Assignment – 9

Name - Md. Firoze Baba

Task 1:

Array Sorting and Searching:

a) Implement a function called BruteForceSort that sorts an array using the brute force approach. Use this function to sort an array created with InitializeArray.

Program:

```
import java.util.Arrays;

public class Task1A {

	public static void bruteForceSort(int[] arr) {
		Arrays.sort(arr);
	}

	public static int[] initializeArray(int n,int min,int max) {
		int[] arr=new int[n];
		for(int i=0;i<n;i++) {
			arr[i]=(int)(Math.random()*(max-min+1)+min);
		}
		return arr;
	}

	public static void main(String[] args) {

		int n = 10;
	   int min = 1;
	   int max = 100;
	   int[] arr = initializeArray(n, min, max);

	   System.out.println("Array:");
	   System.out.println(Arrays.toString(arr));

	   bruteForceSort(arr);
	   System.out.println("Sorted Array:");
	   System.out.println(Arrays.toString(arr));

	}

}
```

Output:

```
Array:
[44, 8, 81, 68, 6, 42, 65, 18, 92, 45]
Sorted Array:
[6, 8, 18, 42, 44, 45, 65, 68, 81, 92]
```

b) Write a function named PerformLinearSearch that searches for a specific element in an array and returns the index of the element if found or -1 if not found.

Program:

```
public class Task1 {

	public static int performLinearSearch(int[] arr,int key) {

		for(int i=0;i<arr.length-1;i++) {
			if(arr[i] == key) {
				return i;
			}
		}

		return -1;
	}

	public static void main(String[] args) {
		int[] arr= {10,15,22,26,34,39,48,56,62};
		int key=39;
		int result=performLinearSearch(arr, key);
		if(result != -1) {
			System.out.println
			("Element found at index of: "+result);
		}
		else {
			System.out.println("Element not found");
		}
	}

}
```

Output:

```
Element found at index of: 5
```

Task 2:

Two-Sum Problem:

Given an array of integers, write a program that finds if there are two numbers that add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice. Optimize the solution for time complexity.

Program:

```
import java.util.HashMap;

public class Task2 {

	public static int[] findTwoSum(int[] arr, int key) {

		HashMap<Integer, Integer> map = new HashMap<>();
		for (int i = 0; i < arr.length; i++) {
		  int complement = key - arr[i];
		  if (map.containsKey(complement)) {
		      return new int[]{map.get(complement), i};
		  }
		  map.put(arr[i], i);
	    }
		throw new IllegalArgumentException("no two sum
		solution");
	}

	public static void main(String[] args) {

	   int[] arr = {4,8,19,25};
	   int key = 23;
	   int[] sum = findTwoSum(arr, key);
	   System.out.println("Indices: " + sum[0] + ", " + sum[1]);

	}

}
```

Output-1:

```
Indices: 0, 2
```

Output-2: (If key is 19)

```
Exception in thread "main" java.lang.IllegalArgumentException:
no two sum solution
	at day4.Task2.findTwoSum(Task2.java:17)
```

```
	at day4.Task2.main(Task2.java:24)
```

Task 3:

Understanding Functions through Arrays:

Write a recursive function named SumArray that calculates and returns the sum of elements in an array, demonstrate with example.

Program:

```
public class Task3 {

	public static int sumArray(int[] arr,int index) {
		if(index == arr.length) {
			return 0;
		}
		else {
			return arr[index] + sumArray(arr, index + 1);
		}
	}

	public static void main(String[] args) {
		int[] arr= {1,2,3,4,5};
		int index=0;
		System.out.println
		("sum of elements in an array is:
		"+sumArray(arr,index));
	}

}
```

Output:

```
sum of elements in an array is: 15
```

Explanation:

- **Base Case:**

  The base case for our recursive function is when the index is greater than or equal to the length of the array. When this condition is met, the function returns 0 because there are no more elements to add.

- **Recursive Case:**

  For the recursive case, the function returns the current element at array[index] plus the result of the function call with the next index (index + 1).

- **Example Execution:**

  Suppose exampleArray is {1, 2, 3, 4, 5}.
  The initial call is SumArray(exampleArray, 0).
  The first call checks if index >= array.length. Since index is 0 and the array length is 5, it proceeds to return array[0] (which is 1) plus SumArray(exampleArray, 1).
  The next call is SumArray(exampleArray, 1), which returns array[1] (2) plus SumArray(exampleArray, 2), and so on.
  This continues until index reaches 5, at which point SumArray(exampleArray, 5) returns 0.
  The calls then return their results back up the call stack, summing the elements as they go:
  SumArray(exampleArray, 4) returns 5 + 0 = 5
  SumArray(exampleArray, 3) returns 4 + 5 = 9
  SumArray(exampleArray, 2) returns 3 + 9 = 12
  SumArray(exampleArray, 1) returns 2 + 12 = 14
  SumArray(exampleArray, 0) returns 1 + 14 = 15

- **Conclusion:**

  This example demonstrates how to use recursion to sum the elements of an array. The key points to understand are defining a clear base case to stop the recursion and ensuring each recursive call processes the next element in the array. The function builds up the sum by adding each element in the array to the sum of the elements that follow it.

Task 4:

Advanced Array Operations:

a) Implement a method SliceArray that takes an array, a starting index, and an end index, then returns a new array containing the elements from the start to the end index.

Program:

```
import java.util.Arrays;

public class Task4A {

	public static int[] sliceArray(int[] arr,int start, int end)
{
		if(start < 0 || start >= arr.length ||
			end < 0 || end > arr.length || start > end) {
			throw new IllegalArgumentException
				("Invalid start or end index");
		}
		int length =end - start;
		int[] slicarr=new int[length];
		for(int i=0; i<length;i++) {
			slicarr[i]=arr[start+i];
		}
		return slicarr;
	}
```

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

       int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
       int start = 2;
       int end = 6;
       int[] slicedArray = sliceArray(arr, start, end);
       System.out.println("Sliced Array:");
       System.out.println(Arrays.toString(slicedArray));

    }

}
```

Output:

```
Sliced Array:
[3, 4, 5, 6]
```

b) Create a recursive function to find the nth element of a Fibonacci sequence and store the first n elements in an array.

Program:

```
import java.util.Arrays;

public class Task4 {

    public static int sum(int n) {
        if(n <= 1) {
            return n;
        }
        return sum(n-1) + sum(n-2);
    }
    public static int[] fibonacci(int n) {
        int[] arr= new int[n];
        for(int i=0;i<n;i++) {
            arr[i]=sum(i);
        }
        return arr;
    }

    public static void main(String[] args) {

        int n=10;

        int[] arrfib=new int[n];
        arrfib=fibonacci(n);
        System.out.println
            ("Fibonacci sequence up to " + n +" elements:
                "+Arrays.toString(arrfib));
        System.out.println
            ("The "+n+"th element of Fibonacci sequence is:
                "+arrfib[n-1]);
    }

}
```

Output:

```
Fibonacci sequence up to 10 elements:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
The 10th element of Fibonacci sequence is: 34
```