1. Ans:

```
#include<iostream>
using namespace std;

class Shape
{
public:
   virtual void Area() = 0;
};

class Circle : public Shape
{
   float radius;
public:
   Circle(float radius)
   {
      this->radius = radius;
   }

   void Area()
   {
      cout << "Area of circle: " << (3.1416 * radius * radius) << endl;
   }
};

class Rectangle : public Shape
{
```

```
    float length;
    float width;
public:
    Rectangle(float length, float width)
    {
        this->length = length;
        this->width = width;
    }

    void Area()
    {
        cout << "Area of rectangle: " << (length * width) << endl;
    }
};

void callArea(Shape *obj)
{
    // To achieve runtime polymorphism
    obj->Area();
}

int main()
{
    Circle c(2.3);
    callArea(&c);

    Rectangle r(1.5, 6.5);
```

```
    callArea(&r);


    return 0;
}
```

2. Ans:

Stack Memory: A stack is a special type of memory that generally stores temporary variables created by a function. Stack follows LIFO order. It stores items that have a very short life such as methods, variables and reference variables of the objects. Memory allocated to stack is available until the function returns. If there is no space for creating the new objects, it throws the “StackOverflowError” error.

Heap Memory: Heap is a hierarchical data structure that provides larger memory than stack. It also supports dynamic memory allocation. Heap memory is used by the application as long as the application runs. It is slower than stack and more costly as well.

If I need to use a very large array I would make use of heap memory. Because, it is larger in size, supports dynamic memory allocation. Also it’ll cost more if I need to initialize the big size array multiple times in my application scope. I want to initialize it once and use it throughout the application.

3. Ans:

We can use the numpy library of python. It takes fairly less time for element-wise matrix multiplication.

4. Ans:

```
int traverseIt(Node *root)
{
   int leftside, rightside;

   if(root == NULL)
      return 0;
   else
   {
      leftside = traverseIt(root->left);
      rightside = traverseIt(root->right);
   }

   return leftside+rightside+1;
}
```