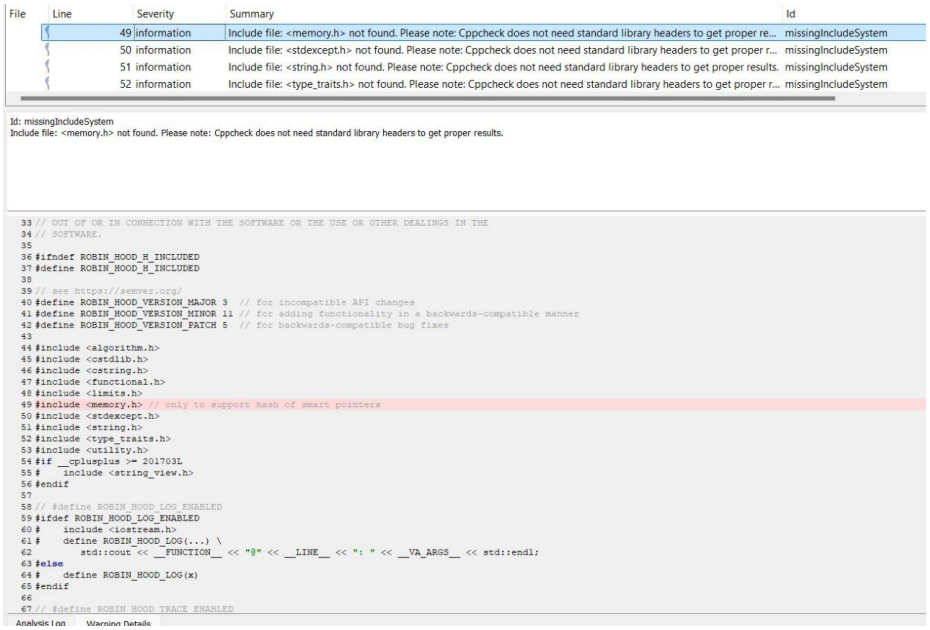
g. Input/Output Errors:
i. Missing File Handling:
Although the code does not directly interact with files, any extensions that involve input/output operations may lead to common file handling issues, such as unclosed files, failure to check for end-of-file conditions, or inadequate error handling.

2. Which category of program inspection would you find more effective?
a. Category A: Data Reference Errors is the most pertinent due to the dependence on manual memory management, pointers, and dynamic data structures. Errors in pointer dereferencing and memory management, such as incorrect allocation or deallocation, can lead to significant issues like crashes, segmentation faults, or memory leaks. Thus, addressing this category is essential. Additionally, Computation Errors and Control-Flow Errors should also be prioritized, especially in larger projects.

3. Which type of error are you unable to identify using program inspection?
    a. Concurrency Issues: The inspection process does not account for problems arising from multi-threading or concurrency, such as race conditions or deadlocks. If the program were to be expanded to incorporate multi-threading, considerations around shared resources, locks, and thread safety would become necessary.
    b. Dynamic Errors: Certain errors, including those related to memory overflow, underflow, or behaviors arising from the runtime environment, might not be detected until the code is executed in real-world conditions.

4. Is the program inspection technique worth applying?
    a. Yes, the program inspection technique is beneficial, particularly for identifying static errors that compilers might overlook, such as pointer mismanagement, array boundary violations, and incorrect control flow. Although it may not detect every dynamic issue or concurrency-related bug, it is a critical step in ensuring code quality, especially in memory-sensitive applications like this C++ implementation of hash tables.

