Ryan Child
Chris Reinke
Shea Watson

100

Homework 1

1.

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499]

We computed this list of primes using a Python implementation of the Sieve of Erathostenes. The code is in Appendix A.

2.

2.1

45 = 3^2 * 5^1

129 = 3^1 * 43^1

5254 = 2^1 * 37^1 * 71^1

We computed these primes based on the table from problem 1. The algorithm proceeds as follows: Starting with the lowest prime, 2, we check to see if the prime can be evenly divided into the number. If so, we find the two factors, and repeat the process with the number that is not a prime. If it does not divide evenly, we move on to the next largest prime. Applied recursively, this finds all the prime factors of the number. Our python implementation of this algorithm is given in Appendix B.

3.

E(41) = 41^2 - 41 + 41 = 41^2 = 1681.

The primal decomposition of 1681 is 41*41 = 41^2.

$E(42) = 42^2 - 42 + 41 = 1763$

The primal decomposition of 1763 is 41*43 (found using the method described in problem 2)

4.

n	M(n)	Prime decomposition
1	1	1
2	3	3
3	7	7
4	15	3*5
5	31	31
6	63	$3^2 * 7$
7	127	127
8	255	3 * 5 * 17
9	511	7 * 73

5.

a) 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97

b) $(4*x1 + 1)(4*x2 + 1) = 16*x1*x2 + 4*x2 + 4*x1 + 1$

$= 4(4*x1*x2 + x2 + x1) + 1$

Since x1 and x2 are natural numbers, $4*x1*x2 + x2 + x1$ must be a natural number as well, so the product of any two Hilbert numbers is always a Hilbert number.

c) $1617 = 4x + 1$

$1616 = 4x$

$x = 404$

H1 = 21

H2 = 77

H3 = 33

H4 = 49

Appendix A

```
Length = 500
primes = [x for x in range(length)]
for i in range(2,length):
  if i**2 > length:
    break
  elif primes[i]:
    for j in range(i**2,length,i):
      primes[j] = 0
```

Appendix B

```
length = int(sys.argv[1])
primes = [x for x in range(length)]
primepow = []

def factor(x):
  global primes
  global primepow
  i = 0
  while True:
    if x % primes[i] == 0:
      primepow[i] += 1
      factor2 = x // primes[i]
      x = factor2
      if factor2 in primes:
        primepow[primes.index(factor2)] += 1
        return
    else:
```

```
    i += 1

def printprimes():
  global primes
  i = 0
  str = ''
  for number in primepow:
    if number:
      str += "%d^%d * " % (primes[i],primepow[i])
    i += 1
  str = str[:len(str)-2]
  print str

for i in range(2,length):
  if i**2 > length:
    break
  elif primes[i]:
    for j in range(i**2,length,i):
      primes[j] = 0

while 0 in primes:
  primes.remove(0)
primes = primes[1:]
print primes

primepow = [0]*len(primes)
n = int(sys.argv[2])
factor(n)
printprimes()
```

100

EECE-622 | Nick Sorrell - Eba Erco - Eliot Gomez
Quantum Computing | January 16, 2012

$\boxed{1}$ There are 95 prime numbers in the range of 500 - They are:
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499}
See attached primeSieve.py for code

$\boxed{2}$

(2.1) Factors of 45 are 3 and 15
(2.2) Factors of 129 are 3 and 43
(2.3) Factors of 5254 are 2 and 2627

The procedure we used is as follows:
1) Generate a list of primes (Sieve method) up to the number we wish to factor, n
2) For each prime, p, starting at beginning of list, perform $\frac{n}{p}$
3) If, where i is integer, $\frac{n}{p} = i$, then p and i are prime factors
4) Else, go to 2
! If end of list is reached without finding i, then n is prime.
See attached primeFactor.py for code

$\boxed{3}$
$E(n) = n^2 - n + 41$
$E(41) = 41^2 - 41 + 41 = 41^2 \rightarrow \ Primal\ decomp = 41 \times 41$
$E(42) = 42^2 - 42 + 41 = 42^2 - 1 = 1764 - 1 = 1763 \rightarrow \ Primal\ decomp = 41 \times 43$

$\boxed{4}$
$M(n) = 2^n - 1$
$M(1) = 2^1 - 1 = 2 - 1 = 1 \rightarrow \ Primal\ decomp = 1 \times 1$
$M(2) = 2^2 - 1 = 4 - 1 = 3 \rightarrow Prime$
$M(3) = 2^3 - 1 = 8 - 1 = 7 \rightarrow Prime$
$M(4) = 2^4 - 1 = 16 - 1 = 15 \rightarrow \ Primal\ decomp = 3 \times 5$
$M(5) = 2^5 - 1 = 32 - 1 = 31 \rightarrow Prime$
$M(6) = 2^6 - 1 = 64 - 1 = 63 \rightarrow \ Primal\ decomp = 3 \times 21 = 3^2 \times 7$
$M(7) = 2^7 - 1 = 128 - 1 = 127 \rightarrow Prime$
$M(8) = 2^8 - 1 = 256 - 1 = 255 \rightarrow \ Primal\ decomp = 3 \times 85 = 3 \times 5 \times 17$
$M(9) = 2^9 - 1 = 512 - 1 = 511 \rightarrow \ Primal\ decomp = 7 \times 73$

$\boxed{5}$

(a) There are 23 Hilbert numbers in the range of 100 - They are:
{9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97}
See attached hilbertPrime.py for code

(b) $(4n+1) \times (4n+1) = 16n^2 + 8n + 1 = 4(4n^2 + 2n) + 1$
Let $N = 4n^2 + 2n$, and N is a natural number since n is also natural. Thus we have:
$(4n+1) \times (4n+1) = 4N + 1$, and thus, another Hilbert number.

