**Name: Abishek S H Date: 14/11/2024**

**Dept: CSE**

**1.Stock buy and sell**

**Code:**

class Solution{

ArrayList<ArrayList<Integer> > stockBuySell(int A[], int n) {

ArrayList<ArrayList<Integer>> result = new ArrayList<>();

int i = 0;

while (i < n - 1) {

while (i < n - 1 && A[i + 1] <= A[i]) {

i++;

}

if (i == n - 1) break;

int buy = i++;

while (i < n && A[i] >= A[i - 1]) {

i++;

}

int sell = i - 1;

ArrayList<Integer> transaction = new ArrayList<>();

transaction.add(buy);

transaction.add(sell);

result.add(transaction);

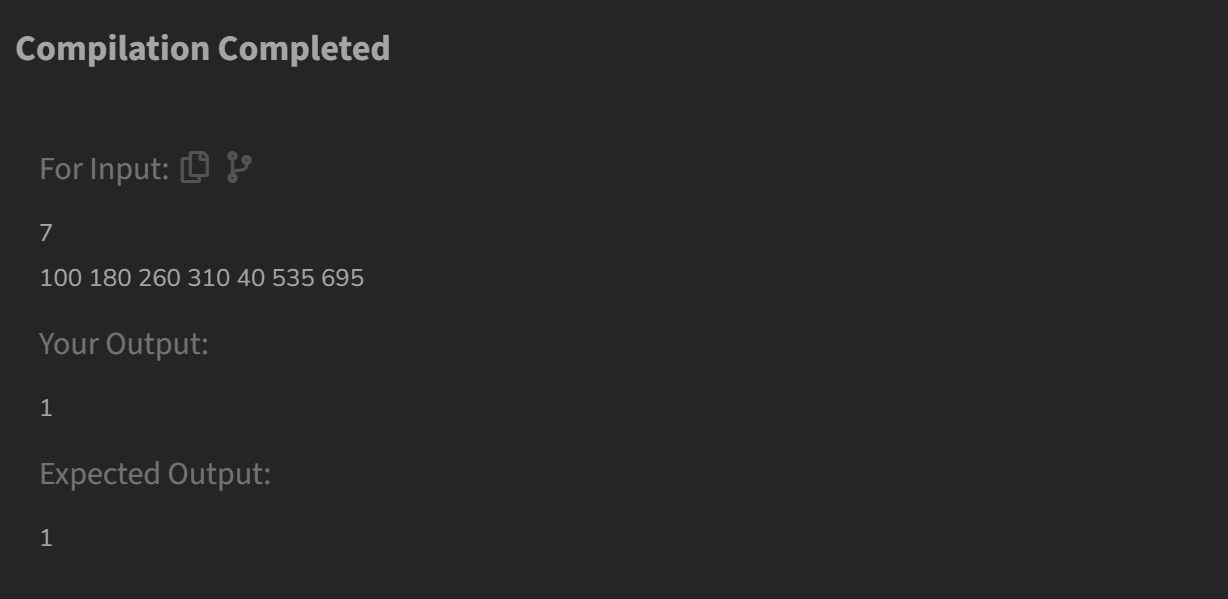
}

return result;

}

}

**Output:**

****

**Time Complexity: O(n)**

**2. Coin change(count ways)**

**Code:**

class Solution {

public int count(int coins[], int sum) {

int n = coins.length;

int[] arr = new int[sum + 1];

arr[0] = 1;

for (int coin : coins) {

for (int j = coin; j <= sum; j++) {

arr[j] += arr[j - coin];

}

}

return arr[sum];

}

}

**Output:**

****

**Time Complexity: O(n\*sum)**

**3. First and last Occurrence**

**Code:**

public int[] findFirstAndLast(int[] arr, int x) {

int first = findFirstOccurrence(arr, x);

int last = findLastOccurrence(arr, x);

return new int[]{first, last};

}

private int findFirstOccurrence(int[] arr, int x) {

int low = 0, high = arr.length - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

result = mid; // potential first occurrence

high = mid - 1; // keep searching left

} else if (arr[mid] < x) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

private int findLastOccurrence(int[] arr, int x) {

int low = 0, high = arr.length - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

result = mid; // potential last occurrence

low = mid + 1; // keep searching right

} else if (arr[mid] < x) {

low = mid + 1;

} else {

high = mid - 1;