- Static analysis tool

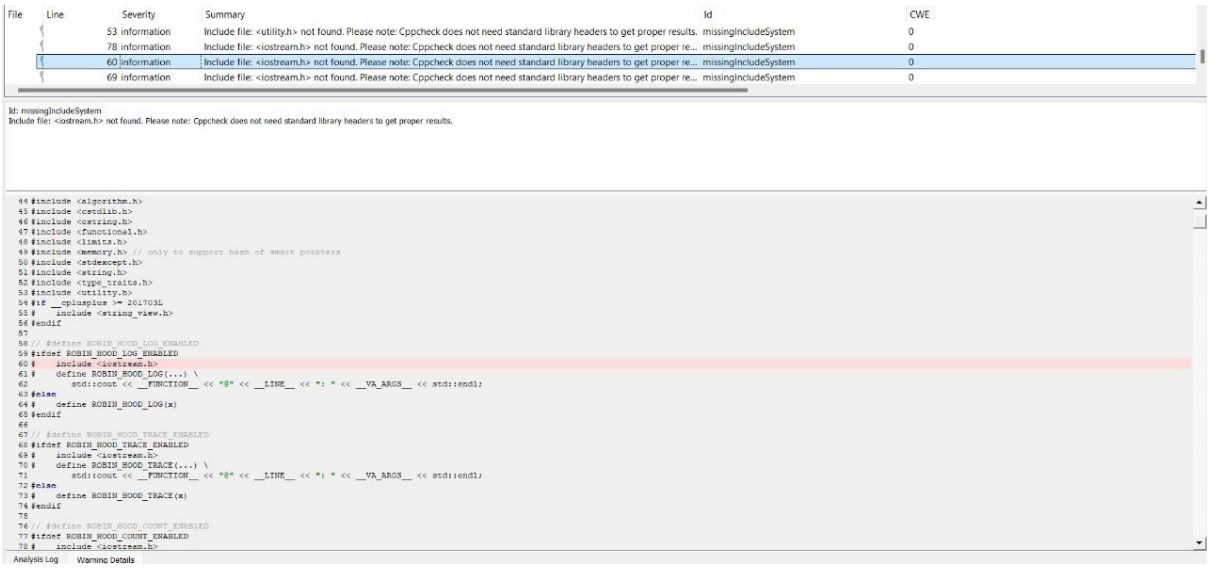


File	Line	Severity	Summary	Id
	49	information	Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem
	50	information	Include file: <stdexcept.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem

Id: missingIncludeSystem
Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
33 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
34 // SOFTWARE.
35
36 #ifndef ROBIN_HOOD_H_INCLUDED
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
```

Analysis Log | Warning Details

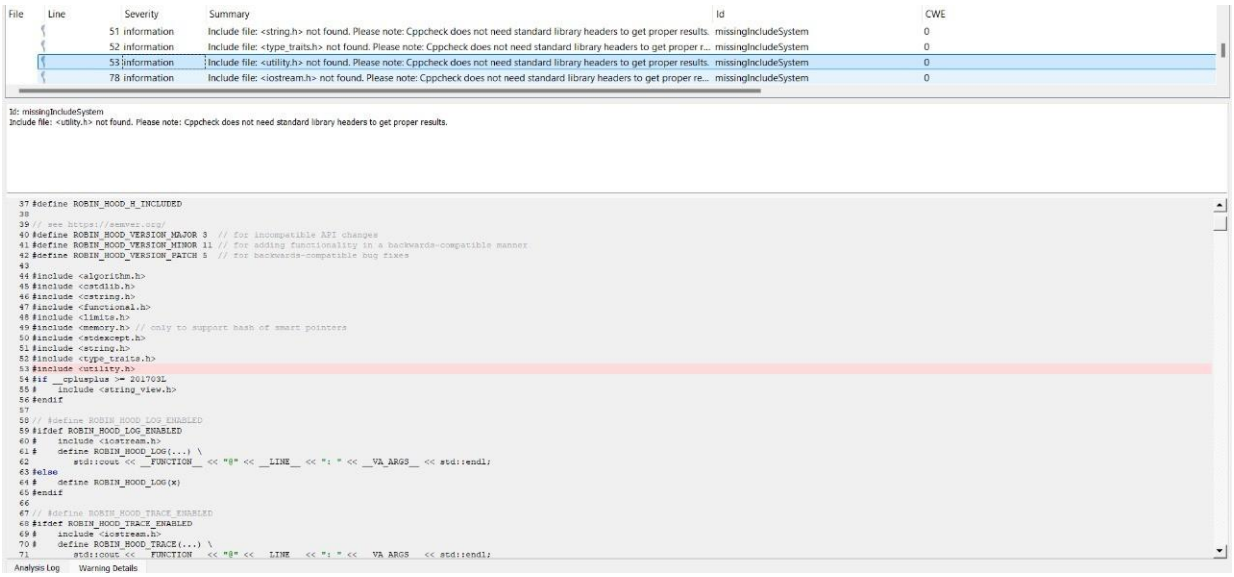


File	Line	Severity	Summary	Id	CWE
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	60	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	69	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
72 #else
73 #    define ROBIN_HOOD_TRACE(x)
74 #endif
75
76 // #define ROBIN_HOOD_COUNT_ENABLED
77 #ifdef ROBIN_HOOD_COUNT_ENABLED
78 #    include <iostream.h>
```