(c) $1617 = 4(404) + 1$, so 404 is our natural number. Also,
$1617 = 21 \times 77 = 49 \times 33$, so $H1 = 21, H2 = 77, H3 = 49, H4 = 33$
$H1 = 21 = 4(5) + 1$ $\qquad H2 = 77 = 4(19) + 1$
$H3 = 49 = 4(12) + 1$ $\qquad H4 = 33 = 4(8) + 1$
See attached hilbertFactor.py for code

```
'''
Created on Dec 11, 2010

@author: nicksorrell
'''
ceilingInt = 500

def primeGenerator():
    #ceilingInt = int(raw_input("Enter the top int: "))
    #setup variables
    primeList = []
    basePrime = 2

    #add first prime to list, 2
    primeList.append(basePrime)

    # create a list of values not containing multiples of 2
    for i in range(basePrime, ceilingInt):
            if ((i % basePrime) != 0):  #if multiple of 2
                primeList.append(i)

    # remove increasing multiples of basePrime
    # where basePrime is the next prime in our list of sieved primes
    currentPosition = 0
    while ((basePrime*basePrime) <= ceilingInt) :
        #increase position in prime list
        currentPosition += 1
        #set sieve prime to new position in list
        basePrime = primeList[currentPosition]

        for i in primeList:
            # if the ith value isn't our current prime, and is a multiple of
basePrime - remove it!
            if ( (i != basePrime) and ((i % basePrime) == 0) ):
                primeList.remove(i)
    return(primeList)

def main():
    findInt = int(raw_input("Enter the int: "))
    primeList = primeGenerator()
    foundPrime = False
    factor1 = 0
    currentPosition = 0

    while (not foundPrime):
         #test prime in ith location
         basePrime = primeList[currentPosition]
         factor1 = findInt/basePrime
         # test if the factor is the integer we're looking for
         if ((factor1*basePrime) == findInt):
             foundPrime = True
```



```
         currentPosition += 1
    print "Factors of", findInt, "are", basePrime, "and", factor1

main()
```

```
'''
Created on Dec 11, 2010

@author: nicksorrell
'''
ceilingInt = 500

def primeGenerator():
    #ceilingInt = int(raw_input("Enter the top int: "))
    #setup variables
    primeList = []
    basePrime = 2

    #add first prime to list, 2
    primeList.append(basePrime)

    # create a list of values not containing multiples of 2
    for i in range(basePrime, ceilingInt):
            if ((i % basePrime) != 0):  #if not a multiple of 2, add to list
                primeList.append(i)

    # remove increasing multiples of basePrime
    # where basePrime is the next prime in our list of sieved primes
    currentPosition = 0
    while ((basePrime*basePrime) <= ceilingInt) :
        #increase position in prime list
        currentPosition += 1
        #set sieve prime to new position in list
        basePrime = primeList[currentPosition]

        for i in primeList:
            # if the ith value isn't our current prime, and is a multiple of
basePrime - remove it!
            if ( (i != basePrime) and ((i % basePrime) == 0) ):
                primeList.remove(i)
    return(primeList)

def main():

    primeList = primeGenerator()
    print "There are", len(primeList), "prime numbers in the range of",
ceilingInt, "-  They are:"
    print primeList

main()
```

```
'''
Created on Dec 11, 2010

@author: nicksorrell
'''
ceilingInt = 100


def primeGenerator():

    # The Hilbert numbers are the natural numbers (apart from 1) which can be
written
    # as 4x(natural number) + 1.
    #ceilingInt = int(raw_input("Enter the top int: "))
    #setup variables
    primeList = []
    natNumber = 2 # since the HW specifies 'other than 1'
    hilbertPrime = 0
    doneSearching = False
    # create a list of values of 4n+1 <= ceilingInt
    while (not doneSearching):
            hilbertPrime = 4 * natNumber + 1
            natNumber += 1
            if (hilbertPrime > ceilingInt):
                doneSearching = True
            else:
                primeList.append(hilbertPrime)

    return(primeList)

def main():

    primeList = primeGenerator()
    print "There are", len(primeList), "Hilbert numbers in the range of",
ceilingInt, "-  They are:"
    print primeList

main()
```

```
'''
Created on Dec 11, 2010

@author: nicksorrell
'''
# int to stop generating numbers at
ceilingInt = 1613

def primeGenerator():
    # The Hilbert numbers are the natural numbers (apart from 1) which can be
written
    # as 4x(natural number) + 1.
    #ceilingInt = int(raw_input("Enter the top int: "))
    #setup variables
    primeList = []
    natNumber = 2
    hilbertPrime = 0
    doneSearching = False

    # create a list of values of 4n+1 <= ceilingInt
    while (not doneSearching):
           hilbertPrime = 4 * natNumber + 1
           natNumber += 1
           if (hilbertPrime > ceilingInt):
               doneSearching = True
           else:
               primeList.append(hilbertPrime)

    return(primeList)

def findNatural(hilbertNum):
    return ((hilbertNum - 1) / 4)

def main():
    findInt = int(raw_input("Enter the int: "))
    primeList = primeGenerator()
    factor1 = 0
    currentPosition = 0
    hilbertFactors = []
    n = findNatural(findInt)
    print "Natural number of Hilbert number", findInt, "is", n

    for i in primeList:
         basePrime = primeList[currentPosition]
         factor1 = findInt/basePrime

         # test if the factor is integer value
         if ((factor1*basePrime) == findInt):
             hilbertFactors.append([factor1, basePrime])
             primeList.remove(factor1)      # remove factor from primeList so
we don't have dupes in list
```


hilbertFactor.py 2012-01-18

```
         currentPosition += 1

    print "Hilbert factors of", findInt, "are"
    for i in hilbertFactors:
        print i
        nat0 = findNatural(i[0])
        nat1 = findNatural(i[1])
        print i[0],"=4(%i)+1" % nat0
        print i[1],"=4(%i)+1" % nat1


main()
```

