Solution to Question -01

I tried to explain the concept of OOP using a python code sample attached below:

```
class Pattern:
    def __init__(self,  value1, value2):
        self.__value1 = value1
        self.__value2 = value2


    def calculate_area(self):
        self.__area = self.__value1 * self.__value2

    def get_area(self):
        return self.__area


class Circle(Pattern):
    def __init__(self, radious):
        super(Circle, self).__init__(radious, radious)

    def get_area(self):
        return 3.1416 * super(Circle, self).get_area()


class Rectangle(Pattern):
    def __init__(self, height, width):
        super(Rectangle, self).__init__(height, width)



circle = Circle(4)
rectangle = Rectangle(3,4)
circle.calculate_area()
print(circle.get_area())
rectangle.calculate_area()
print(rectangle.get_area())
```

I have implemented a **Pattern** class which is the template or blueprint for Circle and Rectangle class. All the attributes and methods are already constructed in this class. The variables with "__" are private variable which cannot be accessible from outside the class limit.

The Circle and Rectangle class inherited the Pattern class for accessing all the functionalities and attributes of this parent class.

The constructor for Circle class accepts only a single value but Pattern class takes two arguments for which we override the constructor and rewrite the code template for this portion. In the same way we override the get_area() method in order to execute some changes in the method which is known as **polymorphism.** I have used concepts like Inheritance, encapsulation, polymorphism/method overriding which are core parts of OOP.

Finally, in the driver code, I have created circle and rectangular classes to test all the functionalities of Object Oriented Programming.

Solution to Question -02

Stack is mainly a linear data structure which is used for static memory allocation. The variables in the stack are directly stored in the memory/ram for which it is faster option to access this memory. The Limit of stack size depends on OS. The memory allocation of stack depends on the compilation of the program. Stack works in basis of Last In Fast Out order which means the latest reserved block memory is always ready to be freed. The LIFO methodology of stack limits the access the elements randomly. Stack can be used when we know the size of data which we are going to allocate and it is not so big.

The variables allocated in the heap used to allocate memory on the runtime and access to this memory is slower compared to stack. Heap has not specific limit on memory size but it can be limited by the size of virtual memory. Element of the heap can be accessed randomly as there is no dependencies of the elements with each other for which we can allocate a block any time and also free it at any time.

If I want to allocate memory for a very large array(100MB) for which I would like to use **Heap** for the memory allocation**.** As we have to allocate a large block of memory for array, it will be better to use heap because the memory allocation has no limit.

On the contrary, stack has a limitation of memory allocation for which it may increase the rate of stack overflow for too many objects.

Solution to Question -03

Yes, we can improve the speed a little bit. According to a research on Ryzen CPUs, it is found matrix multiplication on mat lab runs on AMD with the Intel Math Kernel Library (Intel MKL).But AMD performance can be greatly enhanced by running a script to command mat lab to run AMD processors in AVX2 mode. In this way the performance gains were said to be between 20% - 300% by doing the change.

Solution to Question -04

From my point of view, the algorithm should work well but it will take long time to execute. At first the algorithm will search for the height to get leaf nodes which takes O(log(N)) complexity. In the same way

it takes O(log(N)) time complexity to recursively count the nodes. In total, the time complexity for this approach will be O(log(N)^2) which is a huge time for million size nodes.