Analysis Log | Warning Details



File	Line	Severity	Summary	Id	CWE
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem	0
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0


```
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
```

Analysis Log Warning Details

Section 2:

Armstrong Number: Errors and Fixes

1. How many errors are there in the program?
There are two errors identified in the program.

2. How many breakpoints are needed to fix those errors?
We require two breakpoints to address these errors.

Steps Taken to Fix the Errors:

- **Error 1:** The operations for division and modulus are incorrectly swapped within the while loop.
  Fix: Adjust the code to ensure that the modulus operation retrieves the last digit, while the division operation appropriately reduces the number for subsequent iterations.
- **Error 2:** The variable used for checking is not accumulating values correctly.
  Fix: Revise the logic to guarantee that this variable accurately reflects the sum of each digit raised to the power of the total number of digits in the original number.

```
class Armstrong {
public static void main(String args[]) {
int num = Integer.parseInt(args[0]);
int n = num; // use to check at last time
int check = 0, remainder;
while (num > 0) {
    remainder = num % 10;
check = check + (int)Math.pow(remainder,
3);
num = num / 10;
}
if (check == n)
System.out.println(n + " is an Armstrong
Number");
else
System.out.println(n + " is not an
Armstrong Number");
}
}
```

GCD and LCM: Errors and Fixes

1. How many errors are there in the program?
There is one error identified in the program.

2. How many breakpoints are needed to fix this error?
We need one breakpoint to address this error.

Steps Taken to Fix the Error:

- **Error:** The condition in the while loop of the GCD method is incorrect.
  Fix: Modify the condition to while (a % b != 0) instead of while (a % b == 0). This change ensures that the loop continues until the remainder is zero, thereby correctly calculating the GCD.

```
import java.util.Scanner;
public class GCD_LCM {
static int gcd(int x, int y) {
int r = 0, a, b;
a = (x > y) ? x : y; // a is greater number
b = (x < y) ? x : y; // b is smaller number
r = b;
while (a % b != 0) {
r = a % b;
a = b;
b = r;
}
return r;
}
static int lcm(int x, int y) {
int a;
a = (x > y) ? x : y; // a is greater number
while (true) {
if (a % x == 0 && a % y == 0)
return a;
++a;
}
}
public static void main(String args[]) {
Scanner input = new Scanner(System.in);
```

```
System.out.println("Enter the two numbers: ");
int x = input.nextInt();
int y = input.nextInt();
System.out.println("The GCD of two numbers is:
" + gcd(x, y));
System.out.println("The LCM of two numbers is:
" + lcm(x, y));
input.close();
}
}
```

Knapsack Problem: Errors and Fixes

1. How many errors are there in the program?
There are three errors identified in the program.

2. How many breakpoints are needed to fix these errors?
We need two breakpoints to address these errors.

Steps Taken to Fix the Errors:

- **Error:** In the "take item n" case, the condition is incorrect.
  Fix: Change if (weight[n] > w) to if (weight[n] <= w) to ensure the profit is calculated when the item can be included.
- **Error:** The profit calculation is incorrect.
  Fix: Change profit[n-2] to profit[n] to ensure the correct profit value is used.
- **Error:** In the "don't take item n" case, the indexing is incorrect.
  Fix: Change opt[n++][w] to opt[n-1][w] to properly index the item

```
public class Knapsack {
    public static void main(String[] args) {
    int N = Integer.parseInt(args[0]); // numberof items
    int W = Integer.parseInt(args[1]); // maximum
    weight of knapsack
    int[] profit = new int[N+1];
    int[] weight = new int[N+1];
    // generate random instance, items 1..N
    for (int n = 1; n <= N; n++) {
```

```
    profit[n] = (int) (Math.random() * 1000);
    weight[n] = (int) (Math.random() * W);
    }
    // opt[n][w] = max profit of packing items 1..n
    with weight limit w
    // sol[n][w] = does opt solution to pack items
    1..n with weight limit w include item n?
    int[][] opt = new int[N+1][W+1];
    boolean[][] sol = new boolean[N+1][W+1];
    for (int n = 1; n <= N; n++) {
    for (int w = 1; w <= W; w++) {
    // don't take item n
    int option1 = opt[n-1][w];
    // take item n
    int option2 = Integer.MIN_VALUE;
    if (weight[n] <= w) option2 = profit[n]
    + opt[n-1][w-weight[n]];
    // select better of two options
    opt[n][w] = Math.max(option1, option2);
    sol[n][w] = (option2 > option1)
    }
    }
    // determine which items to take
    boolean[] take = new boolean[N+1];
    for (int n = N, w = W; n > 0; n--) {
    if (sol[n][w]) { take[n] = true; w = w -
    weight[n]; }
    else { take[n] = false;
    }
    }
    // print results
    System.out.println("item" + "\t" + "profit" +
    "\t" + "weight" + "\t" + "take");
    for (int n = 1; n <= N; n++) {
```

```
        System.out.println(n + "\t" + profit[n] +
        "\t" + weight[n] + "\t" + take[n]);
        }
        }
        }
```

Magic Number Check: Errors and Fixes

1. How many errors are there in the program?
There are three errors identified in the program.

2. How many breakpoints are needed to fix these errors?
We need one breakpoint to address these errors.

Steps Taken to Fix the Errors:

- **Error:** The condition in the inner while loop is incorrect.
  Fix: Change while(sum == 0) to while(sum != 0) to ensure that the loop processes digits correctly.
- **Error:** The calculation of s in the inner loop is incorrect.
  Fix: Change s = s * (sum / 10) to s = s + (sum % 10) to correctly sum the digits.
- **Error:** The order of operations in the inner while loop is incorrect.
  Fix: Reorder the operations to s = s + (sum % 10); sum = sum / 10; to correctly accumulate the digit sum.

```
// Program to check if number is Magic number in JAVA
import java.util.*;
public class MagicNumberCheck
{
    public static void main(String args[])
    {
        Scanner ob=new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n=ob.nextInt();
        int sum=0,num=n;
        while(num>9)
        {
            sum=num;int s=0;
            while(sum==0)
```

```
            {
                s=s*(sum/10);
                sum=sum%10
            }
            num=s;
        }
        if(num==1)
        {
            System.out.println(n+" is a Magic Number.");
        }
        else
        {
            System.out.println(n+" is not a Magic Number.");
        }
    }
}

```

Merge Sort: Errors and Fixes

1. How many errors are there in the program?
There are three errors identified in the program.

2. How many breakpoints are needed to fix these errors?
We need two breakpoints to address these errors.

Steps Taken to Fix the Errors:

- **Error:** Incorrect array indexing when splitting the array in mergeSort.
  Fix: Change int[] left = leftHalf(array + 1) to int[] left = leftHalf(array) and int[] right = rightHalf(array - 1) to int[] right = rightHalf(array) to pass the array correctly.
- **Error:** Incorrect increment and decrement in merge.
  Fix: Remove the ++ and -- from merge(array, left++, right--) and instead use merge(array, left, right) to pass the arrays directly.
- **Error:** The array access in the merge function is incorrectly accessing beyond the array bounds.
  Fix: Ensure the array boundaries are respected by adjusting the indexing in the merging logic.

```java
// This program implements the merge sort algorithm for
// arrays of integers.

import java.util.*;

public class MergeSort {
    public static void main(String[] args) {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " + Arrays.toString(list));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    // post: array is in sorted (nondecreasing) order
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            // split array into two halves
            int mid = array.length / 2; // Find the midpoint
            int[] left = Arrays.copyOfRange(array, 0, mid); // Get the left
half
            int[] right = Arrays.copyOfRange(array, mid, array.length); //
Get the right half

            // recursively sort the two halves
            mergeSort(left);
            mergeSort(right);

            // merge the sorted halves into a sorted whole
            merge(array, left, right);
        }
    }

```

```
    // Merges the given left and right arrays into the given
    // result array.
    // pre : result is empty; left/right are sorted
    // post: result contains result of merging sorted lists;
    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;   // index into left array
        int i2 = 0;   // index into right array

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <=
right[i2])) {
                result[i] = left[i1];    // take from left
                i1++;
            } else {
                result[i] = right[i2];   // take from right
                i2++;
            }
        }
    }
}

```

Matrix Multiplication: Errors and Fixes

1. **How many errors are there in the program?**
   There is **1 error** in the program.
2. **How many breakpoints do you need to fix this error?**
   We need **1 breakpoint** to fix this error.
3. **Steps Taken to Fix the Error:**
   - **Error:** Incorrect array indexing in the matrix multiplication logic.
   - **Fix:** Change first[c-1][c-k] and second[k-1][k-d] to first[c][k] and second[k][d]. These changes ensure that matrix elements are correctly referenced during multiplication.

```
4. // Java program to multiply two matrices
```

```
5. import java.util.Scanner;
6.
7. class MatrixMultiplication {
8.     public static void main(String args[]) {
9.         int m, n, p, q, sum = 0, c, d, k;
10.
11.          Scanner in = new Scanner(System.in);
12.          System.out.println("Enter the number of rows and columns of
   first matrix");
13.          m = in.nextInt();
14.          n = in.nextInt();
15.
16.          int first[][] = new int[m][n];
17.
18.          System.out.println("Enter the elements of first matrix");
19.
20.          for (c = 0; c < m; c++)
21.              for (d = 0; d < n; d++)
22.                  first[c][d] = in.nextInt();
23.
24.          System.out.println("Enter the number of rows and columns of
   second matrix");
25.          p = in.nextInt();
26.          q = in.nextInt();
27.
28.          if (n != p)
29.              System.out.println("Matrices with entered orders can't
   be multiplied with each other.");
30.          else {
31.              int second[][] = new int[p][q];
32.              int multiply[][] = new int[m][q];
33.
34.              System.out.println("Enter the elements of second
   matrix");
```

```
35.
36.             for (c = 0; c < p; c++)
37.                 for (d = 0; d < q; d++)
38.                     second[c][d] = in.nextInt();
39.
40.             for (c = 0; c < m; c++) {
41.                 for (d = 0; d < q; d++) {
42.                     for (k = 0; k < n; k++) { // Change p to n for
   correct iteration
43.                         sum = sum + first[c][k] * second[k][d]; //
   Corrected indexing
44.                     }
45.
46.                     multiply[c][d] = sum;
47.                     sum = 0;
48.                 }
49.             }
50.
51.             System.out.println("Product of entered matrices:-");
52.
53.             for (c = 0; c < m; c++) {
54.                 for (d = 0; d < q; d++)
55.                     System.out.print(multiply[c][d] + "\t");
56.
57.                 System.out.print("\n");
58.             }
59.         }
60.
61.         in.close(); // Close the scanner
62.     }
63. }
64.
```

