Design

Prelab part 1:

1.

The function sets all Fibonacci numbers to 1 in the bit vector v

```
fib(v)
    First two numbers of the fib series are 0 an 1 and the next is a temporary holder
    N = 0
    M = 1
    next  = 1
    bv_set_bit(v, m)
    bv_set_bit(v, n)
    While next < bv_get_len(v)
        Bv_set_bit(v, next)
        Next = n + m
        M = n
        N = next
    return
```

The function sets all lucasnumbers to 1 in the bit vector v

```
lucas(v)
    First two numbers of the lucas series are 0 an 1 and the next is a temporary holder
    N = 1
    M = 2
    next  = 1
    bv_set_bit(v, m)
    bv_set_bit(v, n)
    While next < bv_get_len(v)
        Bv_set_bit(v, next)
        Next = n +  m
        M = n
        N = next
    return
```

functions set all Mersenne numbers to 1 in the bit vector v

```
mersenne(v)
    X = 4
    bv_set_bit(v, 1)
    while (x-1 is smaller than bv_len(v))
        bv_set_bit(v, (x-1))
        X *= 2
        return
```

2.

```
For i in range (lenght of number / 2)
    If number(i) does not equal s[1(i+1)]
        The number is not a palindrome
```

Prelab part 2

1. This is given in the document under bv.c
2. You prevent memory leaks when you free allocated memory because it allows the memory not to overlap when creating more allocated memory or when ending the code the might be able to get accessed outside of the document since you never cleared it.
3. Maybe take out bv_set_bit(v, 2) because you already set all bits so you don't need to set bit at position 2 again.

The design of the program can be split into three parts:

bv.h file (header file, initializing the bitvector functions)

bv .c file (c code for all the functions initialized in the bv header file)

Sieve.h (header file, initializing the sieve function)

Sieve.c ((c code for the function initialized in the sieve header file))

Tower.c file (the actual code to find the different prime numbers, and base conversion, and palindrome checker, this file also has the optarg code)

Bv.h

This design came from the lab manual.

Bv.c

The design of this lab and the majority of the code came from Eugene's lab section on Wednesday 11/4/2020. And is very similar to the

Pseudocode for bv.c

```
    Creates bit vector
```

```
bv_create(bit_len)
Initializes bit vector using malloc
If not v meaning bitvectos isn’t initialized
    Return 0
If bitlen is smaller than 1
    Set bitlen = 1
Set v->lenght to bit_len
Initializes v->vecto pointer using calloc
    #using calloc is done because we are allocating starting at 0 which malloc doesn’t always do
    #also space is done by dividing and adding one to round since we are allocating bytes and it is a bit vector so for each 8 bit we need on byte in space
If not v->vector meaning pointer isn’t initialized
    Return 0
```

```
Frees up the bitvector v, to prevent memory leaks
bv_delete(v)
    free(v->vector) frees up the pointer
    v->vector = NULL setting the vector to NULL
    free(v) frees up the bitvector itself
```

```
Returns the length of the bitvector v
bv_get_len(v)
    Return v->length
```

```
In vector v sets at position i to 1
 bv_set_bit(v, i)
    If not v meaning v is not initialized
        Return
    If i is bigger than the length of the vector meaning i is not within the vector
        Return
    I = i % v->length to set the location of the byte which i is located at
    Sets byte to which byte i is in
    Sets position in what position i is in inside the byte
    Sets the byte in which i is located into the byte and 1 << position bitwise operation
```

```
Clears bitvector v at I
Bv_clr_bit(v, i)
```

```
        if not v meaning v is not initialized
            Return
        If i is bigger than the length of the vector meaning i is not within the vector
            Return

        I = i % v->length to set the location of the byte which i is located at
        Sets byte to which byte i is in
        Sets position in what position i is in inside the byte
        Sets maks to the inverted of 1 << position
        Sets the byte in which i is located in to the bitwise and operation of byte and mask

    Returns the bit located at I in bitvector v
    bv_get_bit(v, i)
        if not v meaning v is not initialized
            Return 0
        If i is bigger than the length of the vector meaning i is not within the vector
            Return 0
        Sets position in what position i is in inside the byte
        Returns using position shifts and And bitwise operations the bit at location i

    Set all bits within bitvector v to 1
    Bv_set_all bits(v)
        if not v meaning v is not initialized
            Return
        For i in length of vector
            bv_set_bit(v, i)
```

Sieve.h

The design was given in the assignment.

Sieve.c

The algorithm was given in the lab manual, so I copied and cited it in the c file.

Sequence.c

Design for Lucas and Fibonacci numbers:

Those two functions are pretty basic taking in an empty vector and going through the vector. They if it is Lucas or Fibonacci number and set it 1 if so. The algorithm is pretty easy looking at the numbers. Both work the same but have two different starting numbers since the next number is always the previous two combined.

