**Data Structures and Algorithms**

**Assignment 1(PART 2)**

**Jan 1 Semester**

**Name: Jeslyn Ho Ka Yan:**

**Uow ID:** **8535383**

**Date of Submission:** **29 Jan 2024**

**Table of Content**

1 Algotithm 1 (Sorted array A and array B) 3

2 Algorithm 1 Analysis 5

3 Algotithm 2 (Sorted array A and unsorted array B) 7

4 Algorithm 2 Analysis 9

5 Screen-Shots of the output of Algorithm 1 and 2 11

# Algotithm 1 (Sorted array A and array B)

import java.util.Arrays**;**import java.util.HashSet**;**import java.util.Random**;**import java.util.Set**;**public class Algorithm1 {  
  
 private static void bubbleSort(int[] arr) {  
 int n = arr.length**;** boolean swapped**;** do {  
 swapped = false**;** // Reset the swapped flag for each iteration  
 for (int i = **1;** i < n**;** i++) {  
 if (arr[i - **1**] > arr[i]) {  
 int temp = arr[i - **1**]**;** arr[i - **1**] = arr[i]**;** arr[i] = temp**;** swapped = true**;** }  
 }  
 } while (swapped)**;**// Continue the loop as long as swaps are happening  
 }  
  
 private static int[] generateArrayA(int lengthA) {  
 int[] arr = new int[lengthA]**;** // Initialize an array for A with specified length  
 Set<Integer> uniqueNumbers = new HashSet<>()**;**// Set to keep track of unique numbers  
 Random random = new Random()**;** for (int i = **0;** i < lengthA**;** i++) {  
 int uniqueRandom**;** do {  
 uniqueRandom = random.nextInt(**100**) + **1;** // Generate random number in the range 1 to 10  
 } while (!uniqueNumbers.add(uniqueRandom))**;** // Add returns false if the value is already present  
  
 arr[i] = uniqueRandom**;** }  
  
 return arr**;** }

private static int[] generateArrayB(int lengthA**,** int lengthB) {  
 int[] arr = new int[lengthB]**;** // Initialize an array for B with specified length  
 Set<Integer> uniqueNumbers = new HashSet<>()**;** // Set to keep track of unique numbers  
 Random random = new Random()**;** for (int i = **0;** i < lengthB**;** i++) {  
 int uniqueRandom**;** do {  
 uniqueRandom = random.nextInt(lengthA) + **1;** // Generate random number in the range 1 to lengthA  
 } while (!uniqueNumbers.add(uniqueRandom))**;** // Add returns false if the value is already present  
  
 arr[i] = uniqueRandom**;** }  
 return arr**;** }

public static int[] algorithm1(int[] arrayA**,** int[] arrayB) {  
 *bubbleSort*(arrayA)**;** *bubbleSort*(arrayB)**;** // the only differences between Algorithm 1 and 2  
 int[] result = new int[arrayB.length]**;** for (int i = **0;** i < arrayB.length**;** i++) {  
 int index = arrayB[i] - **1;** //Since arrayB entries fall between 1 and lengthA, adjust the index.  
 result[i] = arrayA[index]**;** // Extract the relevant element from array A that has been sorted.  
 }  
 return result**;** }

public static void main(String[] args) {  
 Random random = new Random()**;** // Generate random lengths for arrays A and B  
 int lengthA = random.nextInt(**10**) + **1;** int lengthB = random.nextInt(lengthA) + **1;** int[] arrayA = *generateArrayA*(lengthA)**;** int[] arrayB = *generateArrayB*(lengthA**,** lengthB)**;** System.*out*.println("Generated Array A: " + Arrays.*toString*(arrayA))**;** System.*out*.println("Generated Array B: " + Arrays.*toString*(arrayB))**;** int[] result = *algorithm1*(arrayA**,** arrayB)**;** // apply algorithm1 to generate result array  
  
 System.*out*.println("Sorted Array A: " + Arrays.*toString*(arrayA))**;** System.*out*.println("Sorted Array B: " + Arrays.*toString*(arrayB))**;** System.*out*.println("Result: " + Arrays.*toString*(result))**;** }  
}

# Algorithm 1 Analysis

|  |
| --- |
|  |
| 1 |
|  |
| N+1 |
| n |
| n\*(n+1) |
| n\*n |
| a |
| a |
| a |
| a |
|  |
|  |
| b |

private static void bubbleSort(int[] arr) {

int n = arr.length;

boolean swapped;

do {

swapped = false;

for (int i = 1; i < n; i++) {

 if (arr[i - 1] > arr[i]) {

int temp = arr[i - 1];

arr[i - 1] = arr[i];

arr[i] = temp;

swapped = true;

}

}

} while (swapped);

}

|  |
| --- |
| 1 |
| 1 |
| 1 |
|  |
| n+1 |
|  |
| N\*(n+1) |
| c |
|  |
| n |
|  |
| d |
|  |

private static int[] generateArrayA(int lengthA) {

int[] arr = new int[lengthA];

Set<Integer> uniqueNumbers = new HashSet<>();

Random random = new Random();

