VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

Digital Assignment - II, SEPTEMBER 2019

B.Tech, Fall-2019-2020

NAME	PRIYAL BHARDWAJ
REG. NO.	18BIT0272
COURSE CODE	ITE1005
COURSE NAME	Software Engineering Principles & Practices
SLOT	C2+TC2
FACULTY	Prof. SANTHOSH KUMAR S V N

Q.1 What are the problems in developing performance tests for a distributed system such as the online library management system?

Performance Testing of distributed systems require meticulous planning and methodical execution. Performance Testing falls into various categories like Load Testing, Stress Testing and Spike Testing to mention a few. Like any other procedures, Performance Testing also comes with a number of Challenges.

- **Selecting the Environment and the Testing Tools**

Most of the clients do not have dedicated environments for performance testing. Ideally tests should be conducted on real scenarios or the production environment. Because of practical problems or budgetary issues, it is not possible many times to conduct the tests in this environment.

It is ideal to create a replica of production environment. But due to budget constraints, the client may not be able to provide the environment. Hence the performance testers have to consider varying scenarios with the limited hardware resource availability in order to get the accurate results.

A simple example could be that it is difficult to test the scenario where thousands of users log in into an online library management system concurrently.

- **Offering a Complete Test Coverage**

It can only be a magician who can cover a performance testing script covering all the functionalities of an application. It is very important that the performance testing scenarios are indicative of various parameters.

It is not feasible to create a performance testing script that covers all the functionality of an application. After gathering various scenarios, the key functionalities to be automated are to be identified to ensure that most of the Use Cases are attended to. Automating the functionality of an application requires the consideration that a variety of end users would be using the system in their own contexts and estimate the user data accordingly.

Data and resource setup play an important role in performance testing. In terms of data, small and large numeric values can be entered, alphabets, words and sentences can also be considered in order to render varying results.

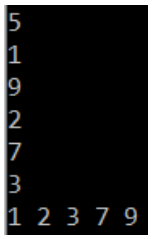
Q.2 Give three situations where the testing of all independent paths through a program may not detect program errors.

Situation 1:

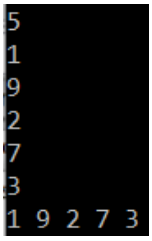
While writing a program for sorting given input of numbers, separate functions for sorting and display are created. The main goal of this program is to perform sorting and display the final sorted output only. Let sorting function call be A and display function call be B. In the main function if the programmer executes the statements in the order <A B> or <B A>, the program is compiled successfully in both cases. But in <A B> we get the sorted output and in <B A> we get input array as it is. So path testing will only check if each path is error free or not but it can detect these errors.

```
1   #include<stdio.h>
2   void sort(int a[], int n)
3   {
4       int temp;
5       for(int i=0;i<n;i++)
6       {
7           for(int j=0;j<n-i-1;j++)
8           {
9               if(a[j+1]<a[j])
10              {
11                  temp=a[j];
12                  a[j]=a[j+1];
13                  a[j+1]=temp;
14              }
15          }
16      }
17  }
18  void display(int a[], int n)
19  {
20      for(int i=0;i<n;i++)
21          printf("%d ",a[i]);
22  }
23  int main()
24  {
25      int n;
26      scanf("%d",&n);
27      int a[n];
28      for(int i=0;i<n;i++)
29          scanf("%d",&a[i]);
30      display(a,n);
31      sort(a,n);
32      return 0;
33  }
```

Expected output:



Actual output:



Situation 2:

Consider the abs function, that returns the absolute value of a number. Here is a test (Python, imagine some test framework):

```
def test_abs_of_neg_number_returns_positive():
  assert abs(-3) == 3
```

This implementation is correct, but it only gets 60% code coverage:

```
def absolute(x):
    if x < 0:
        return -x
    else:
        return x
x=int(input())
print(absolute(x))
```

This implementation is wrong, but it gets 100% code coverage:

```
def absolute(x):
    return -x
x=int(input())
print(absolute(x))
```

Situation 3:

Here's a simpler example to round things off. Consider the following sorting algorithm (in Java):

```
int[] sort(int[] x) { return new int[] { x[0] }; }
```

Now, let's test:

```
sort(new int[] { 0xCAFEBABA });
```

Now, consider that (A) this particular call to sort returns the correct result, (B) all code paths have been covered by this test.
But, obviously, the program does not actually sort.
It follows that coverage of all code paths is not sufficient to guarantee that the program has no bugs. Hence path testing does not detect program error in this case either.

*****