Quadratic Probing Hash Table: Errors and Fixes

1. **How many errors are there in the program?**
    - There is **1 error** in the program.
2. **How many breakpoints do you need to fix this error?**
    - We need **1 breakpoint** to fix this error.
3. **Steps Taken to Fix the Error:**
    - **Error:** In the insert method, the line i += (i + h / h--) % maxSize; is incorrect.
    - **Fix:** The correct logic should be i = (i + h * h++) % maxSize; to correctly implement quadratic probing

```
/**
 * Java Program to implement Quadratic Probing Hash Table
 */

import java.util.Scanner;

    /** Class QuadraticProbingHashTable **/
    class QuadraticProbingHashTable {
        private int currentSize, maxSize;
        private String[] keys;
        private String[] vals;

        /** Constructor **/
        public QuadraticProbingHashTable(int capacity) {
            currentSize = 0;
            maxSize = capacity;
            keys = new String[maxSize];
            vals = new String[maxSize];
        }

        /** Function to clear hash table **/
        public void makeEmpty() {
            currentSize = 0;
```

```
27.             keys = new String[maxSize];
28.             vals = new String[maxSize];
29.         }
30.
31.         /** Function to get size of hash table
  **/
32.         public int getSize() {
33.             return currentSize;
34.         }
35.
36.         /** Function to check if hash table is
  full **/
37.         public boolean isFull() {
38.             return currentSize == maxSize;
39.         }
40.
41.         /** Function to check if hash table is
  empty **/
42.         public boolean isEmpty() {
43.             return getSize() == 0;
44.         }
45.
46.         /** Function to check if hash table
  contains a key **/
47.         public boolean contains(String key) {
48.             return get(key) != null;
49.         }
50.
51.         /** Function to get hash code of a given
  key **/
52.         private int hash(String key) {
53.             return (key.hashCode() % maxSize +
  maxSize) % maxSize; // Ensure positive index
54.         }
```

```
55.
56.         /** Function to insert key-value pair **/
57.         public void insert(String key, String
  val) {
58.             if (isFull()) {
59.                 System.out.println("Hash table is
  full. Cannot insert.");
60.                 return;
61.             }
62.
63.             int tmp = hash(key);
64.             int i = tmp, h = 1;
65.
66.             do {
67.                 if (keys[i] == null) {
68.                     keys[i] = key;
69.                     vals[i] = val;
70.                     currentSize++;
71.                     return;
72.                 }
73.                 if (keys[i].equals(key)) {
74.                     vals[i] = val; // Update
  existing key
75.                     return;
76.                 }
77.                 i = (tmp + h * h) % maxSize; //
  Update index for quadratic probing
78.                 h++;
79.             } while (i != tmp);
80.         }
81.
82.         /** Function to get value for a given key
  **/
83.         public String get(String key) {
```

```
84.             int i = hash(key), h = 1;
85.
86.             while (keys[i] != null) {
87.                 if (keys[i].equals(key))
88.                     return vals[i];
89.                 i = (i + h * h) % maxSize;
90.                 h++;
91.             }
92.
93.             return null;
94.         }
95.
96.         /** Function to remove key and its value
  **/
97.         public void remove(String key) {
98.             if (!contains(key)) return;
99.
100.            /** Find position key and delete **/
101.            int i = hash(key), h = 1;
102.
103.            while (!key.equals(keys[i])) {
104.                i = (i + h * h) % maxSize;
105.                h++;
106.            }
107.
108.            keys[i] = vals[i] = null;
109.
110.            /** Rehash all keys **/
111.            for (i = (i + h * h) % maxSize;
  keys[i] != null; i = (i + h * h) % maxSize) {
112.                String tmp1 = keys[i], tmp2 =
  vals[i];
113.                keys[i] = vals[i] = null;
114.                currentSize--;
```

```
115.                 insert(tmp1, tmp2);
116.             }
117.             currentSize--;
118.         }
119.
120.         /** Function to print HashTable **/
121.         public void printHashTable() {
122.             System.out.println("\nHash Table: ");
123.             for (int i = 0; i < maxSize; i++)
124.                 if (keys[i] != null)
125.                     System.out.println(keys[i] +
  " " + vals[i]);
126.             System.out.println();
127.         }
128.     }
129.
130.     /** Class QuadraticProbingHashTableTest **/
131.     public class QuadraticProbingHashTableTest {
132.         public static void main(String[] args) {
133.             Scanner scan = new
  Scanner(System.in);
134.             System.out.println("Hash Table
  Test\n\n");
135.             System.out.println("Enter size");
136.
137.             /** Make object of
  QuadraticProbingHashTable **/
138.             QuadraticProbingHashTable qpht = new
  QuadraticProbingHashTable(scan.nextInt());
139.
140.             char ch;
141.             /** Perform QuadraticProbingHashTable
  operations **/
142.             do {
```

```
143.                System.out.println("\nHash Table
  Operations\n");
144.                System.out.println("1. insert ");
145.                System.out.println("2. remove");
146.                System.out.println("3. get");
147.                System.out.println("4. clear");
148.                System.out.println("5. size");
149.
150.                int choice = scan.nextInt();
151.                switch (choice) {
152.                    case 1:
153.                        System.out.println("Enter
  key and value");
154.                        qpht.insert(scan.next(),
  scan.next());
155.                        break;
156.                    case 2:
157.                        System.out.println("Enter
  key");
158.                        qpht.remove(scan.next());
159.                        break;
160.                    case 3:
161.                        System.out.println("Enter
  key");
162.                        System.out.println("Value
  = " + qpht.get(scan.next()));
163.                        break;
164.                    case 4:
165.                        qpht.makeEmpty();
166.                        System.out.println("Hash
  Table Cleared\n");
167.                        break;
168.                    case 5:
```

```
169.                         System.out.println("Size
  = " + qpht.getSize());
170.                         break;
171.                     default:
172.                         System.out.println("Wrong
  Entry \n ");
173.                         break;
174.                 }
175.                 /** Display hash table **/
176.                 qpht.printHashTable();
177.
178.                 System.out.println("\nDo you want
  to continue (Type y or n) \n");
179.                     ch = scan.next().charAt(0);
180.             } while (ch == 'Y' || ch == 'y');
181.
182.             scan.close(); // Close scanner to
  avoid resource leak
183.         }
184.     }
185.
```