A math symbols with black letters

Description automatically generated with medium confidence for (int i = 0; i < lengthA; i++) {

int uniqueRandom;

do {

uniqueRandom = random.nextInt(100) + 1;

} while (!uniqueNumbers.add(uniqueRandom));

arr[i] = uniqueRandom;

}

return arr;

}

private static int[] generateArrayB(int lengthA, int lengthB) {

|  |
| --- |
| 1 |
| 1 |
| 1 |
|  |
| N+1 |
|  |
| N\*(n+1) |
| c |
|  |
| n |
|  |
|  |
| d |

int[] arr = new int[lengthB];

Set<Integer> uniqueNumbers = new HashSet<>();

Random random = new Random();

A math symbols with black letters

Description automatically generated with medium confidence

for (int i = 0; i < lengthB; i++) {

int uniqueRandom;

do {

uniqueRandom = random.nextInt(lengthA) + 1;

} while (!uniqueNumbers.add(uniqueRandom));

arr[i] = uniqueRandom;

}

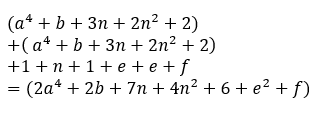
return arr;

}

public static int[] algorithm1(int[] arrayA, int[] arrayB) {

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

bubbleSort(arrayA);

 bubbleSort(arrayB);

int[] result = new int[arrayB.length];

for (int i = 0; i < arrayB.length; i++) {

int index = arrayB[i] - 1;

result[i] = arrayA[index];

}

return result;

}

public static void main(String[] args) {

|  |
| --- |
| 1 |
|  |
| 1 |
| 1 |
|  |
|  |
|  |
|  |
| 1 |
| 1 |
|  |
|  |
|  |
| 1 |
| 1 |
| 1 |
|  |

Random random = new Random();

A math equations on a white background

Description automatically generated

int lengthA = random.nextInt(10) + 1;

int lengthB = random.nextInt(lengthA) + 1;

int[] arrayA = generateArrayA(lengthA);

int[] arrayB = generateArrayB(lengthA, lengthB);

System.out.println("Generated Array A: " + Arrays.toString(arrayA));

System.out.println("Generated Array B: " + Arrays.toString(arrayB));

int[] result = algorithm1(arrayA, arrayB);

System.out.println("Sorted Array A: " + Arrays.toString(arrayA));

System.out.println("Sorted Array B: " + Arrays.toString(arrayB));

System.out.prin

tln("Result: " + Arrays.toString(result));

}

Where:  
 a, c, e = 0 or 1 or 0.5 (best case is 1, worst case is 0)

b, d, f = 0 or 1 or 0.5 (**best case is 0**, **worst case is 1**)

|  |
| --- |
|  |
|  |
| 1 |
|  |
| N+1 |
| e |
| e |
|  |
| f |

For the worst case scenario, the runtime efficiency for Alogrithm 1

is

# Algotithm 2 (Sorted array A and unsorted array B)

import java.util.Arrays**;**import java.util.HashSet**;**import java.util.Random**;**import java.util.Set**;**public class Algorithm1 {  
  
 private static void bubbleSort(int[] arr) {  
 int n = arr.length**;** boolean swapped**;** do {  
 swapped = false**;** // Reset the swapped flag for each iteration  
 for (int i = **1;** i < n**;** i++) {  
 if (arr[i - **1**] > arr[i]) {  
 int temp = arr[i - **1**]**;** arr[i - **1**] = arr[i]**;** arr[i] = temp**;** swapped = true**;** }  
 }  
 } while (swapped)**;**// Continue the loop as long as swaps are happening  
 }  
  
 private static int[] generateArrayA(int lengthA) {  
 int[] arr = new int[lengthA]**;** // Initialize an array for A with specified length  
 Set<Integer> uniqueNumbers = new HashSet<>()**;**// Set to keep track of unique numbers  
 Random random = new Random()**;** for (int i = **0;** i < lengthA**;** i++) {  
 int uniqueRandom**;** do {  
 uniqueRandom = random.nextInt(**100**) + **1;** // Generate random number in the range 1 to 10  
 } while (!uniqueNumbers.add(uniqueRandom))**;** // Add returns false if the value is already present  
  
 arr[i] = uniqueRandom**;** }  
  
 return arr**;** }

private static int[] generateArrayB(int lengthA**,** int lengthB) {  
 int[] arr = new int[lengthB]**;** // Initialize an array for B with specified length  
 Set<Integer> uniqueNumbers = new HashSet<>()**;** // Set to keep track of unique numbers  
 Random random = new Random()**;** for (int i = **0;** i < lengthB**;** i++) {  
 int uniqueRandom**;** do {  
 uniqueRandom = random.nextInt(lengthA) + **1;** // Generate random number in the range 1 to lengthA  
 } while (!uniqueNumbers.add(uniqueRandom))**;** // Add returns false if the value is already present  
  
 arr[i] = uniqueRandom**;** }  
 return arr**;** }

public static int[] algorithm1(int[] arrayA**,** int[] arrayB) {  
 *bubbleSort*(arrayA)**;**

