KloverCloud Online Assessment

Question 1:

In object oriented programming we avoid copying and pasting of code for simple redundant tasks by describing objects. It facilitates easier understanding, modification and extension of the code in complex or large scale production environments. The basic concepts of the OOP include encapsulation, abstraction, inheritance and polymorphism.

In the given task we are going to use the concepts of oop to describe circle and rectangle. So, circles and rectangles are shapes that can be described by their properties like length, radius, color, area, perimeter etc.

For **encapsulation** we create circle and rectangle object with their respective property and methods:

```
class  Circle():
        def __init__(radius, color):
              self.radius = radius
              self.color = color


    class Rectangle():
          def __init__(height, base, color):
                self.height = height
                self.base = base
                self.color = color
```

Abstraction hides the complexity of the internal functions in the code from the user. In this way we use an abstract class as the base class that can act as the basic structure for the concrete classes in the code.

```
   abstract class Shape():
         def shapetype(self):
               pass
```

```
class Rectangle(Shape):
      def shapetype(self):
            print("This is a rectangle")

class Circle(Shape):
      def shapetype(self):
            print("This is a triangle")
```

In object oriented programming the term polymorphism is used to describe the ability of an object to have multiple forms, to show multiple characteristics without having to describe the object each time it achieves a different property.

In **inheritance** we reduce the redundant use of the code by having the properties that are in common defined in a generic object rather than having them defined in each object.

```
class Shape():
      def __init__(self,color):
            self.color = color


  class Circle(Shape):
        def __init__(self, radius):
              self.radius = radius
```

Here the class circle inherits the property of color from the parent Shape class and so can the rectangle.

In **polymorphism** we define a particular part of an object that is common to the other object to be executed according to the object.

```
  class Shape:
    def area(self)

class Circle(Shape):
      def __init__(radius, color):
          self.radius = radius

      def area(self):
            return pi*self.radius*self.radius
```

```
class Rectangle(Shape):
      def __init__(height, base, color):
            self.height = height
            self.base = base

      def area(self):
      return self.length * self.base
```

In object oriented programming the term polymorphism is used to describe the ability of an object to have multiple forms, to show multiple characteristics without having to describe the object each time it achieves a different property.

Question 2:

The variables that we describe in our program are stored in the RAM of the system. When we declare a variable, our computer allocates memory in the heap for our variable and in the stack a reference is created to that object.

For example, if we declare a string variable name = “Arefin” then, the value “Arefin” of the variable is stored in the heap and the memory address to the value of the “Arefin” is stored in the stack. Therefore, if we say for example create another variable with the same value the same address to that heap will be stored in the stack for the second variable.

A stack follows LIFO order and memory allocation in the stack lasts as long as the execution continues. Whereas, heap dynamically allocates memory blocks and a garbage collector is required to clear the unused objects of the program in the heap.

Provided with a garbage collector and manual cleaning up of the memory, allocating memory on a heap dynamically for the array is the best as it allows for allocating larger memories. In this process memory has to be explicitly freed to avoid a memory leak.

Question 3:

Parallelism can be used to achieve faster computation of element-wise multiplication of large matrices.Particularly, GPU’s are a lot more capable in parallel computations than CPUs. CPU’s are constructed for operation of a large variety of tasks due to the wider spectrum of its instruction set while GPUs are task specific and can achieve high performance in tasks that are repetitive like the element-wise matrix multiplication in the given example.

So, I conclude that incorporating some powerful gaming GPUs to the system will drastically improve the execution of the multiplication task.

Question 4:

In this example of tree traversal a recursive method has been used to traverse a tree that is humongous in size and heavily left-skewed. The problem is that, in the recursive process until the base case is recognized there is no convergence. In the given code we traverse both left and right nodes using the recursion for the million nodes recursively even though the tree is left skewed. Therefore, in this case of a huge binary tree, the space and time requirement for the program is greater than that of the iterative process.
So due to the nature of the binary tree being very large and also left skewed there is no need to use recursion which is overwhelming in this case due to the overhead. It can lead to a crash, a stack overflow.
So we will use an iterative approach to count the number of nodes in a tree. We can use a stack in place of the recursion.

```
struct Node{
    Node* left;
    Node* right;
}
int traverse(Node* node){
    stack<Node *> s;
    if (node == null) return 0; //base case
    while (node != NULL || s.empty() == false){
       while (node !=  NULL)
        {
            s.push(node);
            node = node->left; //traversing left
        }
        node = s.top();
        s.pop();

        count++

        node = node->right; // traversing right
    }
    return count;
}
```