DESIGN

Pre-Lab Part 1:

1) Same answer as question 2

2) def isMersennePrime(input bitvector instance v, input index) { //check if num is mersenne
    Boolean variable mersenne = false
    If input index is a prime number:
      Double log_base_2_index = log_base_10_index / log_base10_2 (change of base rule)
      if (log_base_2_index = floor(log_base_2_index): //i.e: check it's an integer
        Mersenne = true
      return mersenne

```
  def fibonacci() {  //Return next fibonacci number
    Static variables n0 = 0, n1 = 1
    Variable n = n0 + n1;
    n0 = n1
    n1 = n
    return n
}
def lucas() {  //Return next lucas number
  Static variables n0 = 0, n1 = 1, case1 = 1, case2 = 2
  If case2 == 2: //Return 2 on first call
    Case2 = 0 //Modify case so it's never run again
    Return 2
  Variable n = n0 + n1
  n0 = n1
  n1 = n

  Return n
}
```

Pre-Lab Part 2:

Bit Vector Implementation:

```
def *bv_create(input bitvector length) {
  Allocate bitvector pointer bv memory the size of a bitvector object
  If bv doesn't point nowhere {
    Variable bv length = input bitvector length+1
    Input bitvector length = bitvector length / 8 + 1 To determine bytes of memory needed for bits
    Allocate memory of size ((bitvector lenght+1) * size of one byte) for variable vector array
  }
  return bv pointer;
}

def bv_delete(input Bitvector instance v) {
  free memory allocated to bitvector v's array
  set vector array pointer to null
  Free memory allocated to bitvector v object
  Set bitvector pointer v to null
  return
}

def bv_get_len(input Bitvector instance v) {
  Return v->bv length
}

def bv_set_bit(input Bitvector instance, input index) {
//to modify bit at input index.
  Variable index_in_vector_array = input index / 8
  Variable index_within_uint8_t = input index % 8
  Variable bit_shift = absolute value of (7-index-within_uint8t)
   Bv vector[vector index] = (Bv vector[vector index]) BITWISE-OR (0x1 shifted left by
bit_shift)
}

def bv_clr_bit(input Bitvector instance, input index) {
//to modify bit at input index.
  Variable index_in_vector_array = input index / 8
  Variable index_within_uint8_t = input index % 8
  Variable bit_shift = absolute value of (7-index-within_uint8t)
```

```
   Bv vector[vector index] = (Bv vector[vector index]) BITWISE-AND NOT(0x1 shifted left by
bit_shift)
}

def  bv_get_bit(input Bitvector instance, input index) {
  Variable index_in_vector_array = input index / 8
  Variable index_within_uint8_t = input index % 8
  Variable bit_shift = absolute value of (7-index-within_uint8t)
  return (((bv vector[vector_index] BITWISE-AND  (0x1 shifted left by shift_index value)))
shifted right by shift_index);
}

def bv_set_all_bits(input Bitvector instance) {
//to modify bit at input index.
  For loop iterating over each input index till bitvector length {
    Variable index_in_vector_array = input index / 8
    Variable index_within_uint8_t = input index % 8
     Variable bit_shift = absolute value of (7-index-within_uint8t)
     Bv vector[vector index] = (Bv vector[vector index]) BITWISE-OR (0x1 shifted left by
bit_shift)
  }
}
```

2) When user allocated memory is freed because it's no longer needed, the memory can then be repurposed for different reasons either by the computer or the user. However, if the memory allocated is never freed then it can't be repurposed by the computer or user even after the need to access the stored data in that memory is gone. As a result, that part of memory becomes unusable for further memory allocation. The computer, overall, has less memory to be able to use then.
3) The implementation of the algorithm uses two for loops which may increase the complexity of the function. If the same kind of algorithm can be implemented with one for loop, the program may be able to run faster.

Main program flow: User entered flags are stored in boolean variables as true if they are passed in. Based on the values of the boolean variables, either all types of primes are printed or all palindromic primes are printed. Or, if the boolean variables associated with printing primes and palindromes are both true then both are printed. The -s flag is execute by running printPrime function once. The -p flag is executed by running palindrome_print for each base(2,10,14,31). Code for isPalindromePrime(string) based on pseudocode provided in assignment pdf.

Note: The sieve code for determining primes was provided by Professor Darrell Long.

Pseudocode for printing primes (Function run once)

```
def printPrime(input BitVector object v) {
  Variables stored_lucas_number, stored_fibonacci_number
   For-loop iterating over all bit indices in Bitvector:
    if  index is prime(bit associated with index is 1):
      if isMersennePrime(v, index) is true:
        print “, “
        Print “mersenne”

       if  stored_lucas_number = current for-loop index:
          Print “, “
          print “lucas“
       Else:
         While loop till stored_lucas_number is less than current for-loop index:
           stored_lucas_number =lucas()
            If stored_lucas number == i:
            Print “, “
            Print “lucas”

      If  stored_fibonacci_number = current for-loop index:
          Print “, “
          print “fibonacci“
       Else:
         While loop till stored_fibonacci_number is less than current for-loop index:
           stored_lucas_number = fibonacci()
            If stored_lucas number == i:
            Print “, “
```

```
            Print “fibonacci”

    print new line

}
```

Pseudocode for base change:

```
def base_change(input number, input base, input string to modify) {
  quotient = number
  Remainder = 0
  array_iterator = 0;
  While quotient != 0:
     remainder = quotient % base
     quotient = quotient / base
      If remainder >= 10:
        Add to string[array_iterator] as alphabetic character by adding 87
      Else:
        Add to string[array_iterator] as numeric character by adding 48
        Increment array_iterator by 1


  String[array_iterator] = null terminator

}
```

Pseudo code for palindrome_print. (Function run for each base)

```
def palindrome_print(input BitVector object v, input string, input base) {
  Print “Base” + base
  Print “-----       -----”

  For-loop iterating indices of bitvector:
     if index is prime(bit associated equals 1):
        base_change(index, input base, input string)
        If isPalindromePrime(input string) == true:
           print index + “ = ” + string
}
```