Sorting Array

Identified Issues:

- **Total Errors**: The program contains **2 errors**.
- **Breakpoints Required**: To resolve these errors, **2 breakpoints** are necessary.

Error Analysis and Corrections:

1. **Error 1**: The loop initialization for iterating through the array is incorrect.
    - **Original Condition**: for (int i = 0; i >= n; i++);
    - **Correction**: Update it to for (int i = 0; i < n; i++) to ensure proper iteration over the array elements.
2. **Error 2**: The condition within the inner loop that determines the sorting order is reversed.

- **Original Condition**: if (a[i] <= a[j])
- **Correction**: Modify it to if (a[i] > a[j]) to accurately implement ascending order sorting.

```
3.// Sorting the array in ascending order
4.import java.util.Scanner;
5.
6.public class AscendingOrder
7.{
8.    public static void main(String[] args)
9.    {
10.           int n, temp;
11.           Scanner s = new Scanner(System.in);
12.           System.out.print("Enter no. of
  elements you want in array: ");
13.           n = s.nextInt();
14.           int a[] = new int[n];
15.           System.out.println("Enter all the
  elements:");
16.
17.           // Reading elements into the array
18.           for (int i = 0; i < n; i++)
19.           {
20.               a[i] = s.nextInt();
21.           }
22.
23.           // Corrected loop for sorting the
  array
24.           for (int i = 0; i < n; i++) //
  Changed condition from 'i >= n' to 'i < n'
25.           {
26.               for (int j = i + 1; j < n; j++)
27.               {
28.                   // Corrected condition for
  sorting
```

```
                    if (a[i] > a[j]) // Changed from 'a[i] <= a[j]' to 'a[i] > a[j]'
                    {
                        // Swap elements
                        temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                    }
                }
            }

            // Printing the sorted array
            System.out.print("Ascending Order: ");
            for (int i = 0; i < n - 1; i++)
            {
                System.out.print(a[i] + ", ");
            }
            System.out.print(a[n - 1]); // Print the last element without a comma
        }
    }

```

Stack Implementation Errors and Resolutions

- **Total Errors:** 2
- **Breakpoints Required:** 2

Error Analysis and Corrections

1. **Error in push Method:**

- **Description of Error:** The statement top-- is incorrectly implemented. This operation incorrectly decrements the stack pointer, which can lead to stack underflow.
- **Proposed Fix:** Modify the line to top++ to correctly increment the stack pointer, allowing for the addition of new elements to the stack.