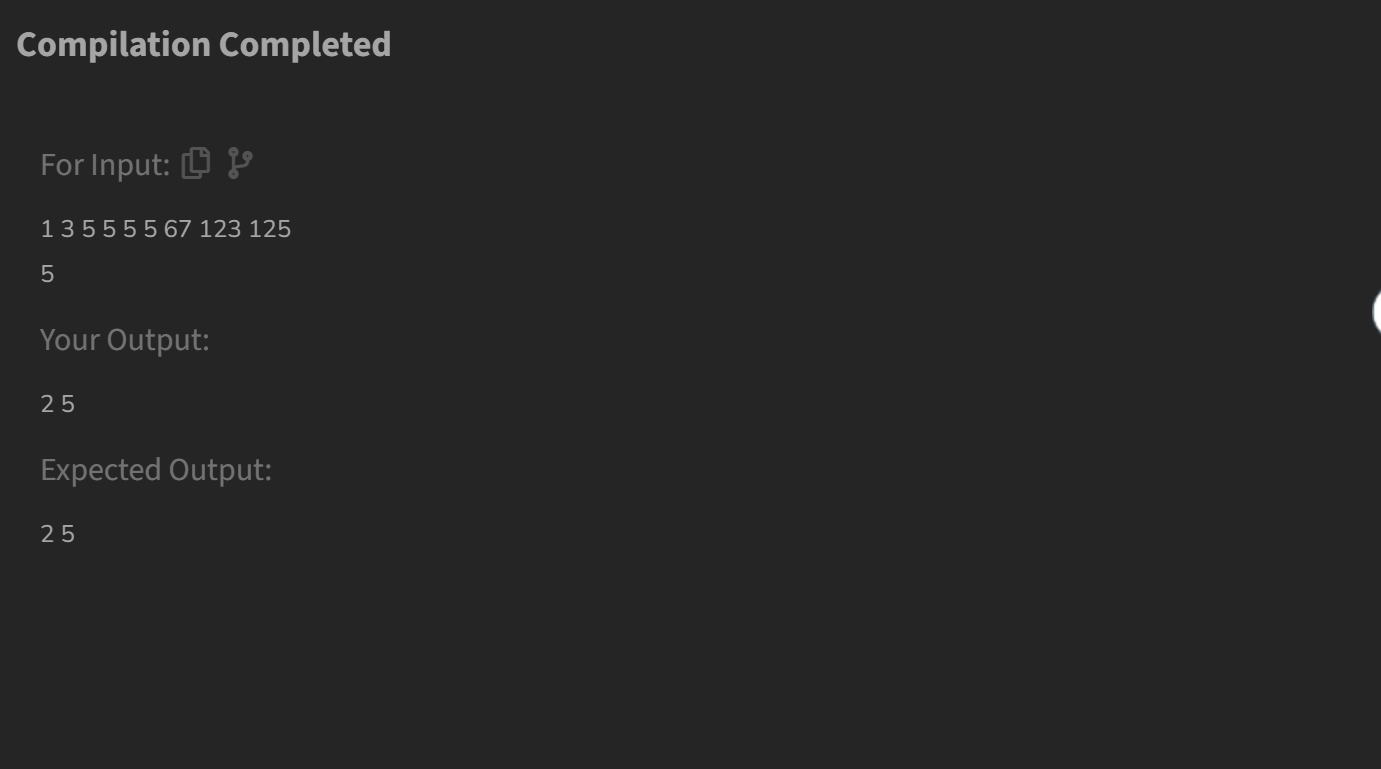
}

}

return result;

}

**Output:**



**Time Complexity: O(log n)**

**4. Find transition**

**Code:**

class Solution {

int transitionPoint(int arr[]) {

int low=0;

int high=arr.length-1;

while(low<=high){

int mid=low+(high-low)/2;

if(arr[mid]==0){

low=mid+1;

}

else if(arr[mid]==1){

if (mid == 0 || (mid > 0 && arr[mid - 1] == 0)){

return mid;

}

high=mid-1;

}

}

return -1;

}

}

**Output:**

****

**Time Complexity: O(log n)**

**5. Find repeating element**

**Code:**

int min = -1;

*// Creates an empty hashset*

HashSet<Integer> set = **new** HashSet<>();

*// Traverse the input array from right to left*

**for** (int i = arr.length - 1; i >= 0; i--) {

*// If element is already in hash set, update min*

**if** (set.contains(arr[i]))

min = i;

**else** *// Else add element to hash set*

set.add(arr[i]);

}

*// Print the result*

**if** (min != -1)

System.out.println(

"The first repeating element is "

+ arr[min]);