Intro to Quantum Computing : ECES622 - Winter 2012

100

Homework 1

18 January 2012

Hua Li, Kevin Pasko, Allen Welch

1. The sieve of Eratosthenes was used to generate all the prime numbers from 2 to 500. The states at each step of sieve are shown on the following pages. Code in Python to generate the sieve is shown below. Sieverange(min,max) will generate a 1D array with all primes in the provided range. The sieve at each step is shown on the next three pages.

```
# Homework1.py
# Problem 1: Sieve, lists prime up to user inputted value.
# The list a covers the range for the Sieve beginning with first prime.
def sieve(maxvalue,outputenabled):
    import math
    a=range(2,maxvalue+1)
    # The Sieve will replace all non-primes with 0.
    # Outside while loop iterates through all the values.
    # If statement checks to see if current cell is prime.
    # Inside while loop sets non-prime numbers to 0.
    i=0
    while i in range(0,int(math.ceil(math.sqrt(len(a))))):
        j=1
        if a[i] != 0:
            while i+a[i]*j in range(i,len(a)):
                a[i+a[i]*j]=0
                j=j+1
        i=i+1
         # Output of Sieve data at every prime in 10 columns.
        if outputenabled == 1 and a[i-1] != 0:
            print repr(0).rjust(3),
            for k in range(0,9):
                    if k<8:
                        print repr(a[k]).rjust(3),
                    else:
                        print repr(a[k]).rjust(3)
            for l in range(1,(len(a)+1)/10):
                for k in range(-1,9):
                    if k<8:
                        print repr(a[k+l*10]).rjust(3),
                    else:
                        print repr(a[k+l*10]).rjust(3)
            print '\n'
    # Output of final Sieve data in 10 columns.
    if outputenabled == 2:
        print repr(0).rjust(3),
        for k in range(0,9):
                if k<8:
                    print repr(a[k]).rjust(3),
                else:
                    print repr(a[k]).rjust(3)
        for l in range(1,(len(a)+1)/10):
            for k in range(-1,9):
                if k<8:
                    print repr(a[k+l*10]).rjust(3),
                else:
                    print repr(a[k+l*10]).rjust(3)
        print '\n'
    x=0
    primes=[]
    for x in range(0,len(a)):
          if a[x] != 0:
              primes.append(a[x])
    return primes

#Sieverange() returns all primes in range (min,max) in a 1D array.
def sieverange(minvalue,maxvalue):
    a=sieve(maxvalue,0)
    while a[0] < minvalue:
        a.remove(a[0])
    return a
```

2

-	2	3	-	5	-	7	-	9	-
11	-	13	-	15	-	17	-	19	-
21	-	23	-	25	-	27	-	29	-
31	-	33	-	35	-	37	-	39	-
41	-	43	-	45	-	47	-	49	-
51	-	53	-	55	-	57	-	59	-
61	-	63	-	65	-	67	-	69	-
71	-	73	-	75	-	77	-	79	-
81	-	83	-	85	-	87	-	89	-
91	-	93	-	95	-	97	-	99	-
101	-	103	-	105	-	107	-	109	-
111	-	113	-	115	-	117	-	119	-
121	-	123	-	125	-	127	-	129	-
131	-	133	-	135	-	137	-	139	-
141	-	143	-	145	-	147	-	149	-
151	-	153	-	155	-	157	-	159	-
161	-	163	-	165	-	167	-	169	-
171	-	173	-	175	-	177	-	179	-
181	-	183	-	185	-	187	-	189	-
191	-	193	-	195	-	197	-	199	-
201	-	203	-	205	-	207	-	209	-
211	-	213	-	215	-	217	-	219	-
221	-	223	-	225	-	227	-	229	-
231	-	233	-	235	-	237	-	239	-
241	-	243	-	245	-	247	-	249	-
251	-	253	-	255	-	257	-	259	-
261	-	263	-	265	-	267	-	269	-
271	-	273	-	275	-	277	-	279	-
281	-	283	-	285	-	287	-	289	-
291	-	293	-	295	-	297	-	299	-
301	-	303	-	305	-	307	-	309	-
311	-	313	-	315	-	317	-	319	-
321	-	323	-	325	-	327	-	329	-
331	-	333	-	335	-	337	-	339	-
341	-	343	-	345	-	347	-	349	-
351	-	353	-	355	-	357	-	359	-
361	-	363	-	365	-	367	-	369	-
371	-	373	-	375	-	377	-	379	-
381	-	383	-	385	-	387	-	389	-
391	-	393	-	395	-	397	-	399	-
401	-	403	-	405	-	407	-	409	-
411	-	413	-	415	-	417	-	419	-
421	-	423	-	425	-	427	-	429	-
431	-	433	-	435	-	437	-	439	-
441	-	443	-	445	-	447	-	449	-
451	-	453	-	455	-	457	-	459	-
461	-	463	-	465	-	467	-	469	-
471	-	473	-	475	-	477	-	479	-
481	-	483	-	485	-	487	-	489	-
491	-	493	-	495	-	497	-	499	-

3

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	25	-	-	-	29	-
31	-	-	-	35	-	37	-	-	-
41	-	43	-	-	-	47	-	49	-
-	-	53	-	55	-	-	-	59	-
61	-	-	-	65	-	67	-	-	-
71	-	73	-	-	-	77	-	79	-
-	-	83	-	85	-	-	-	89	-
91	-	-	-	95	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	115	-	-	-	119	-
121	-	-	-	125	-	127	-	-	-
131	-	133	-	-	-	137	-	139	-
-	-	143	-	145	-	-	-	149	-
151	-	-	-	155	-	157	-	-	-
161	-	163	-	-	-	167	-	169	-
-	-	173	-	175	-	-	-	179	-
181	-	-	-	185	-	187	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	203	-	205	-	-	-	209	-
211	-	-	-	215	-	217	-	-	-
221	-	223	-	-	-	227	-	229	-
-	-	233	-	235	-	-	-	239	-
241	-	-	-	245	-	247	-	-	-
251	-	253	-	-	-	257	-	259	-
-	-	263	-	265	-	-	-	269	-
271	-	-	-	275	-	277	-	-	-
281	-	283	-	-	-	287	-	289	-
-	-	293	-	295	-	-	-	299	-
301	-	-	-	305	-	307	-	-	-
311	-	313	-	-	-	317	-	319	-
-	-	323	-	325	-	-	-	329	-
331	-	-	-	335	-	337	-	-	-
341	-	343	-	-	-	347	-	349	-
-	-	353	-	355	-	-	-	359	-
361	-	-	-	365	-	367	-	-	-
371	-	373	-	-	-	377	-	379	-
-	-	383	-	385	-	-	-	389	-
391	-	-	-	395	-	397	-	-	-
401	-	403	-	-	-	407	-	409	-
-	-	413	-	415	-	-	-	419	-
421	-	-	-	425	-	427	-	-	-
431	-	433	-	-	-	437	-	439	-
-	-	443	-	445	-	-	-	449	-
451	-	-	-	455	-	457	-	-	-
461	-	463	-	-	-	467	-	469	-
-	-	473	-	475	-	-	-	479	-
481	-	-	-	485	-	487	-	-	-
491	-	493	-	-	-	497	-	499	-