int[] result = new int[arrayB.length]**;** for (int i = **0;** i < arrayB.length**;** i++) {  
 int index = arrayB[i] - **1;** //Since arrayB entries fall between 1 and lengthA, adjust the index.  
 result[i] = arrayA[index]**;** // Extract the relevant element from array A that has been sorted.  
 }  
 return result**;** }

public static void main(String[] args) {  
 Random random = new Random()**;** // Generate random lengths for arrays A and B  
 int lengthA = random.nextInt(**10**) + **1;** int lengthB = random.nextInt(lengthA) + **1;** int[] arrayA = *generateArrayA*(lengthA)**;** int[] arrayB = *generateArrayB*(lengthA**,** lengthB)**;** System.*out*.println("Generated Array A: " + Arrays.*toString*(arrayA))**;** System.*out*.println("Generated Array B: " + Arrays.*toString*(arrayB))**;** int[] result = *algorithm1*(arrayA**,** arrayB)**;** // apply algorithm1 to generate result array  
  
 System.*out*.println("Sorted Array A: " + Arrays.*toString*(arrayA))**;** System.*out*.println("Unsorted Array B: " + Arrays.*toString*(arrayB))**;** System.*out*.println("Result: " + Arrays.*toString*(result))**;** }  
}

# Algorithm 2 Analysis

|  |
| --- |
|  |
| 1 |
|  |
| N+1 |
| n |
| n\*(n+1) |
| n\*n |
| a |
| a |
| a |
| a |
|  |
|  |
| b |

private static void bubbleSort(int[] arr) {

int n = arr.length;

boolean swapped;

do {

swapped = false;

 for (int i = 1; i < n; i++) {

if (arr[i - 1] > arr[i]) {

int temp = arr[i - 1];

arr[i - 1] = arr[i];

arr[i] = temp;

swapped = true;

}

}

} while (swapped);

}

|  |
| --- |
| 1 |
| 1 |
| 1 |
|  |
| n+1 |
|  |
| N\*(n+1) |
| c |
|  |
| n |
|  |
| d |
|  |

private static int[] generateArrayA(int lengthA) {

int[] arr = new int[lengthA];

Set<Integer> uniqueNumbers = new HashSet<>();

Random random = new Random();

A math symbols with black letters

Description automatically generated with medium confidence

for (int i = 0; i < lengthA; i++) {

int uniqueRandom;

do {

uniqueRandom = random.nextInt(100) + 1;

} while (!uniqueNumbers.add(uniqueRandom));

arr[i] = uniqueRandom;

}

return arr;

}

private static int[] generateArrayB(int lengthA, int lengthB) {

|  |
| --- |
| 1 |
| 1 |
| 1 |
|  |
| N+1 |
|  |
| N\*(n+1) |
| c |
|  |
| n |
|  |
|  |
| d |

int[] arr = new int[lengthB];

Set<Integer> uniqueNumbers = new HashSet<>();

Random random = new Random();

A math symbols with black letters

Description automatically generated with medium confidence

for (int i = 0; i < lengthB; i++) {

int uniqueRandom;

do {

uniqueRandom = random.nextInt(lengthA) + 1;

} while (!uniqueNumbers.add(uniqueRandom));

arr[i] = uniqueRandom;

}

return arr;

}

public static int[] algorithm1(int[] arrayA, int[] arrayB) {

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

bubbleSort(arrayA);

A math symbols on a white background

Description automatically generated int[] result = new int[arrayB.length];

for (int i = 0; i < arrayB.length; i++) {

int index = arrayB[i] - 1;

result[i] = arrayA[index];

}

return result;

}

A math equations with black text

Description automatically generated with medium confidence public static void main(String[] args) {

|  |
| --- |
| 1 |
|  |
| 1 |
| 1 |
|  |
|  |
|  |
|  |
| 1 |
| 1 |
|  |
|  |
|  |
| 1 |
| 1 |
| 1 |
|  |

Random random = new Random();

int lengthA = random.nextInt(10) + 1;

int lengthB = random.nextInt(lengthA) + 1;

int[] arrayA = generateArrayA(lengthA);

int[] arrayB = generateArrayB(lengthA, lengthB);

System.out.println("Generated Array A: " + Arrays.toString(arrayA));

System.out.println("Generated Array B: " + Arrays.toString(arrayB));

int[] result = algorithm1(arrayA, arrayB);

System.out.println("Sorted Array A: " + Arrays.toString(arrayA));

System.out.println("Sorted Array B: " + Arrays.toString(arrayB));

System.out.prin

tln("Result: " + Arrays.toString(result));

}

|  |
| --- |
|  |
|  |
| 1 |
|  |
| N+1 |
| e |
| e |
|  |
| f |

Where:  
 a, c, e = 0 or 1 or 0.5 (best case is 1, worst case is 0)

b, d, f = 0 or 1 or 0.5 (**best case is 0**, **worst case is 1**)

For the worst case scenario, the runtime efficiency for Alogrithm 2

is

# Screen-Shots of the output of Algorithm 1 and 2

**Algorithm1 Output**

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

**Algorithm2 Output**

A black background with white text

Description automatically generated

A black background with white text

Description automatically generatedA black background with white text

Description automatically generated