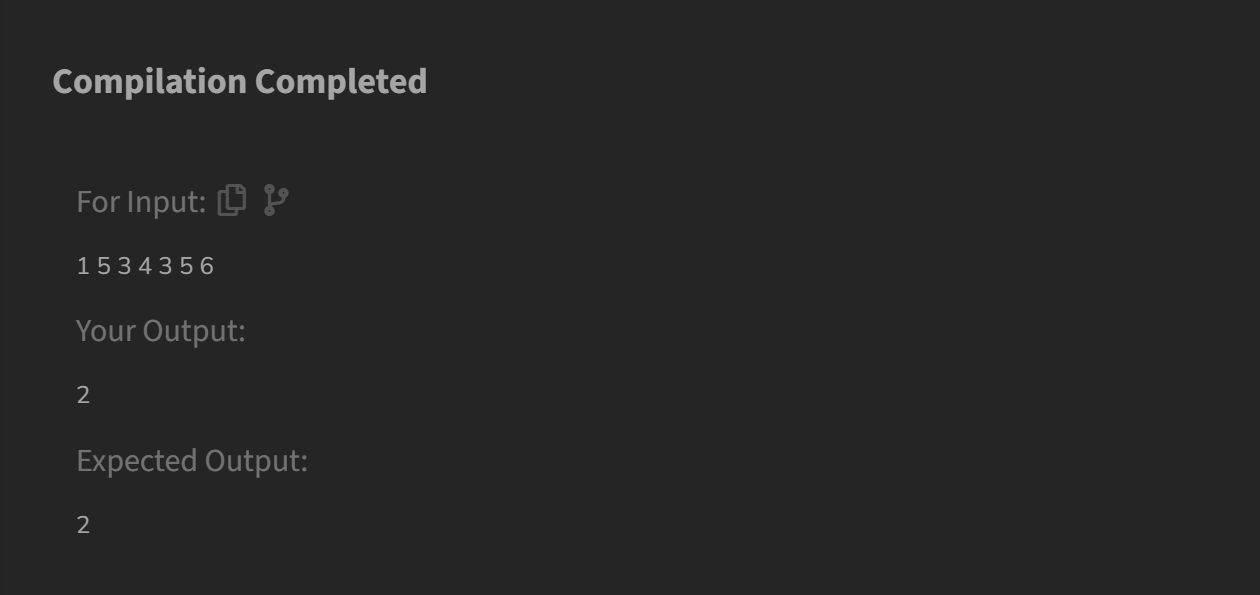
**else**

System.out.println(

"There are no repeating elements");

}

**Output:**

****

**Time Complexity: O(n)**

**6.Remove duplicates from sorted array**

**Code:**

class Solution {

public int remove\_duplicate(List<Integer> arr) {

int n = arr.size();

if (n == 0) {

return 0;

}

int index = 1;

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

if (!arr.get(i).equals(arr.get(index - 1))) {

arr.set(index, arr.get(i));

index++;

}

}

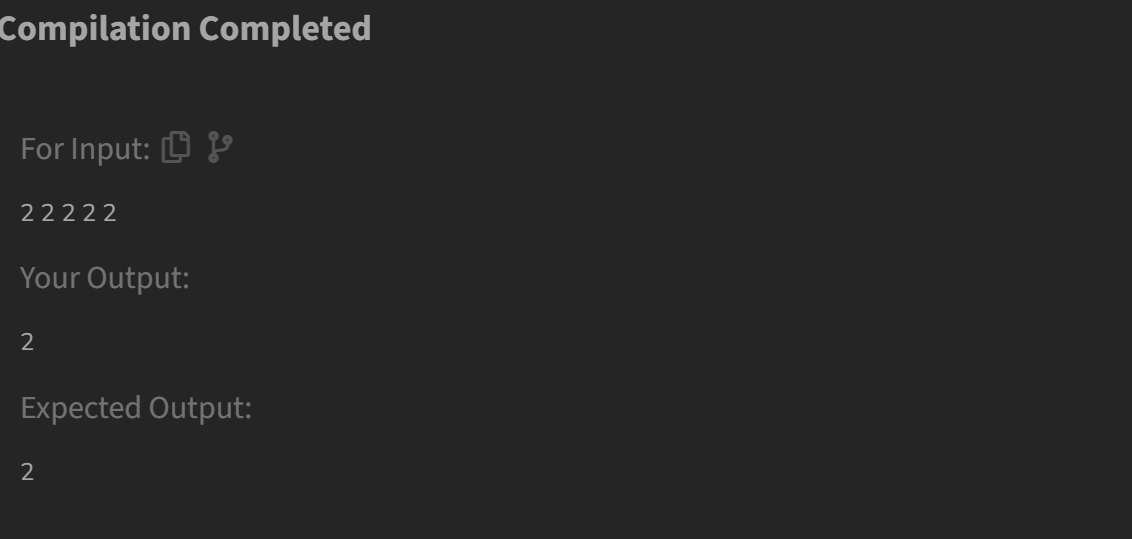
while (arr.size() > index) {

arr.remove(arr.size() - 1);

}

return index;

}}

**Output:  
**

**Time Complexity: O(n)**

**7.Minimum index**

**Code:**

package sample4;

import java.util.\*;

public class minInd {

static int minIndex(int arr[], int pos)