5

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	49	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	77	-	79	-
-	-	83	-	-	-	-	-	89	-
91	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	119	-
121	-	-	-	-	-	127	-	-	-
131	-	133	-	-	-	137	-	139	-
-	-	143	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
161	-	163	-	-	-	167	-	169	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	187	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	203	-	-	-	-	-	209	-
211	-	-	-	-	-	217	-	-	-
221	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	247	-	-	-
251	-	253	-	-	-	257	-	259	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	287	-	289	-
-	-	293	-	-	-	-	-	299	-
301	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	319	-
-	-	323	-	-	-	-	-	329	-
331	-	-	-	-	-	337	-	-	-
341	-	343	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
361	-	-	-	-	-	367	-	-	-
371	-	373	-	-	-	377	-	379	-
-	-	383	-	-	-	-	-	389	-
391	-	-	-	-	-	397	-	-	-
401	-	403	-	-	-	407	-	409	-
-	-	413	-	-	-	-	-	419	-
421	-	-	-	-	-	427	-	-	-
431	-	433	-	-	-	437	-	439	-
-	-	443	-	-	-	-	-	449	-
451	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	469	-
-	-	473	-	-	-	-	-	479	-
481	-	-	-	-	-	487	-	-	-
491	-	493	-	-	-	497	-	499	-

7

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	-	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	-	-	79	-
-	-	83	-	-	-	-	-	89	-
-	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	-	-
121	-	-	-	-	-	127	-	-	-
131	-	-	-	-	-	137	-	139	-
-	-	143	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
-	-	163	-	-	-	167	-	169	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	187	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	-	-	-	-	-	-	209	-
211	-	-	-	-	-	-	-	-	-
221	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	247	-	-	-
251	-	253	-	-	-	257	-	-	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	-	-	289	-
-	-	293	-	-	-	-	-	299	-
-	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	319	-
-	-	323	-	-	-	-	-	-	-
331	-	-	-	-	-	337	-	-	-
341	-	-	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
361	-	-	-	-	-	367	-	-	-
-	-	373	-	-	-	377	-	379	-
-	-	383	-	-	-	-	-	389	-
391	-	-	-	-	-	397	-	-	-
401	-	403	-	-	-	407	-	409	-
-	-	-	-	-	-	-	-	419	-
421	-	-	-	-	-	-	-	-	-
431	-	433	-	-	-	437	-	439	-
-	-	443	-	-	-	-	-	449	-
451	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	-	-
-	-	473	-	-	-	-	-	479	-
481	-	-	-	-	-	487	-	-	-
491	-	493	-	-	-	-	-	499	-

11

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	-	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	-	-	79	-
-	-	83	-	-	-	-	-	89	-
-	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	-	-
-	-	-	-	-	-	127	-	-	-
131	-	-	-	-	-	137	-	139	-
-	-	-	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
-	-	163	-	-	-	167	-	169	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	-	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	-	-	-	-	-	-	-	-
211	-	-	-	-	-	-	-	-	-
221	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	247	-	-	-
251	-	-	-	-	-	257	-	-	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	-	-	289	-
-	-	293	-	-	-	-	-	299	-
-	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	-	-
-	-	323	-	-	-	-	-	-	-
331	-	-	-	-	-	337	-	-	-
-	-	-	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
361	-	-	-	-	-	367	-	-	-
-	-	373	-	-	-	377	-	379	-
-	-	383	-	-	-	-	-	389	-
391	-	-	-	-	-	397	-	-	-
401	-	403	-	-	-	-	-	409	-
-	-	-	-	-	-	-	-	419	-
421	-	-	-	-	-	-	-	-	-
431	-	433	-	-	-	437	-	439	-
-	-	443	-	-	-	-	-	449	-
-	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	-	-
-	-	-	-	-	-	-	-	479	-
481	-	-	-	-	-	487	-	-	-
491	-	493	-	-	-	-	-	499	-

13

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	-	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	-	-	79	-
-	-	83	-	-	-	-	-	89	-
-	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	-	-
-	-	-	-	-	-	127	-	-	-
131	-	-	-	-	-	137	-	139	-
-	-	-	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
-	-	163	-	-	-	167	-	-	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	-	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	-	-	-	-	-	-	-	-
211	-	-	-	-	-	-	-	-	-
-	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	-	-	-	-
251	-	-	-	-	-	257	-	-	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	-	-	289	-
-	-	293	-	-	-	-	-	-	-
-	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	-	-
-	-	323	-	-	-	-	-	-	-
331	-	-	-	-	-	337	-	-	-
-	-	-	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
361	-	-	-	-	-	367	-	-	-
-	-	373	-	-	-	-	-	379	-
-	-	383	-	-	-	-	-	389	-
391	-	-	-	-	-	397	-	-	-
401	-	-	-	-	-	-	-	409	-
-	-	-	-	-	-	-	-	419	-
421	-	-	-	-	-	-	-	-	-
431	-	433	-	-	-	437	-	439	-
-	-	443	-	-	-	-	-	449	-
-	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	-	-
-	-	-	-	-	-	-	-	479	-
-	-	-	-	-	-	487	-	-	-
491	-	493	-	-	-	-	-	499	-

