**SYED NADEEM G**

**SDE – Practice Day – 8**

(20/11/2024)

1. **3Sum Closest:**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Enter the size of the array:");

int n = scanner.nextInt();

int[] nums = new int[n];

System.***out***.println("Enter the elements of the array:");

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

nums[i] = scanner.nextInt();

}

System.***out***.println("Enter the target value:");

int target = scanner.nextInt();

int result = new Practice().threeSumClosest(nums, target);

System.***out***.println("The closest sum to the target is: " + result);

}

public int threeSumClosest(int[] nums, int target) {

int n = nums.length;

Arrays.*sort*(nums);

int min = Integer.***MAX\_VALUE***;

int ans = 0;

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

if (i > 0 && nums[i] == nums[i - 1]) continue;

int j = i + 1;

int k = n - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

if (sum == target) return sum;

else if (sum > target) k--;

else j++;

int dif = Math.*abs*(sum - target);

if (dif < min) {

ans = sum;

min = dif;

}

}

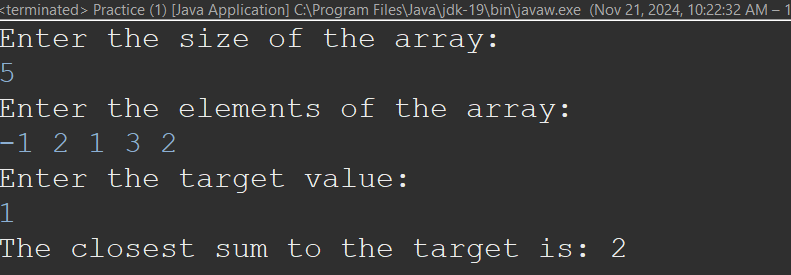
}

return ans;

}

}

Output:



Time Complexity: O(n^2)

1. **Jump Game II:**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

// Input: Number of elements

System.***out***.println("Enter the number of elements:");

int n = sc.nextInt();

// Input: Array elements

int[] nums = new int[n];

System.***out***.println("Enter the elements of the array:");

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

nums[i] = sc.nextInt();

}

// Output: Minimum jumps

int result = new Practice().jump(nums);

System.***out***.println("The minimum number of jumps to reach the end is: " + result);

}

public int jump(int[] nums) {

int near = 0;

int far = 0;

int ans = 0;

int jump = 0;

int n = nums.length - 1;

while (far < n) {

for (int i = near; i <= far; i++) {

ans = Math.*max*(ans, i + nums[i]);

}

near = far + 1;

far = ans;

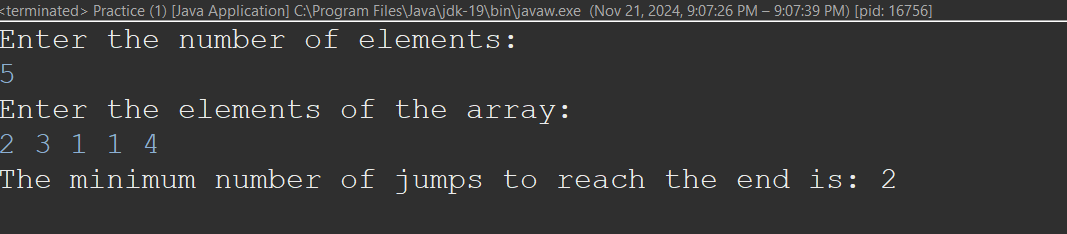
jump++;

}

return jump;

}

}

Output: 

Time Complexity: O(n)

1. **Group Anagrams**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

// Input size of the array

System.***out***.print("Enter the size of the array: ");

int n = scanner.nextInt();

scanner.nextLine(); // Consume the newline character

// Input elements of the array

System.***out***.println("Enter the elements of the array:");

String[] strs = new String[n];

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

strs[i] = scanner.nextLine();

}

// Invoke the method and print results

Solution solution = new Solution();

List<List<String>> result = solution.groupAnagrams(strs);

System.***out***.println("Grouped anagrams:");

for (List<String> group : result) {

System.***out***.println(group);

}

}

}

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> map = new HashMap<>();

for (String s : strs) {

char[] c = s.toCharArray();

Arrays.*sort*(c);

String res = new String(c);

if (!map.containsKey(res)) {

map.put(res, new ArrayList<>());

}

map.get(res).add(s);

