Offline Assignment 2

1. The following recursive code finds out the maximum value from an array of integers. **[4]**

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

int max_elm(int arr[], int sz){

    ///if the array contains only 1 element then that element is the maximum itself
    if(sz==1){
        return arr[0];
    }
    else{
        ///keeping the last array element for myself
        int mypart=arr[sz-1];

        ///forwarding the rest of the array elements to my friend to search for maximum element
        int friendmax=max_elm(arr,sz-1);

        if(mypart>friendmax){
            return mypart;
        }
        else{
            return friendmax;
        }
    }
}

int main()
{
    int arr[]={1,3,-100,1000, 999};
    int sz=sizeof(arr)/sizeof(int);

    cout<<"The maximum is: "<<max_elm(arr,sz)<<endl;

    return 0;
}
```

Now first try to understand the above problem and its solution, and then
write a C/C++ code that will take input an integer and will return the maximum digit found within that integer.

Sample Input	Sample Output
1956	9
2410	4

2. The following recursive code checks whether a string is palindrome or not. **[4]**

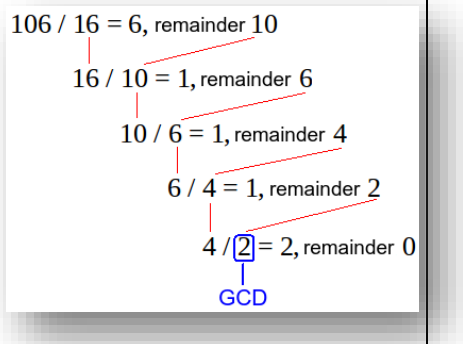
```
#include <iostream>
#include <string>
using namespace std;

bool is_palindrome(string s, int tracker){
    int first_ind=tracker;
    int last_ind=s.size()-1-tracker;

    ///if the first index crosses the last index then we have nothing to check i.e. default palindrome
    if(first_ind>last_ind){
        return true;
    }
    else{
        ///keeping the first and last characters for myself
        char firstchar=s[first_ind];
        char lastchar=s[last_ind];
        ///forwarding the remaining intermediate portion to my friend
        bool friend_decision=is_palindrome(s,tracker+1);

        if(friend_decision==true && firstchar==lastchar){
            return true;
        }
        else{
            return false;
        }
    }

}

int main()
{
    string s1="madam", s2="abda";

    cout<< is_palindrome(s1, 0) <<endl;
    cout<< is_palindrome(s2, 0) <<endl;

    return 0;
}
```

Now first try to understand the above problem and its solution, and then
write a C/C++ code that will take input an array of integers and will check whether the array is palindrome or not.

Sample Input	Sample Output
{1,2,3,2,1}	1
{1,0,1,0}	0

3. The following recursive code finds out the GCD of two integers. **[4]**

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

int two_gcd(int dividend, int divisor){
    ///if the divisor is zero then the dividend is the gcd solution
    if(divisor==0){
        return dividend;
    }
    else{
        ///calculating the new dividend and divisor value
        int new_dividend=divisor;
        int new_divisor=dividend%divisor;

        ///forwarding the new dividend and new divisor to a friend for gcd calculation
        int friend_gcd=two_gcd(new_dividend,new_divisor);

        return friend_gcd;
    }
}

int main()
{
    cout << two_gcd(17,13) << endl;
    cout << two_gcd(25,15) << endl;
    return 0;
}
```



Now first try to understand the above problem and its solution.

We can calculate the GCD of **n** integers as follow:

$$GCD(a_1, a_2, a_3, a_4, \dots, a_n) = GCD(\quad GCD(a_1, a_2, a_3, a_4, \dots, a_{n-1}), \quad a_n\ )$$

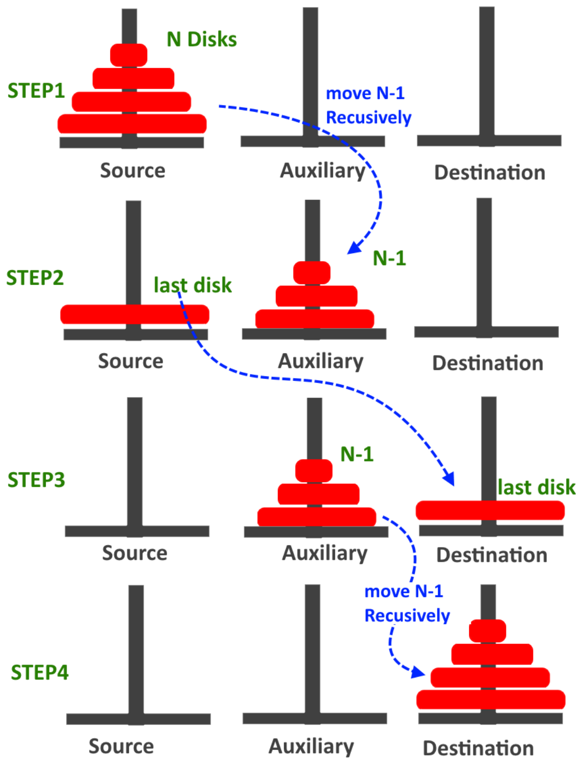
meaning if we know the GCD of the remaining (n-1) integers then the GCD of all the n integers is the GCD between the last integer (a_n) and the GCD of remaining (n-1) integers.