17

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	-	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	-	-	79	-
-	-	83	-	-	-	-	-	89	-
-	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	-	-
-	-	-	-	-	-	127	-	-	-
131	-	-	-	-	-	137	-	139	-
-	-	-	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
-	-	163	-	-	-	167	-	-	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	-	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	-	-	-	-	-	-	-	-
211	-	-	-	-	-	-	-	-	-
-	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	-	-	-	-
251	-	-	-	-	-	257	-	-	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	-	-	-	-
-	-	293	-	-	-	-	-	-	-
-	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	-	-
-	-	-	-	-	-	-	-	-	-
331	-	-	-	-	-	337	-	-	-
-	-	-	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
361	-	-	-	-	-	367	-	-	-
-	-	373	-	-	-	-	-	379	-
-	-	383	-	-	-	-	-	389	-
-	-	-	-	-	-	397	-	-	-
401	-	-	-	-	-	-	-	409	-
-	-	-	-	-	-	-	-	419	-
421	-	-	-	-	-	-	-	-	-
431	-	433	-	-	-	437	-	439	-
-	-	443	-	-	-	-	-	449	-
-	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	-	-
-	-	-	-	-	-	-	-	479	-
-	-	-	-	-	-	487	-	-	-
491	-	-	-	-	-	-	-	499	-

19

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	-	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	-	-	79	-
-	-	83	-	-	-	-	-	89	-
-	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	-	-
-	-	-	-	-	-	127	-	-	-
131	-	-	-	-	-	137	-	139	-
-	-	-	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
-	-	163	-	-	-	167	-	-	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	-	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	-	-	-	-	-	-	-	-
211	-	-	-	-	-	-	-	-	-
-	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	-	-	-	-
251	-	-	-	-	-	257	-	-	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	-	-	-	-
-	-	293	-	-	-	-	-	-	-
-	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	-	-
-	-	-	-	-	-	-	-	-	-
331	-	-	-	-	-	337	-	-	-
-	-	-	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
-	-	-	-	-	-	367	-	-	-
-	-	373	-	-	-	-	-	379	-
-	-	383	-	-	-	-	-	389	-
-	-	-	-	-	-	397	-	-	-
401	-	-	-	-	-	-	-	409	-
-	-	-	-	-	-	-	-	419	-
421	-	-	-	-	-	-	-	-	-
431	-	433	-	-	-	-	-	439	-
-	-	443	-	-	-	-	-	449	-
-	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	-	-
-	-	-	-	-	-	-	-	479	-
-	-	-	-	-	-	487	-	-	-
491	-	-	-	-	-	-	-	499	-

23

-	2	3	-	5	-	7	-	-	-
11	-	13	-	-	-	17	-	19	-
-	-	23	-	-	-	-	-	29	-
31	-	-	-	-	-	37	-	-	-
41	-	43	-	-	-	47	-	-	-
-	-	53	-	-	-	-	-	59	-
61	-	-	-	-	-	67	-	-	-
71	-	73	-	-	-	-	-	79	-
-	-	83	-	-	-	-	-	89	-
-	-	-	-	-	-	97	-	-	-
101	-	103	-	-	-	107	-	109	-
-	-	113	-	-	-	-	-	-	-
-	-	-	-	-	-	127	-	-	-
131	-	-	-	-	-	137	-	139	-
-	-	-	-	-	-	-	-	149	-
151	-	-	-	-	-	157	-	-	-
-	-	163	-	-	-	167	-	-	-
-	-	173	-	-	-	-	-	179	-
181	-	-	-	-	-	-	-	-	-
191	-	193	-	-	-	197	-	199	-
-	-	-	-	-	-	-	-	-	-
211	-	-	-	-	-	-	-	-	-
-	-	223	-	-	-	227	-	229	-
-	-	233	-	-	-	-	-	239	-
241	-	-	-	-	-	-	-	-	-
251	-	-	-	-	-	257	-	-	-
-	-	263	-	-	-	-	-	269	-
271	-	-	-	-	-	277	-	-	-
281	-	283	-	-	-	-	-	-	-
-	-	293	-	-	-	-	-	-	-
-	-	-	-	-	-	307	-	-	-
311	-	313	-	-	-	317	-	-	-
-	-	-	-	-	-	-	-	-	-
331	-	-	-	-	-	337	-	-	-
-	-	-	-	-	-	347	-	349	-
-	-	353	-	-	-	-	-	359	-
-	-	-	-	-	-	367	-	-	-
-	-	373	-	-	-	-	-	379	-
-	-	383	-	-	-	-	-	389	-
-	-	-	-	-	-	397	-	-	-
401	-	-	-	-	-	-	-	409	-
-	-	-	-	-	-	-	-	419	-
421	-	-	-	-	-	-	-	-	-
431	-	433	-	-	-	-	-	439	-
-	-	443	-	-	-	-	-	449	-
-	-	-	-	-	-	457	-	-	-
461	-	463	-	-	-	467	-	-	-
-	-	-	-	-	-	-	-	479	-
-	-	-	-	-	-	487	-	-	-
491	-	-	-	-	-	-	-	499	-

2. To decompose a number in to its prime factors, the first step is to attempt to divide it by the lowest prime number, 2. If this does not produce an integer value, continue to the next higher prime number, 3. Repeat this process until the number can be represented with prime factors.

```
#Homework 1 Problem 2
#Prime factorization

def pfact(integerinput,printenabled):

    a=[integerinput]
    prime=sieve(integerinput,0)
    print a

    k=0

    if a[0] > 3:
        while prime.count(a[len(a)-1]) == 0:
            if printenabled == 1:
                print a[len(a)-1],"%",prime[k],"=",a[len(a)-1] % prime[k]
            if a[len(a)-1] % prime[k] == 0:
                a.append(a[len(a)-1]/prime[k])
                a[len(a)-2]=prime[k]
                k=0
                if printenabled == 1:
                    print a
            else:
                k=k+1

    return a
```

2-1. Beginning with 45, first divide by the lowest prime, 2. $45/2 = 22.5$. The result is not an integer. Move to the next highest prime number that is lower than the square root of 45, 3. $45/3 = 15$. This is an integer. 45 is a factor of 3 and 15, and 3 is prime. Repeat the same process on 15. $15/2 = 7.5$. $15/3 = 5$. 45 is a factor of 3 and 3 and 5. All three of these numbers are prime numbers, and the factorization is complete.