2. **Error in display Method:**
   - **Description of Error:** The loop condition for (int i = 0; i > top; i++) is incorrectly set. This condition prevents any elements from being displayed if top is greater than or equal to zero.
   - **Proposed Fix:** Change the loop condition to for (int i = 0; i <= top; i++) to ensure that all elements in the stack are displayed correctly.

```
3.// Stack implementation in Java
4.import java.util.Arrays;
5.
6.public class StackMethods {
7.    private int top;
8.    int size;
9.    int[] stack;
10.
11.        public StackMethods(int arraySize) {
12.            size = arraySize;
13.            stack = new int[size];
14.            top = -1; // Initialize top to -1 to
  indicate an empty stack
15.        }
16.
17.        public void push(int value) {
18.            if (top == size - 1) {
19.                System.out.println("Stack is
  full, can't push a value");
20.            } else {
21.                top++; // Increment the top
  pointer
22.                stack[top] = value; // Push the
  value onto the stack
23.            }
```

```
24.         }
25.
26.         public void pop() {
27.             if (!isEmpty()) {
28.                 System.out.println("Popped value:
  " + stack[top]); // Print the popped value
29.                 top--; // Decrement the top
  pointer
30.             } else {
31.                 System.out.println("Can't
  pop...stack is empty");
32.             }
33.         }
34.
35.         public boolean isEmpty() {
36.             return top == -1; // Check if the
  stack is empty
37.         }
38.
39.         public void display() {
40.             if (isEmpty()) {
41.                 System.out.println("Stack is
  empty.");
42.                 return;
43.             }
44.
45.             System.out.print("Stack elements: ");
46.             for (int i = 0; i <= top; i++) { //
  Corrected loop condition to display all elements
47.                 System.out.print(stack[i] + " ");
48.             }
49.             System.out.println();
50.         }
51.     }
```

```
52.
53.     public class StackReviseDemo {
54.         public static void main(String[] args) {
55.             StackMethods newStack = new
  StackMethods(5);
56.             newStack.push(10);
57.             newStack.push(1);
58.             newStack.push(50);
59.             newStack.push(20);
60.             newStack.push(90);
61.
62.             newStack.display(); // Display stack
  elements
63.             newStack.pop(); // Pop last value
64.             newStack.pop(); // Pop next value
65.             newStack.pop(); // Pop next value
66.             newStack.pop(); // Pop next value
67.             newStack.display(); // Display stack
  after pops
68.         }
69.     }
70.
```

Tower of Hanoi Implementation

Errors and Fixes:

- **How many errors are there in the program?** There is 1 error in the program.
- **How many breakpoints do you need to fix this error?** We need 1 breakpoint to fix this error.
- **Steps Taken to Fix the Error:**

- **Error:** In the recursive call doTowers(topN ++, inter--, from+1, to+1);, incorrect increments and decrements are applied to the variables.
- **Fix:** Change the call to doTowers(topN - 1, inter, from, to); for proper recursion and to follow the Tower of Hanoi logic.

```
// Tower of Hanoi
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3;
        doTowers(nDisks, 'A', 'B', 'C');
    }

    public static void doTowers(int topN, char
from, char inter, char to) {
        if (topN == 1) {
            System.out.println("Disk 1 from " +
from + " to " + to);
        } else {
            doTowers(topN - 1, from, to, inter);
// Move top N-1 disks from source to intermediate
            System.out.println("Disk " + topN + "
from " + from + " to " + to); // Move the Nth disk
            doTowers(topN - 1, inter, from, to);
// Move N-1 disks from intermediate to destination
        }
    }
}

```