{

int target = arr[pos];

for (int i = pos; i >= 0; i--) {

if (arr[i] != target) {

return i + 1;

}

}

return 0;

}

public static void main(String[] args)

{

int arr[] = { 2, 1, 1, 1, 5, 2 };

System.***out***.println(*minIndex*(arr, 2));

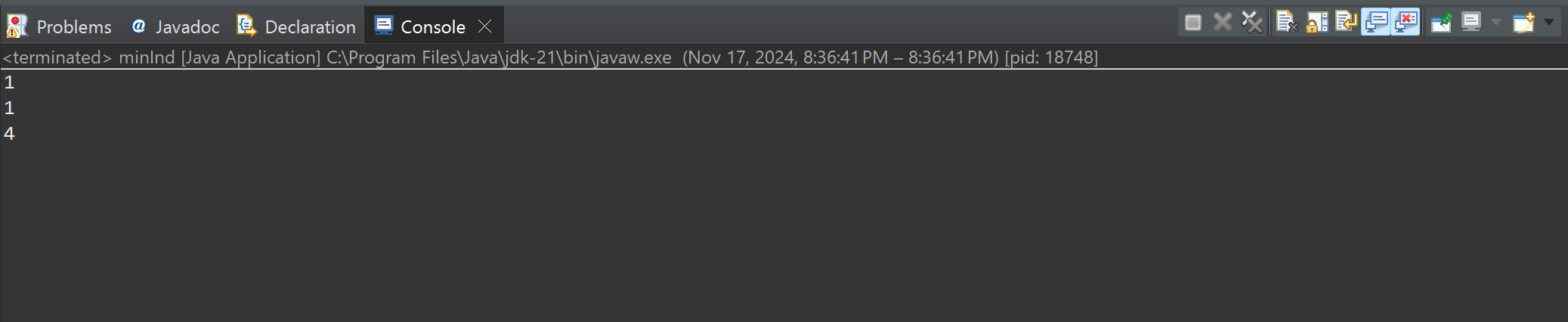
System.***out***.println(*minIndex*(arr, 3));

System.***out***.println(*minIndex*(arr, 4));

}

}

**Output:**

****

**Time Complexity: O(n log n)**

**8. Wave Array**

**Code:**

package sample4;

import java.util.\*;

public class waveArray {

static void solve(int[] arr) {

Arrays.*sort*(arr);

int temp=0;

for(int i=0;i<arr.length-1;i+=2) {

temp=arr[i];

arr[i]=arr[i+1];

arr[i+1]=temp;

}

System.***out***.println(Arrays.*toString*(arr));

}

public static void main(String[] args) {

int[] lst= {1,2,3,4,5};

*solve*(lst);

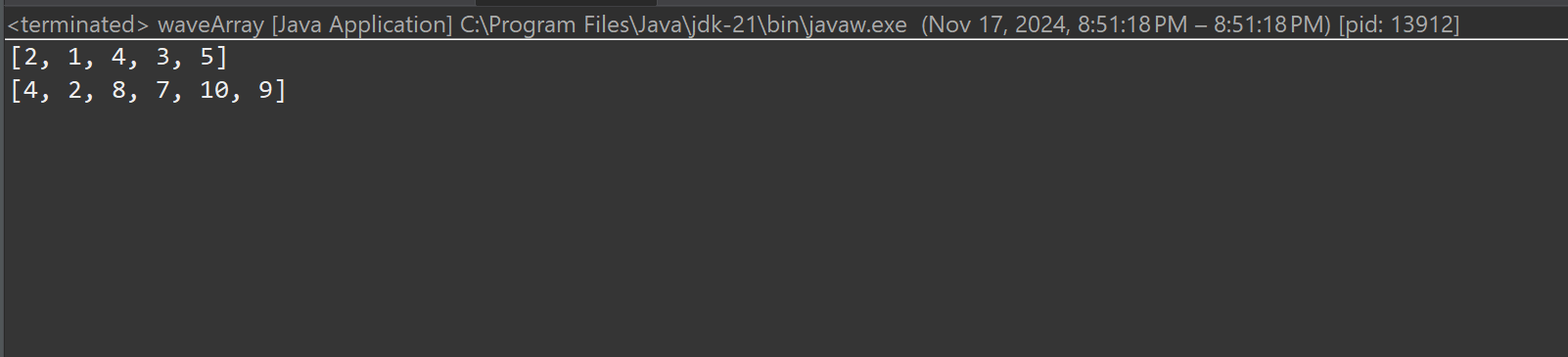
int[] lst1= {2, 4, 7, 8, 9, 10};

*solve*(lst1);

}

}

**Output:**

****

**Time Complexity: O(n)**