Task - Data Structures and OOPS concept.

1. How do you find a duplicate integer in a given array ?

Code:-

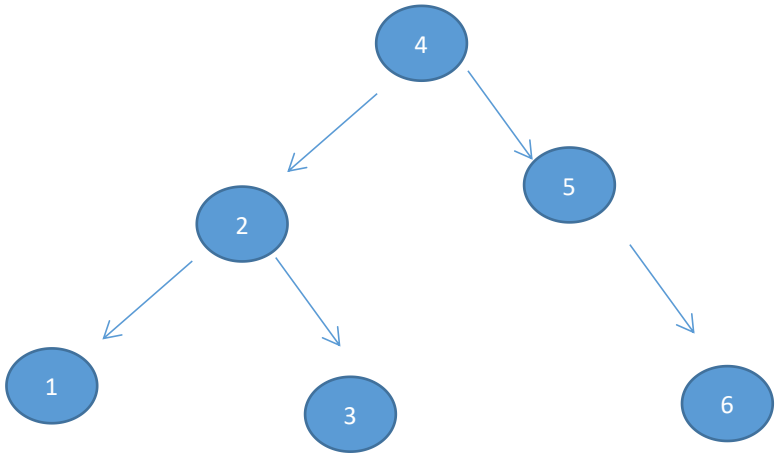
```
#include<bits/stdc++.h>
using namespace std;

int main() {
	int arr[] = {1 , 2 , 3 , 4 , 2 };
	int n = sizeof(arr) / sizeof(int);
	//first
	int cnt[*max_element(arr , arr + n) + 1] = {0};
	for(int i = 0 ; i < n ; i++) {
		cnt[arr[i]]++;
		if(cnt[arr[i]] == 2) {
			cout << "The repeating integer is : " <<  arr[i] << endl;
			break;
		}
	}
	//second
	map<int , int> mp;
	for(int i = 0 ; i < n ; i++) {
		if(mp[arr[i]] == 1) {
			cout << "The repeating integer is : " <<  arr[i] << endl;
			break;
		}
		mp[arr[i]] = 1;
	}
}
```

2. Explain how a binary search tree is implemented.
Ans:-

A binary search tree is used to find a element quickly in a sorted set of elements.
The structure of the binary search tree is implemented as:-

1. A root element is choosen
2. In the left subtree of the root all the elements which are lesser than the root element are placed.
3. In the right subtree of the root all the elements which are greater than the root element are placed.
4. This happens recursively on the subtrees itself.

For example :
For arr[] = {1 , 4 ,5 ,2 , 3 ,6 }
The tree would look like :-



3) How will you check if two strings are anagrams of each other ?
Code:-

```
#include<bits/stdc++.h>
using namespace std;

int main() {
	string s1 = "ABCD" , s2 = "DCAB";
	sort(s1.begin() , s1.end());
	sort(s2.begin() , s2.end());
	if(s1 == s2) {
		cout << "These are anagrams of each other." << endl;
	}
	else{
		cout << "These are not anagrams of each other." << endl;
	}
}
```

4) What is polymorphism and why is it required ? Provide examples

Ans. Polymorphism is a concept where an entity can be of many forms such as same function can be overloaded or overridden by child classes in OOPS.

There are two types of polymorphism :-
1. Compile Time
2. Runtime

Compile time polymorphism is done by overloading function with the same name as in base class but with different number of arguments.
So what happens is when we create an object of child class and calls the overloaded function compiler presumes that we are referring to the one implemented in child class.

Runtime polymorphism is done by overridding the base class function in child class , in case of runtime polymorphism compiler decides which function to be called the one which is implemented in child class or the one in the base class by looking at the type of the object during runtime.

It is required as it provides Reusablity of code of the base class and also makes debugging less cumbersome.

Example :-

Compile Time Polymorphism

```
class Car {
	public :
		int gear;
		void shiftGear() {
			gear++;
		}
};

class ferrari : public Car {
	public :
	int gear;
	void shiftGear() {
		gear += 2;
	}
};
```

```
int main() {
	ferrari f1;
	// the one implemented in ferrari class will be called
	f1.shiftGear();
}
```

Runtime Polymorphism

```
class Car {
	public :
		int gear;
		virtual void shiftGear() {
			gear++;
		}
};

class ferrari : public Car {
	public :
	int gear;
	void shiftGear() {
		gear += 2;
	}
};

int main() {
	ferrari f1;
	Car* c1 = f1;
	// the one implemented in the base class will run
	c1.shiftGear();
}
```