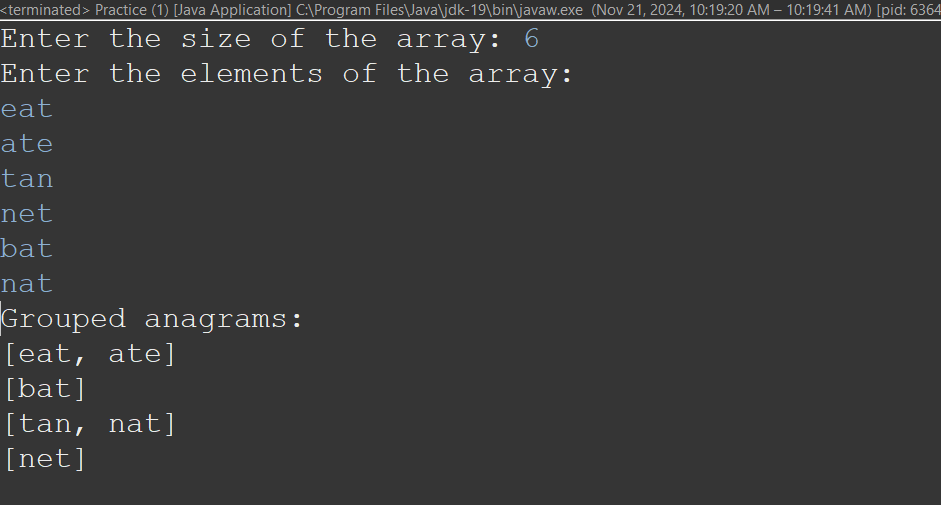
}

return new ArrayList<>(map.values());

}

}

Output:



Time Complexity: O(n)

1. **Decode Ways**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

// Input

System.***out***.println("Enter the string:");

String s = sc.nextLine();

// Output

int result = new Practice().numDecodings(s);

System.***out***.println("The number of ways to decode the string is: " + result);

}

public int numDecodings(String s) {

if (s == null || s.length() == 0) {

return 0;

}

int n = s.length();

int[] dp = new int[n + 1];

dp[0] = 1;

dp[1] = s.charAt(0) == '0' ? 0 : 1;

for (int i = 2; i <= n; i++) {

int one = Integer.*valueOf*(s.substring(i - 1, i));

int two = Integer.*valueOf*(s.substring(i - 2, i));

if (one >= 1 && one <= 9) {

dp[i] += dp[i - 1];

}

if (two >= 10 && two <= 26) {

dp[i] += dp[i - 2];

}

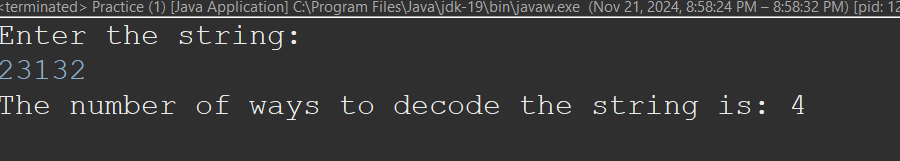
}

return dp[n];

}

}

Output:



Time Complexity: O(n)

1. **Best Time To sell and Buy Stocks II**

Code:

import java.util.Scanner;

class Practice{

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the Size of Array:");

int n = sc.nextInt();

int[] nums = new int[n];

System.***out***.println("Enter the elements");

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

nums[i] = sc.nextInt();

}

System.***out***.println("The Profit: "+*buySell*(nums));

}

private static int buySell(int[] nums) {

int n = nums.length;

int profit = 0;

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

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

profit += nums[i]-nums[i-1];

}

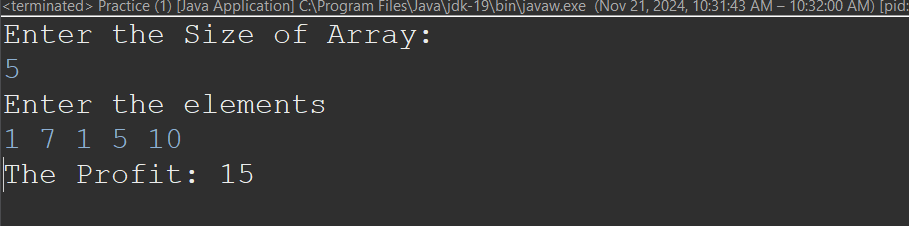
}

return profit;

}

}

Output:



Time Complexity: O(n)

1. **Number Of Islands**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

// Input grid dimensions

System.***out***.println("Enter the number of rows:");

int n = scanner.nextInt();

System.***out***.println("Enter the number of columns:");

int m = scanner.nextInt();

// Input grid

char[][] grid = new char[n][m];

System.***out***.println("Enter the grid (row by row, with 1 for land and 0 for water):");

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

String row = scanner.next();

grid[i] = row.toCharArray();

}

// Invoke the method and print the result

Solution solution = new Solution();

int result = solution.numIslands(grid);

System.***out***.println("The number of islands is: " + result);

}

}

class Solution {

public int numIslands(char[][] grid) {

int n = grid.length;

int m = grid[0].length;

int count = 0;

int[][] vis = new int[n][m];

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

for (int j = 0; j < m; j++) {

if (grid[i][j] == '1' && vis[i][j] == 0) {

dfs(i, j, grid, vis);

count++;

}

}

}

return count;

}

int[] x = {-1, +1, 0, 0};

int[] y = {0, 0, -1, +1};

private void dfs(int start, int end, char[][] grid, int[][] vis) {

int n = grid.length;

int m = grid[0].length;

if (start < 0 || start >= n || end < 0 || end >= m || grid[start][end] == '0' || vis[start][end] == 1) {

return;

}

vis[start][end] = 1;

for (int i = 0; i < 4; i++) {

int newS = start + x[i];