The program performs this operation iteratively using a list of prime numbers generated by the Sieve of Eratosthenes program from question 1, but it checks for a remainder rather than perform operations on floating point values. Output is shown below and this format is used for all further prime factorization operations in the homework.

```
[45]
45 % 2 = 1
45 % 3 = 0
[3, 15]
15 % 2 = 1
15 % 3 = 0
[3, 3, 5]
```

2-2. $129/2 = 64.5$. 129/3=43. 3 and 43 are prime numbers. The prime factors are 3 and 43.

```
[129]
129 % 2 = 1
129 % 3 = 0
[3, 43]
```

2-3. 5254/2 = 2627, therefore one of the prime factors is 2. 2627 is not prime, so we repeat the process for 2627. 2627/37=71. 37 and 71 are both prime. Prime factors of 5254 are 2, 37 and 71.

```
[5254]
5254 % 2 = 0
[2, 2627]
2627 % 2 = 1
2627 % 3 = 2
2627 % 5 = 2
2627 % 7 = 2
2627 % 11 = 9
2627 % 13 = 1
2627 % 17 = 9
2627 % 19 = 5
2627 % 23 = 5
2627 % 29 = 17
2627 % 31 = 23
2627 % 37 = 0
[2, 37, 71]
```

3. $E(n) = n^2 - n + 41$

3-1. $E(41) = (41)^2 - 41 + 41 = (41)^2 = 1681$. 41 is prime, therefor the prime factors are 41, 41.

```
[1681]
1681 % 2 = 1
1681 % 3 = 1
1681 % 5 = 1
1681 % 7 = 1
1681 % 11 = 9
1681 % 13 = 4
1681 % 17 = 15
1681 % 19 = 9
1681 % 23 = 2
1681 % 29 = 28
1681 % 31 = 7
1681 % 37 = 16
1681 % 41 = 0
[41, 41]
```

3-2. $E(42) = (42)^2 - 42 + 41 = 1764 - 42 + 41 = 1763$. 41 and 43 are both prime. The prime factors are 41 and 43.

```
[1763]
1763 % 2 = 1
1763 % 3 = 2
1763 % 5 = 3
1763 % 7 = 6
1763 % 11 = 3
1763 % 13 = 8
1763 % 17 = 12
1763 % 19 = 15
1763 % 23 = 15
1763 % 29 = 23
1763 % 31 = 27
1763 % 37 = 24
1763 % 41 = 0
[41, 43]
```

4. Mersenne numbers follow the equation: $M(n) = 2^n - 1$

The first 9 Mersenne numbers (from n=9 to n=1) and their prime decompositions are listed in the table below. Following that is the output of the factorization code to find the prime factors.

n	$2^n - 1$	Prime Factors
1	1	1
2	3	3
3	7	7
4	15	3,5
5	31	31
6	63	3,3,7
7	127	127
8	255	3,5,17
9	511	7,73

```
M(1)=1 is prime.

M(2)=3 is prime.

M(3)=7 is prime.

M(4)=15 is not prime.  The prime factors are 3 and 5.
[15]
15 % 2 = 1
15 % 3 = 0
[3, 5]

M(5)=31 prime.

M(6)=63 is not prime.  The prime factors are 3, 3 and 7.
[63]
63 % 2 = 1
63 % 3 = 0
[3, 21]
21 % 2 = 1
21 % 3 = 0
[3, 3, 7]

M(7)=127 is prime.

M(8)=255 is not prime.  The prime factors are 3, 5 and 17.
[255]
255 % 2 = 1
255 % 3 = 0
[3, 85]
85 % 2 = 1
85 % 3 = 1
85 % 5 = 0
[3, 5, 17]
```

```
M(9)=511 is not prime.  The prime factors are 7 and 73.
[511]
511 % 2 = 1
511 % 3 = 1
511 % 5 = 1
511 % 7 = 0
[7, 73]
```

5.(a).

Hilbert Number=4×(Natural Number)+1

Natural Number	Hilbert Number	Natural Number	Hilbert Number
1	5 *	13	53 *
2	9 *	14	57 *
3	13 *	15	61 *
4	17 *	16	65
5	21 *	17	69 *
6	25	18	73 *
7	29 *	19	77 *
8	33 *	20	81
9	37 *	21	85
10	41 *	22	89 *
11	45	23	93 *
12	49 *	24	97 *

Numbers marked with an asterisk (*) are Hilbert primes.

(b).

Assume that there are two natural numbers x_1 and x_2. Corresponding Hilbert numbers are $(4x_1+1)$ and $(4x_2+1)$ respectively.

Multiply of these two Hilbert numbers is

$(4x_1+1)\ (4x_2+1)=16x_1x_2+4x_1+4x_2+1=4(4x_1x_2+x_1+x_2)+1$.

$\because x_1,\ x_2$ are both natural numbers.

$\therefore x_1x_2$ is a natural number.

$\therefore (4x_1x_2+x_1+x_2)$ is a natural number.

$\therefore [4(4x_1x_2+x_1+x_2)+1]$ satifies the form of [4×(Natural Number)+1].

$\therefore$ The number $[4(4x_1x_2+x_1+x_2)+1]$ is a Hilbert number.

(c).

$1617=4\times404+1$. 404 is a natural number, so 1617 is a Hilbert number.

$1617^{0.5}=40.21$

$\because$ 40.21 is not a Hilbert number.

$\therefore$ For any two Hilbert numbers whose multiply is 1617, one MUST be smaller than 40.21, and the other MUST be larger than 40.21.

Divide 1617 by Hilbert numbers smaller than 40.21 (listed in the above table) one by one, and will get the following result.

$1617=21\times77=33\times49$

21, 77, 33 and 49 are all Hilbert primes.

$\therefore H_1$, H_2, H_3 and H_4 are 21, 77, 33 and 49 respectively.

Intro to Quantum Computing: ECES 622 - Winter 2012

Homework 1: Team effort , Max: 100 pts

Due date: January 18, 2012, in class.

