Question_1:

Ans: OOP circulates around creating objects which contain both code and data . Object is basically an abstract data type which contains properties, method, data. It is created by class which can be compared as program code template. By using class and object, oop provides four main principles which are abstraction, encapsulation, inheritance, and polymorphism.

Lets take a circle and a rectangle as example. A circle and rectangle both have some unique features, data like a circle has radius where a rectangle has width and height. They also have some common feature like both of them have area though calculating the area will be different. So for circle and a rectangle two class can be defined which will contain general details about circle and rectangle and by using these class we can create different specific radius of circle. Also we can create rectangle of many sizes. These are called the objects.

```
abstract class Shape{
        private String color ;
        public abstract double calculateArea();
        void setColor(String color){
                this.color = color;
        }
        String getColor(){
                return this.color;
        }
}
```

```
class Circle extends Shape {
	private double radius;
	private double area;
	public Circle(double radius){
		this.radius = radius;
	}
	protected void calculateArea(){
		this.area = pi*radius*radius;
	}
	void getArea(){
		return this.area;
	}
}
class Rectangle extends Shape {
	private double length, width;
	private double area;
	public Rectangle (double length, double width){
		this.length = length;
		this.width = width;
	}
	protected void calculateArea(){
		this.area = length*width;
	}
	void getArea(){
		return this.area;
	}
```

```
}
public class App{
public static void main(String args[]){
        Shape smallCircle = new Circle(.5);
        Shape largeCircle = new Circle(11);
        Shape square = new Rectangle(5,5);

        smallCircle.setColor = “Blue”;
        square.setColor = “White”;

        smallCircle.calculateArea();
        square.calculateArea();
        }
}
```

Encapsulation: In circle class we encapsulated the area variable as access modifier private was imposed. GetArea method bundled data with method.

Inheritance: Both circle and rectangle have common feature which is is their color. It was defined in another class Shape which can be seen as common details container for circles and rectangles. Both circle and rectangle inherited these common features.

Abstraction: Hiding implantation details. Here area calculation of circle and rectangle is different. Their implementation was in their class. But we calculated them using their super abstract class method. By doing this we got abstraction in OOP.

Polymorphism: We can see many forms of calculateArea. One is in circle class. Another in rectangle class.

Runtime Polymorphism: Suppose there is an implementation of calculateArea method in Shape class. Then when we called square.calculateArea() in main method, oop compiler will automatically finds out which method to run. Here implantation in Square class will be compiled. This is called runtime polymorphism.

Question_2:

Ans: Stack and heap memory both are stored in computer ram. Stack is used to store the reference of an object where heap is used to store the object. In a multi-threaded application, each thread will have its own stack. But, all the different threads will share the same heap. So, data on the stack is automatically deallocated when variables go out of scope but data stored on the heap has to be deleted manually or by garbage collector on high level languages. Stack-memory is contiguous and has less storage space as compared to Heap-memory. As a result, stack is set to a fixed size, and can not grow past it's fixed size. So, if there is not enough room on the stack memory, an error stack overflow occurs. So for a very large array I would definitely prefer heap memory as it is large, non-contiguous and allows dynamic allocation.

Question_3:

Ans: Element wise multiplication process can be sped up. We do not need to iterate through the whole matrix because in every iteration , we can find multiplication result od two indexes. One is result of index [i][j] and another is index [j][i]. So our iteration steps become half as a result j is initialized equal to i in the second nested loop.

```
for (i = 0 ; i<Length; i++){
    for(j = i ; j< width ; j++){
        ans[i][j] = A[i][j] * B[i][j];
        ans[j][i] = A[j][i] * B[j][i];
    }
}
```

Question_4:

Ans: The tree is huge and heavily left skewed. So in order to traverse in this we need to avoid depth first search approach. We also need to avoid recursive approach. As it has millions of nodes and heavily left skewed, this code will recursively go to the child node function which will most likely result in stack overflow due to the large number of functions in the stack

memory. Also avoiding depth first approach, going with breadth first approach can decrease the time complexity as it is heavily left skewed. Basically, we will do level order traversal in this tree. We will need queue data structure for this.

```
struct Node{
	Node* left;
	Node* right;
}

int traverse(Node* node){
	if (node == NULL) return 0;
	queue<Node *>q;
	q.push(node);

	while (q.empty() == false){
		Node *temp =q.front();
		cunt++;
		q.pop();
		if(temp->left!= NULL){
			q.push(temp->left);
		}
		if(temp->right!= NULL){
			q.push(temp->right);
		}
	   }
	return cunt ;
}
```

As the queue is left skewed, only a few nodes are in the queue at any given time. So, the space complexity can be considered as O(1). Time complexity is O(n).