So, each time we will keep the last integer (a_n) for ourself and ask our friend to find out the GCD of the remaining (n-1) integers. Our final solution will be the GCD between our last integer (a_n) and our friends returned result. We can directly calculate the GCD without any recursive call when the array contains only 2 elements.

Code template: implement the n_gcd() function according to the comments. Remember, the dividend value must be greater than the divisor value in two_gcd() function.

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

int two_gcd(int dividend, int divisor){
    ///if the divisor is zero then the dividend is the gcd solution
    if(divisor==0){
        return dividend;
    }
    else{
        ///calculating the new dividend and divisor value
        int new_dividend=divisor;
        int new_divisor=dividend%divisor;

        ///forwarding the new dividend and divisor to a friend for gcd calculation
        int friend_gcd=two_gcd(new_dividend,new_divisor);

        return friend_gcd;
    }
}

///implement this function
int n_gcd(int arr[], int sz){
    ///if array size is 2 then directly calculate the gcd between arr[0] and arr[1]
    ///call two_gcd() to calculate the gcd of two integers
    if(...){

    }
    else{
        ///reserve the last array integer for yourself

        ///call your friend to find out the remaining array integers gcd value


        ///your gcd result is the gcd between your reserved integer and your friend gcd result
        ///call two_gcd() for calculating the gcd value between your integer and friend result

        ///return the calculated gcd value
    }
}

int main()
{
    int arr[]={8,6,4,2}
    int sz=sizeof(arr)/sizeof(int);

    cout<< n_gcd(arr,sz) <<endl; ///output will be: 2

    return 0;
}
```

4. The following recursive code calculates the number of moves necessary for Tower of Hanoi problem with 1 intermediate pole. **[4]**
How TOH works:



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

int TOH(int n, char src, char dest, char tmp){
    ///if you have only 1 disk left then directly move the disk from src to dest i.e. 1 move required
    if(n==1){
        return 1;
    }
    else{
        ///first friend will move the top n-1 disks from src to tmp
        int friend1moves=TOH(n-1, src, tmp, dest);

        ///then I will move the remaining last 1 disk from src to dest
        int mymove=TOH(1,src,dest,tmp);

        ///then second friend will move back the n-1 disks from tmp to dest pole
        int friend2moves=TOH(n-1,tmp,dest,src);


        ///so total moves required
        int totalmoves=friend1moves+mymove+friend2moves;

        return totalmoves;
    }
}

int main()
{
    cout << TOH(6, 'S', 'D', 'T') << endl;
    return 0;
}
```

Now first try to understand the above problem and its solution.
Write a C/C++ code that will implement the Tower of Hanoi problem but with 2 intermediate poles. Working mechanism is described below:

The given poles are: src_pole, dest_pole, tmp_pole1, tmp_pole2

First, you will move the top (n-2) disks from src_pole to tmp_pole1
then, you will move the next 1 disk from src_pole to tmp_pole2
then, you will move the last 1 disk from src_pole to dest_pole
then, you will move back the 1 disk from tmp_pole2 to dest_pole
then, you will move back all the (n-2) disks from tmp_pole1 to dest_pole.

5. Suppose you are given a distance of n units and you can move either 1 unit/2 units/3 units at a time. You need to find out the total number of ways you can cover that distance. **[4]**

Hints:

- If you choose 1 unit to cover in 1 way, then forward the rest (n-1) units to your friends.
  Total no of ways = 1 * friend_ways
- If you choose 2 units to cover in 1 way, then forward the rest (n-2) units to your friends.
  Total no of ways = 1 * friend_ways
- If you choose 3 units to cover in 1 way, then forward the rest (n-3) units to your friends.
  Total no of ways = 1 * friend_ways

Finally, you can cover the whole distance by summing all the results you have achieved in every ways.

If the distance is 0, then you will know that you have finished covering that distance in 1 way.
If the distance is negative, then you will know that you can't cover that distance so return 0 way.

Sample Input	**Sample Output**
3	4
4	7

[Miss at your own risk]
[Any kind of plagiarism is strictly prohibited.]