Pseudocode

```
The function sets all Fibonacci numbers to 1 in the bit vector v
fib(v)
    First two numbers of the fib series are 0 an 1 and the next is a temporary holder
    N = 0
    M = 1
    next  = 1
    bv_set_bit(v, m)
    bv_set_bit(v, n)
    While next < bv_get_len(v)
        Bv_set_bit(v, next)
        Next = n + m
        M = n
        N = next
    return

The function sets all lucasnumbers to 1 in the bit vector v
lucas(v)
    First two numbers of the lucas series are 0 an 1 and the next is a temporary holder
    N = 1
    M = 2
    next  = 1
    bv_set_bit(v, m)
    bv_set_bit(v, n)
    While next < bv_get_len(v)
        Bv_set_bit(v, next)
        Next = n +  m
        M = n
        N = next
    return
```

Design for Lucas and Fibonacci numbers:

The Mersenne numbers are different but the function works is similar to Lucas and Fibonacci functions, it iterates through a given vector and checks if it is one of the Mersenne numbers. The Mersenne calculation works like this, it (n^2) - 1, but you can also do the way I implemented it and that is (n-1)*2 and than -1.

```
functions set all Mersenne numbers to 1 in the bit vector v
mersenne(v)
```

```
X = 4
bv_set_bit(v, 1)
while (x-1 is smaller than bv_len(v))
    bv_set_bit(v, (x-1))
    X *= 2
return
```

Palindrome and base change design:

The function takes in a number and a base that it needs to change into. The binary conversion works pretty easy using modulo of the base it will get the pop off the last digit of the base, so if the number is 15 and you do modulo 2 to change to base 2 the last digit will be 0, and if you continue doing while also dividing the number with the base rounding to whole numbers, that till number equals 0 and you have changed base. The issue was that when adding I would have a number in reverse. This isn't an issue for the palindrome checker since it should be in the same in reverse. But when printing I had to go through it in reverse instead of just printing the new number.

```
function changes the base from 10 to the given base for the number i and checks if it is a palindrome
pseudocode for the palindrome checker is from the lab manual
chage_base_palin_check(base, i)
    Digit = "0123456789abcdefghijklmnopqrstuvwxyz" all possible digit for bases
    New[10000] to allocate enough space for changed base number
    Temp = i since we don't actually wanna change i
    X = 0
    while(temp ! = 0)
        New[x] = digits[temp % base]
        X += 1
        Temp /= base
    x-= to set x to last position of the number

    Palin = true

     For y in the length of x / 2
        If new[y] does not equal new[x-y]
            The number is not a palindrome
            Palin = false

    If palin == true
        Print "i ="
        Have to print new backwards since the base conversion puts the number backwards
        While x >= 0
            Print new[x]
            x-=1
        Print “\n”
    return

Main function the main code where the functions are called and the getopt code is
main(argc, arg)
    C = 0

    Max is the range from 0 - max where it going to check primenumbers
    Max = 1000

    Bools for getopt loop
    All_primes = fales
    Pal_prime = false

    While loop through the command line arguments
        If n
            Sets number give after n to max
        If s
            Sets all_primes true
        If p
            Sets pal_primes true

    If no args is given
        Returns an error message

    If give numbers after n is less than one
        Returns an error message


    Initializes prime_numbers vector
    This is done before any of the if statements since it will be used in both if statements
    Prime_numbers = bv_create(max)
    Sets all prime numbers to 1
    sieve(prime_numbers)
```

```
If all_primes is true
    Creates these three vectors
    fib_numbers = bv_create(max)
    lucas_numbers = bv_create(max)
    mers_numbers = bv_create(max)

    finds all the Fibonacci numbers inside the vector.
    fib(fib_numbers)
    finds all the Lucas numbers inside the vector.
    lucas(lucas_numbers)
    finds all the Mersenne numbers inside the vector.
    mersenne(mers_numbers)

    For i in prime_numbers
        If i is one meaning it is a prime number
            print (i, ": prime")
            If i is a Mersenne number
                print(", mersenne")
            If i is a Lucas number
                print(", mersenne")
            If i is a Fibonacci number
                print(", mersenne")
            print("\n")
    frees up the allocated vectors to prevent memory leak.
    bv_delete(fib_numbers)
    bv_delete(lucas_numbers)
    bv_delete(mers_numbers)
adds a blank space between the two if both are true
If all_primes and pal_primes are true
    Print(\n)

if pal_primes is true print primes in palindromic primes in bases 2, 9, 10, 10+last name initial(r = 18th number in alphabet)= 28.
If pal_pries is true
    Base = 2
    print("Base  ", base)
    print("---- --")
    For i in length of prime_numbers
        If i is a prime number
```

```
            change_base_palin_check(base, i)
    Base = 9
    print(“Base  ”, base)
    print(“---- --”)
    For i in length of prime_numbers
            If i is a prime number
                change_base_palin_check(base, i)
    Base = 10
    print(“Base ”, base)
    print(“---- --”)
    For i in length of prime_numbers
            If i is a prime number
                    change_base_palin_check(base, i)
    Base = 28
    print(“Base ”, base)
    print(“---- --”)
    For i in length of prime_numbers
            If i is a prime number
                    change_base_palin_check(base, i)
Frees up the bitvector prime_numbers
bv_delete(prime_numbers)
Return 0
```

Infer and Valgrind

Infer gave two errors:

A memory leak at the end of bv_create

This is because infer gives en error when allocated memory isn’t freed. But this is done in the function bv_delete so this is most likely a false positive.

An uninitialized value

This is true but isn’t really an issue since it isn’t really a big structure we are passing through

Valgrind gave no error:

This is why I am not so worried about the infer error since valgrind didn’t report any issues