100

Group Names:

Brandon Bright

Mark Holdsworth

Adam Yager

```
function[prime]=numprime(limit,path)
%Prime number counter developed by Mark Holdsworth
% For Quantum Computing Winter 2012
%How many prime numbers are there less than *limit*,
% and what are they? Output answer to text file,
%return total number.


fid=fopen(path,'wt');
prime=0;

for i=2:1:limit;

    if isprime(i) == 1

 fprintf(fid, strcat(num2str(i),'\n'));

 prime=prime+1;

    else

    end % end if

end  % end For loop


fprintf(fid,strcat('\n','Total Number Prime: ',num2str(prime)));

fclose(fid);

%% ALGORITHM FOR IDENTIFYING PRIME NUMBERS
p = 1:2:n; % Starting from 1 catalog every odd number from 1 to N
%in an array
q = length(p); % q= number of elements in array
p(1) = 2; % Reset first array element to 2 (first prime number)

%The for loop starts at k=3 identifies the each subsequent prime number
%for example k=3 yields p((k+1)/2)=p(2)= 3, the second prime number
%Next every kth value is set to zero, or rather "siezed".
for k = 3:2:sqrt(n)
  if p((k+1)/2) % example k=3, p(2)=3
     p(((k*k+1)/2):k:q) = 0;% example k=3, p(5)=9 not prime set to 0
     % after setting 9 to 0, skip two values, and zet to zero.
     %the limit is sqrt(n) for each k value, you skip k places before
     %setting the next value to 0, which is a function of n^2
  end
end
p = p(p>0) ;%only non zero values remain, which are prime numbers only.
```

Problem # 1

Prime

2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
103
107
109
113
127
131
137
139
149
151
157
163
167
173
179
181
191
193
197
199
211
223
227
229
233
239
241
251
257
263
269
271
277
281
283
293
307

Prime

311
313
317
331
337
347
349
353
359
367
373
379
383
389
397
401
409
419
421
431
433
439
443
449
457
461
463
467
479
487
491
499

Total Number Prime:95

Problem 2

- **(1) 45 = 5 * 3 * 3**

```
45 / 5
9
9 / 3
3
```

- **(2) 129 = 43 * 3**

```
129 / 3
43
```

- **(3) 5254 = 2 * 37 * 71**

```
5254 / 2
2627
2627 / 37
71
```

- **Explanation of Procedure**

The first thing to look for when doing a prime decomposition is looking at the 'ones' digit and seeing if it ends in an even number, a five, or a zero. If it ends in an even number, then the number is divisible by 'two'. If it ends in a five, then the number is divisible by 'five'. If it ends in a zero, then the number is divisible by 'two' and 'five'.

(1) - When solving the first problem, we notice it ends in a five. After dividing by five we are left with nine. Nine is a known square of 'three'. Thus we have 'five', 'three', and 'three'.

(2) - When solving the second problem, we notice the first two numbers '12' and the last number '9' is divisible by three. After dividing the number by 'three' we are left with forty-three. Forty-three is a larger number. One way of handling this is to divide forty-three by every number below half of forty-three. A more efficient way is to divide by every number below half of forty-three that is prime. Forty-three was found to be a prime number thus we have 'forty-three' and 'three'.

(3) - When solving the third problem, we notice the last digit is even thus it is divisible by 'two'. This leaves us with 2627. Since this is a large number, we use the method above to find the next prime decomposition which is 'thirty-seven'. This leaves us with seventy-one which after dividing by an appropriate amount of numbers, we realize it is prime. Thus we have 'two', 'thirty-seven', and 'seventy-one'.

Problem 3

```
Euler[n_] := n^2 - n + 41
```

```
Euler[41]
Euler[42]
```

1681

1763

```
41 * 41
41 * 43
```

1681

1763

■ Explanation

After solving for n = 41 and n = 42 of the Euler formula, we do a prime decomposition like that in Problem 2. The square root of E[41] or 1681 is 41 thus we have two prime decompositions (with one distinct prime number) of E[41]. E[42] was prime decomposed using the method stated in problem 2 to give the values 41 and 43. This is proof that the Euler generic formula contains errors or non prime numbers.

Problem 4

```
Mersenne[n_] := 2^n - 1
Table[Mersenne[n], {n, 1, 9}]
```

{1, 3, 7, 15, 31, 63, 127, 255, 511}

■ Finding Prime Decomposition

At n = 4, 15 has a prime decomposition of 5 * 3.
At n = 6, 63 has a prime decomposition of 3 * 3 * 7.
At n = 8, 255 has a prime decomposition of 3 * 5 * 17.
At n = 9, 511 has a prime decomposition of 7 * 73.

Problem 5

```
Hilbert[k_] := 4 k + 1
```

■ (a) Determine Hilbert Primes ≤ 100

```
Table[Hilbert[k], {k, 0, 24}]
```

{1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41,
 45, 49, 53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97}

A Hilbert prime is a Hilbert number not divisible by a smaller Hilbert number. Looking at the above table, everything divisible by 5 is removed; then everything divisible by 9 is removed; then everything divisible by 13 is removed; and so on.

{5, 9, 13, 17, 21, 29, 33, 37, 41, 49, 53, 57, 61, 69, 73, 77, 89, 93, 97}

■ (b) Two Hilbert Numbers multiplied together equal a Hilbert Number

See Attachment for proof.

(c) 1617 = H1 * H2 = H3 * H4

```
Solve[1617 == Hilbert[k]]
Solve[1617 == 4 k + 1]
```

```
{{k → 404}}
```

```
{{k → 404}}
```

Using Problem 5 b' s Proof :

```
H1H2[x_, y_] := 4 x y + x + y
```

```
H1H2[19, 5]
```

```
404
```

```
Hilbert[19]
Hilbert[5]
Hilbert[19] * Hilbert[5]
```

```
77
```

```
21
```

```
1617
```

```
H3H4[8, 12]
```

```
404
```

```
Hilbert[8]
Hilbert[12]
Hilbert[8] * Hilbert[12]
```

```
33
```

```
49
```

```
1617
```

Problem #5b.

