Ronald Chatelier

1. Write code implementing Selection Sort. Your program should let the user enter how large of an array they wish to sort, followed by the values they wish to be in the array. It should print the unsorted and sorted array. Start with the bubblesort example from class/notes and replace the bubblesort function with a select sort function.

```
proceedure SelectionSort(array a, length(A) = n)
    for i in 0 to n - 2
        maxIndex = i
        for j in (i + 1) to (n - 1)
            if a[j] > A[maxIndex]
                maxIndex = j
        tmp = A[i]
        A[i] = A[maxIndex]
        A[maxIndex] = tmp
```

```cpp
#include <iostream>

int main()

{

    int size;

    int maxIndex, j, i, temp, max;

    int* array = NULL;



    // the size for the array

    std::cout << "Array's size: ";

    std::cin >> size;



    array = new int[size];
```

```cpp
// elements picked for the array
std::cout << "Elements in array: ";
for (int i = 0; i < size; i++)
{
    std::cin >> array[i];
}


// results for unsorted array
std::cout << "Unsorted array: ";
for (int i = 0; i < size; i++)
{
    std::cout << array[i] << " ";
}


// selection sort
for (int i = 0; i < size - 1; i++)
{
    maxIndex = i;
    max = array[i];
    for (int j = i + 1; j < size; j++)
```

```
        {
            if (max > array[j])
            {
                max = array[j];
                maxIndex = j;
            }
        }
        temp = array[i];
        array[i] = array[maxIndex];
        array[maxIndex] = temp;
    }
    // results for sorted array
    std::cout << std::endl << "Sorted array: ";
    for (int i = 0; i < size; i++)
    {
        std::cout << array[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

2. Extended Euclidian Algorithm (page 4 of last lecture, more on algorithms) Write code that asks for and gets two integers, then computes and displays their greatest common divisor using the Extended Euclidian Algorithm (EEA). The EEA should be implemented as a function that takes two integers as arguments and prints their GCD. Define 3 temp values to fix the following code if it doesn't work!

```
proceedure EEA(int: a, b)
        s = 0
        old_s = 1
        t = 1
        old_t = 0
        r = b
        old_r = a
        while r != 0
                q = old_r div r
                (old_r , r)    (r , old_r - q * r)
                (old_s , s)    (s , old_s - q * s)
                (old_t , t)    (t , old_t - q * t)
        print("GCD=" old_r)
        return (t,s)
```

```cpp
#include<iostream>

// EEA: Extended Euclidian Algorithm

int EEA(int a, int b) {



    int s = 0;

    int old_s = 1;

    int t = 1;

    int old_t = 0;
int r = b;

int old_r = a;



while (r != 0) {



    int q = (old_r / r);

    int temp1 = old_r;

    int temp2 = old_s;

    int temp3 = old_t;



    old_r = r;

    r = (temp1 - q * r);

    old_s = s;

    s = temp2 - q * s;

    old_t = t;

    t = temp3 - q * t;

}

// GCD:

std::cout << "GCD: " << old_r << std::endl;
```

```
    return (t,s);

}



int main()

{

    //Inside main function,take two integer type variables a and b

    int a;

    int b;

    //Ask user to enter a value for a

    std::cout << "First value: ";

    std::cin >> a;

    //Ask user to enter a value for b

    std::cout << "Second value: ";

    std::cin >> b;

    //Call EEA() function for a and b

    EEA(a, b);

    return 0;

}
```

3. Finding 2 largest numbers in a list (page 5 of last lecture, more on algorithms)

TwoLargest should be written as a function that is called in main. The user should be able to enter any list of numbers that they wish. **Hint**: You can use parts of the code from the bubblesort example such as vector_get() to create the list of numbers. Additionally, length(A) = n can also be found with A.size() in C++ so TwoLargest can be written with only one argument.

```
proceedure TwoLargest(A = [a1...an], length(A) = n)
        large_1 = 0
        large_2 = 0
        for i = 1 to n
                if A[i] > large_1
                        large_2 = large_1
                        large_1 = A[i]
                else if large_2 < A[i]
                        large2 = A[i]
```

```cpp
#include <iostream>
#include <vector>

void TwoLargest(int array[], int n)

{
    int large_1 = array[0];
    int large_2 = array[0];

    for (int i = 1; i < n; i++)

    {
        if (array[i] > large_1)

        {

            large_2 = large_1;
            large_1 = array[i];

        }

        else if (large_2 < array[i])
            large_2 = array[i];
    }
    std::cout << "The two largest numbers: " << large_1 << "," << large_2 <<
std::endl;

}

int main()

{

    int size;

    std::vector<int> v;
    std::cout << "List's size : ";
    std::cin >> size;
    int array[size];
    std::cout << "Numbers in list : ";

    for (int i = 0; i < size; i++)

        std::cin >> array[i];

    TwoLargest(array, sizeof(array) / sizeof(array[0]));

    return 0;

}
```

4. Write a program where you implement Modular Exponentiation (ME) using the square and multiply approach as a function which is called in main. ME calculates a^k mod n. The program should get values for a, k and n from the user. This code requires two steps. First k must be converted to a binary representation K consisting of a list of 0s and 1s. Second, Modular Exponentiation must be performed using a, n and K[] as arguments.

procedure BinaryK(k)

```
    K = empty list //hint: make K a vector
    tmp = k
    i = 0
    while tmp > 0
        add tmp mod 2 to K //hint: use pushback
        tmp = (tmp-K[i])/2
        i++
    return K
```

```
procedure ModularExpo(a, K, n)
    if n = 1
        return 0
    b = 1
    if K = 0
        return b
    A = a
    if K[0] = 1
        b = a
    for i = 1 to length(K)-1
        A = A*A mod n
        if K[i] = 1
            b = A*b mod n
    return b
```

```cpp
#include <iostream>

#include <string>

#include <vector>



std::vector<int> BinaryK(int k)

{

    std::vector<int> K;

    int tmp = k;

    int i = 0;
    while (tmp > 0)

    {

        K.push_back(tmp % 2);

        tmp = (tmp - K[i]) / 2;

        i++;

    }


    return K;

}


// Modular Exponentiation (ME) calculation

int ModularExpo(int a, std::vector<int> K, int n, int k)

{

    int b;

    int i;

    int A;


    if (n == 1)

        return 0;

    b = 1;

    if (k == 0)

        return b;

    A = a;
    if (K[0] == 1)
        b = a;

    for (i = 1; i <= K.size() - 1; i++)
    {

        A = (A * A) % n;

        if (K[i] == 1)
            b = (A * b) % n;
    }

    return b;
}

//Main function
int main()
{
    std::vector<int> K;
    int a;
    int k;
    int n;
    int i;
```

```cpp
    int result;


    std::cout << "Value a: ";

    std::cin >> a;

    std::cout << "Value k: ";

    std::cin >> k;

    std::cout << "Value n: ";

    std::cin >> n;


    K = BinaryK(k);

    result = ModularExpo(a, K, n, k);

    std::cout << "Results: " << result;

    return 0;

}
```