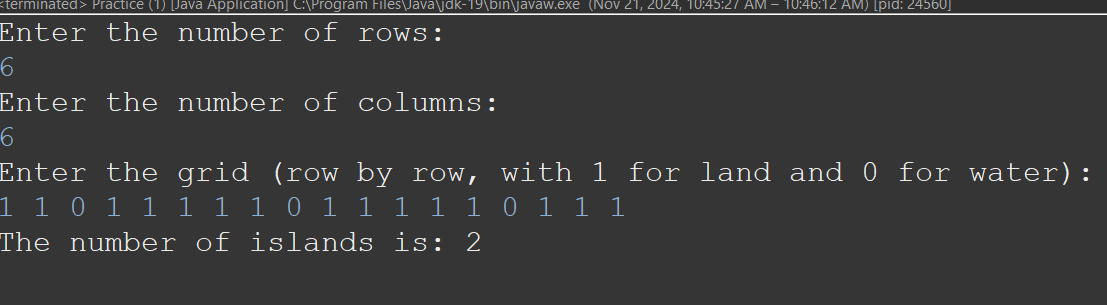
int newE = end + y[i];

dfs(newS, newE, grid, vis);

}

}

}

Output: 

Time Complexity: O(n\*m)

1. **Ternary Search**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

// Input

System.***out***.println("Enter the number of elements:");

int n = sc.nextInt();

// Input

int[] nums = new int[n];

System.***out***.println("Enter the elements:");

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

nums[i] = sc.nextInt();

}

System.***out***.println("Enter the element to search :");

int k = sc.nextInt();

int result = *TernarySearch*(nums, k);

System.***out***.println("The Position is: " + result);

}

private static int TernarySearch(int[] nums, int k) {

int n = nums.length;

int l = 0;

int r = n-1;

while(l<=r) {

int mid1 = l+(r-l)/3;

int mid2 = r-(r-l)/3;

if(nums[mid1]==k) {

return mid1;

}

else if(nums[mid2]==k) {

return mid2;

}

else if(nums[mid1]>k){

r = mid1-1;

}

else if(nums[mid2]<k) {

l = mid2+1;

}

else {

l = mid1+1;

r = mid2-1;

}

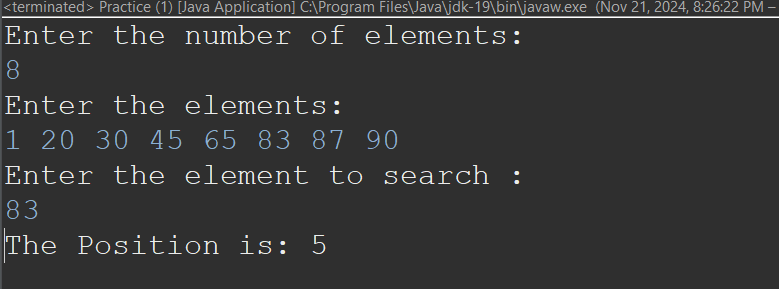
}

return -1;

}

}

Output:



Time Complexity: O(2\*log3 n)

1. **Interpolation Search**

Code:

import java.util.\*;

public class Practice {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

// Input

System.***out***.println("Enter the number of elements:");

int n = sc.nextInt();

// Input

int[] nums = new int[n];

System.***out***.println("Enter the elements (sorted in ascending order):");

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

nums[i] = sc.nextInt();

}

System.***out***.println("Enter the element to search:");

int k = sc.nextInt();

int result = *InterpolationSearch*(nums, k);

if (result != -1) {

System.***out***.println("The position is: " + result);

} else {

System.***out***.println("Element not found.");

}

}

private static int InterpolationSearch(int[] nums, int x) {

int n = nums.length;

int l = 0;

int r = n - 1;

while (l <= r && x >= nums[l] && x <= nums[r]) {

// Prevent division by zero

if (nums[l] == nums[r]) {

if (nums[l] == x) return l;

else break;

}

// Calculate the probable position

int pos = l + (int)((double)(r - l) / (nums[r] - nums[l]) \* (x - nums[l]));

// Check if the element is found

if (nums[pos] == x) {

return pos;

}

// Narrow down the search range

else if (nums[pos] > x) {

r = pos - 1;

} else {

l = pos + 1;

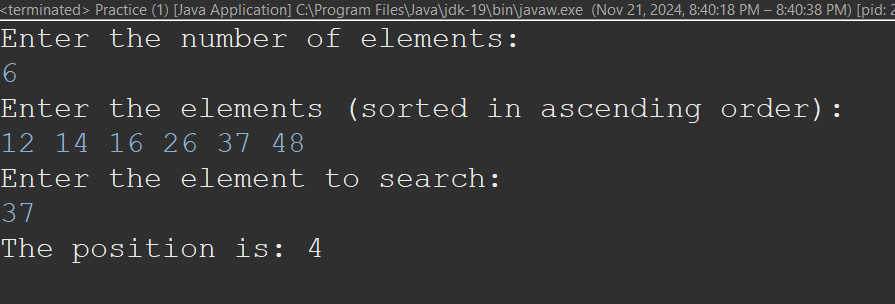
}

}

return -1; // Element not found

}

}

Output:  


Time Complexity: O(log2(log2(n))