PRELAB PART 1

```
1.For i in Prime numbers:
        check( Fibonacci)
        check(Lucas)
        Check (mer)i

Fibonacc(n)i:
Set term 0=0
Set term 1=1
If (term is <=1) :
        Return n)
for(int i=2; i<=n;i++):
  answer+=Add previous two terms


Lucas(n):
term1=2
term2=1

Return if term is 1 or 2:

Add previous two terms

meri(n):
Return n = (1<<n)-1
```

2. Check what base it is. And put into the functions

PRELAB PART 2

BitVector ADT:

```
BV_create:
Allocate space in the struct->vector
Allocate length
Return v

BV_delete:
free(v)

BV_getlength:
Return Vector of length
```

```
BV_set:
Place 1 at desire position of i/8
Bitwise OR to place

BV_clear:
Put ~1 at desire position of i/8
Bitwise And to clear

BV_get:
Get value at i/8 position
Use Bitwise AND to determine if 1 or 0

BV_set_all:
For i in length of input:
Set_bit to 1
```

2.To avoid memory leak, I would use a loop to go through each index in array and free(), then i would set them to NULL

3.I guess the only bad thing about the sieve code, is its O(n^2)?
I would probably not set them all to 1's and just leave it with zeros, and only set to 1 if prime?

PSUEDOCODE:

```
Int main{
Create BitVector

getopt()--Take in argument in command line with flags

-s=prime function, -p=palindrome, -n=number length

Case 's':
Call sieve function
For each prime number in vector array <= number length:
        Check Lucas, check Fibo, check Meri:
        Print if the prime number is a part of the above functions

Case 'p'
Call sieve function
For each prime in vector array <=length:
        Call Convert(i) function
        Check if Palindrome:
        If palindrome then Print

Palindrome:
Go through array string
Check if left is equal to right
Then increment down by 1

Fib(n):
Return at desired position

Lucas(n);
Return at desired position

Mers(n):
Return at desired position

Convert(n):
While n>0:
n%base wanted
Save input in array
n/base wanted
return
```