


# Harsh Bhagat

**Test ID:** 432006079919720 |  9970422026 |  harshbhagat.23d@stvincentngp.edu.in

**Test Date:** August 14, 2024

## Automata

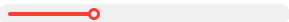
**31** /100




## Automata

 **31** / 100

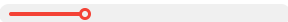
### Programming Ability

 **30** / 100

### Programming Practices

 **50** / 100

### Functional Correctness

 **26** / 100

## 1 | Introduction

### About the Report

This report provides a detailed analysis of the candidate's performance on different assessments. The tests for this job role were decided based on job analysis, O\*Net taxonomy mapping and/or criterion validity studies. The candidate's responses to these tests help construct a profile that reflects her/his likely performance level and achievement potential in the job role

This report has the following sections:

The **Summary** section provides an overall snapshot of the candidate's performance. It includes a graphical representation of the test scores and the subsection scores.

The **Response** section captures the response provided by the candidate. This section includes only those tests that require a subjective input from the candidate and are scored based on artificial intelligence and machine learning.

### Score Interpretation

All the test scores are on a scale of 0-100. All the tests except personality and behavioural evaluation provide absolute scores. The personality and behavioural tests provide a norm-referenced score and hence, are percentile scores. Throughout the report, the colour codes used are as follows:

- Scores between 67 and 100
- Scores between 33 and 67
- Scores between 0 and 33

## 2 | Response

### Automata



31 / 100

[Code Replay](#)

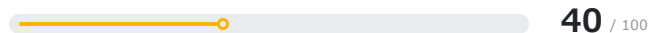
#### Question 1 (Language: C++)

You are playing an online game. In the game a list of N numbers is given. The player has to arrange the numbers so that all the odd numbers of the list come after the even numbers.

Write an algorithm to arrange the given list such that all the odd numbers of the list come after the even numbers.

#### Scores

##### Programming Ability



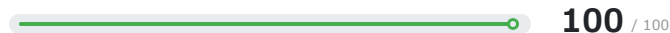
Emerging basic structure. Appropriate keywords and tokens present, showing some understanding of a part of the problem.

##### Functional Correctness



Partially correct basic functionality. The source code compiles and passes only some of the basic test cases. Some advanced or edge cases may randomly pass.

##### Programming Practices



High readability, high on program structure. The source code is readable and does not consist of any significant redundant/improper coding constructs.

#### Final Code Submitted

**Compilation Status: Pass**

```

1 // Header Files
2 #include<iostream>
3 #include<string>
4 #include<vector>
5 using namespace std;
6
7
8 /*
9 *
10 */
11 void funcArrange(int arr[],int inputArr_size,int answer[])
12 {
13
14
15     for(int i=0;i<inputArr_size;i++){
16         if(i%2==0){
17             answer[i] = arr[i];

```

#### Code Analysis

##### Average-case Time Complexity

**Candidate code:** Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

**Best case code:** O(N)

\*N represents size of the list.

##### Errors/Warnings

There are no errors in the candidate's code.

##### Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

```

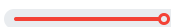
18     }
19 }
20
21
22 for(int i=0;i<inputArr_size;i++){
23     if(i%2!=0){
24         answer[i] = arr[i];
25     }
26 }
27
28 }
29
30 int main()
31 {
32
33     //input for inputArr
34     int inputArr_size;
35     cin >> inputArr_size;
36     int arr[inputArr_size];
37     int answer[inputArr_size];
38     for ( int idx = 0; idx < inputArr_size; idx++ )
39     {
40         cin>>arr[idx];
41     }
42
43     //output
44     funcArrange(arr,inputArr_size,answer);
45     for ( int i = 0; i <inputArr_size ; i++)
46     {
47         cout << answer[i]<< " ";
48     }
49
50     return 0;
51 }
52

```

## Test Case Execution

Passed TC: **33.33%**

Total score



4/12

**60%**

Basic(3/5)

**0%**

Advance(0/6)

**100%**

Edge(1/1)

## Compilation Statistics

9

Total attempts

3

Successful

6

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:30:48

Average time taken between two compile attempts:

00:03:25

Average test case pass percentage per compile:

11.11%

### Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

### Test Case Execution

There are three types of test-cases for every coding problem:

**Basic:** The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

**Advanced:** The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

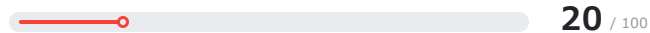
**Edge:** The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

## Question 2 (Language: C++)

You are given a list of integers and an integer  $K$ . Write an algorithm to find the number of elements in the list that are strictly less than  $K$ .

## Scores

### Programming Ability



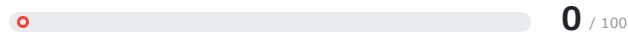
Code seems to be unrelated to the given problem.

### Functional Correctness



The source code does not pass any basic test cases. It is either due to incorrect logic or runtime errors. Some advanced or edge cases may randomly pass.

### Programming Practices



Programming practices score cannot be generated. This is because source code has syntax/runtime errors and is unparseable or the source code does not meet the minimum code-length specifications.

## Final Code Submitted

Compilation Status: Pass

```

1 // Header Files
2 #include<iostream>
3 #include<string>
4 #include<vector>
5 using namespace std;
6
7
8 /*
9  * element, representing the vector with size of element_size.
10  * num, representing the integer to be compared(K).
11  */
12 int noOfElement (vector<int> element, int num,int element_size)
13 {
14     int answer;
15     // Write your code here
16     for(int i=0;i<element_size;i++){
17         if(element.at(i)<num){
18             answer +=1;
19         }
20     }
21
22
23     return answer;
24 }
25
26 int main()
27 {
28
29     //input for element
30     int element_size;
```

## Code Analysis

### Average-case Time Complexity

**Candidate code:** Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

**Best case code:**  $O(N)$

\*N represents number of elements in the input array

### Errors/Warnings

There are no errors in the candidate's code.

### Structural Vulnerabilities and Errors

There are no errors in the candidate's code.

```

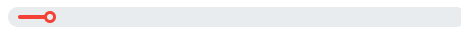
31 cin >> element_size;
32 vector<int> element;
33 for ( int idx = 0; idx < element_size; idx++ )
34 {
35     int temp;
36     cin >> temp;
37     element.push_back(temp);
38 }
39 //input for num
40 int num;
41 cin >> num;
42
43
44 int result = noOfElement(element, num,element_size);
45 cout << result;
46
47
48 return 0;
49 }
50

```

## Test Case Execution

Passed TC: **6.25%**

Total score

 1/16

**0%**

Basic(0/6)

**0%**

Advance(0/8)

**50%**

Edge(1/2)

## Compilation Statistics

**6**

Total attempts

**6**

Successful

**0**

Compilation errors

**0**

Sample failed

**0**

Timed out

**3**

Runtime errors

Response time:

**00:12:31**

Average time taken between two compile attempts:

**00:02:05**

Average test case pass percentage per compile:

**3.13%**

### Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

### Test Case Execution

There are three types of test-cases for every coding problem:

**Basic:** The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

**Advanced:** The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

**Edge:** The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code