Pre-Lab Part 1

1)

Since it says any number, it is the same process because in the computer, any number base still represents the same number, just a different way of looking at it. But the value is the same. So a prime number is still a prime number regardless of its representation.

2)

These functions calculate the numbers until it is equal or greater than the input. Does not restart the sequence, but saves it in the static ints.

Fibonacci (int n)

```
Static Int first;
Static int second;
Static Int fib;
If n == first or n == second{
        Return n;
}
while (fibonacci < number){
        Fib = first + second;
        First = second;
        Second = sum;
}
return sum;
```

Lucas (int n)

```
Static int first;
Static int second
Static int luc
If n == first or n == second{
        Return n
}
while (lucas < number){
        Luc = first + second
        First = second;
        Second = luc
}
return luc
```

Mersenne (int n)

```
Static Int mers;
If n == mers{
        Return mers;
}
While (mers < n){
        Mers *= n
        Mers += 1
}
return mers;
```

Pre-Lab Part 2

1)

Bv_create (bit_len){

```
Bitvector bitvector = malloc(size of bitvector struct);
if (bitvector == null){
        Printf error;
        Return null
}
bitvector->vector = malloc(bit len / 8 + 1 * sizeofuint8);
If (bitvector == null){
        Printf error;
        Return null;
}
bitvector->length = bit_len
Return bitvector
}
```

bv_delete(bitvector v){

```
free(v->vector);
free(v);
}
```

bv_get_len(bitvector v){

```
Return v->length;
}
```

bv_set_bit(bitvector v, int i){

```
        Setter = 10000000;
        Setter = setter >> (i % 8);
        v->vector[i/8] | setter
}
bv_clr_bit(bitvector v, int i){
        Clearer = 10000000;
        Clearer = clearer >> (i % 8);
        Invert clearer;
        v->vector[i/8] & clearer
}

bv_get_bit(bitvector v, int i){
        Getter = 10000000;
        Getter = getter >> (i % 8);
        If (v->vector[i/8] & getter is greater than 0){
                Return 1;
        }else{
                Return 0;
        }
}

bv_set_all_bits(bitvector v){
        Setter = 11111111;
        For (int i = 0; i < length / 8 + 1; i++){
                v->vector[i] & setter;
        }
}
```

2)

In order to avoid any memory leaks from my BitVector ADT, the function bv_delete will free the memory allocated by the BitVector ADT. That function first clears the vector of bits, then the actual structure itself. This will be called at the end of the program to avoid any memory leakage.

3)

Not sure this will improve runtime but before the main loop in sieve, turn all even numbers except 2 to 0, and then go through the loop starting from 3 and only going through the odd numbers.

// prints each prime and if it is special by checking if it is equal to fib, luc, or mers function calls

Print function

```
for (int i = 0; i < bv.length; i++){
        if bv_get_bit(i) == 1 {
                print i;
                Call fib, luc, and mers with i
                        Print if function calls == i
        }
}
```

// converts number to a string of a chosen base, used to check for palindrome

Convert function(int n, int base){

```
        String/pointer s;
        do{
                S += n % base + '0'
                n /= base;
        } while (n > 0);
        Return s;
}
```

// checks string if its palindrome

Bool isPalindrome (String s){

```
        Bool f = true;
        For (int i = 0, goes through half the string){
                If s[i] != s[length - i]{
                        F =false;
                }
        }
        Return f;
}
```