→ Example: $H(1) = [4 \cdot 1] + 1 = 5$

$H(2) = [4 \cdot 2] + 1 = 9$

↳ $H(1) \cdot H(2) = 5 \cdot 9 = 45$

↳ $45 = H(11) = [4 \cdot 11] + 1$

↳ $H(1) \cdot H(2) = H(11)$

→ Proof: $H(x) \cdot H(y) = H(z)$

1.) → $[(4x)+1] \cdot [(4y)+1] = 16xy + 4x + 4y + 1$

2.) → $16xy + 4x + 4y + = [4 \cdot (4xy + x + y)] + 1$

3.) → $[4 \cdot (4xy + x + y)] = H(z)$

4.) → $z = 4xy + x + y$

→ Example 2

↳ $H(3) \cdot H(5) = 13 \cdot 21 = 273$ ✓

↳ $H(z) = 273$

↳ $z = 4xy + x + y = 4 \cdot 3 \cdot 5 + 3 + 5 = 68$

↳ $H(68) = 4 \cdot 68 + 1 = 272 + 1 = 273$ ✓

Quantum Computing

100

Homework One

Tom Dickman

John Gideon

Ben Zerhusen

```
function [ out ] = decompose( in )
%DECOMPOSE returns array of primes to decompose the integer input
% This recursive function loops from 2 to the square root of the input
% number checking the modulus of all values. If at any time the modulus
% results in a zero, the interated value must be a factor. Additionally,
% because the program began iterating with the smallest values, the
% interator must be prime. The program then recursively finds the other
% factors of the input divided by the factor and concatenates the result.
% If no factor is found, the value is just simply returned as the base
% case.

    for i = 2:floor(sqrt(in))
        if mod(in,i) == 0 %The value "i" is a factor
            out = [i; decompose(in/i)];
            return
        end
    end
    out = [in];

end
```

```
function [ ] = displayFactors( num, factors )
%DISPLAYFACTORS Prints the factors of a number
% The format for the printed factors is "num = fac1 * fac2 * ... * facn"
% In addition, the flag (PRIME) is added to the end if the value is prime.

    fprintf('Prime Decomposition: %i =', num);
    for i = 1:size(factors)-1
        fprintf(' %i *', factors(i));
    end
    fprintf(' %i', factors(end));
    if size(factors) == 1
        fprintf(' (PRIME)');
    end
    fprintf('\n\n');
end
```

```
function [ out ] = isPrime( in )
%ISPRIME Returns true if prime, false otherwise
% This function decomposes the value and then determines if the returned
% size is one. Because a prime number cannot be decomposed any further, the
% size will be one of prime.

    decom = decompose(in);
    if size(decom,1) == 1
        out = true;
    else
        out = false;
    end

end
```

PROBLEM ONE

Initialize array 500 long. Initially set all to ones, and change to zero if not prime. Proceed through the array starting with 2 and changing all its multiples to zero. Keep iterating though all values until the array has been traversed. Print the results as a 20x25 table.

```
array = ones(500,1);

% 1 isn't in our list
array(1) = 0;

for n=2:size(array)
    if array(n) == 1
        % Remove (set to zero) all multiples up to 500
        for i=2*n:n:size(array)
            array(i) = 0;
        end
    end
end

% Print proper results
for n=0:19
    for i = 0:24
        index = (n*25)+i+1;
        if array(index) == 1
            fprintf('%3i', index)
        else
            fprintf('    ')
        end
    end
    fprintf('\n')
end
```

95 primes

```
       2   3       5       7              11      13              17      19              23
              29      31                      37              41      43              47
          53                      59      61                      67              71      73
              79              83                      89                              97
 101 103             107 109             113
     127             131                     137 139                                     149
 151                     157                     163             167                     173
             179 181                                     191 193             197 199
                                         211                                             223
     227 229             233                     239 241
 251                     257                     263                     269 271
     277             281 283                                     293
                         307             311 313             317
                     331                     337                                     347 349
         353                     359                             367                     373
             379             383                     389                             397
 401                             409                                     419 421
                     431 433                     439             443                     449
                         457             461 463             467
             479                             487             491                             499
```

Published with MATLAB® 7.10

PROBLEM TWO

Use the two supporting functions "displayFactors" and "decompose" to calculate and print the prime decompositions of numbers. Further descriptions are given for this algorithm and the printing function in their headers.

```
displayFactors(45,decompose(45))
displayFactors(129,decompose(129))
displayFactors(5254,decompose(5254))
```

```
Prime Decomposition: 45 = 3 * 3 * 5

Prime Decomposition: 129 = 3 * 43

Prime Decomposition: 5254 = 2 * 37 * 71
```

Published with MATLAB® 7.10

PROBLEM THREE

By using the displayFactors and decompose functions created for problem 2 we are able to show the primal decomposition of each of these two numbers. As seen in the results, this prime decomposition consists of two prime numbers, as predicted in the question.

```
for n=41:42
    E=n^2-n+41;
    displayFactors(E,decompose(E))
end
```

*Prime Decomposition: 1681 = 41 * 41*

*Prime Decomposition: 1763 = 41 * 43*

Published with MATLAB® 7.10

PROBLEM FOUR

By using the displayFactors and decompose functions created for problem 2 we are able to show the primal decomposition of each of these numbers. As seen in the results, all of these are not prime numbers.

```
for n=1:9
    M=2^n-1;
    displayFactors(M,decompose(M))
end
```

```
Prime Decomposition: 1 = 1 (PRIME)

Prime Decomposition: 3 = 3 (PRIME)

Prime Decomposition: 7 = 7 (PRIME)

Prime Decomposition: 15 = 3 * 5

Prime Decomposition: 31 = 31 (PRIME)

Prime Decomposition: 63 = 3 * 3 * 7

Prime Decomposition: 127 = 127 (PRIME)

Prime Decomposition: 255 = 3 * 5 * 17

Prime Decomposition: 511 = 7 * 73
```

Published with MATLAB® 7.10

Table of Contents

Problem 5(a)

Determine all the Hilbert numbers also referred to as Hilbert primes less or equal to 100

```
i = 0;
H = 0;
k = 0;
while (H <= 100)
    H = i*4 + 1;
    if (H <= 100)
        if k <= 4
            fprintf('%6i',H);
            k = k + 1;
        else
            fprintf('%6i\n',H);
            k = 0;
        end
    end
    i = i + 1;
end
fprintf('\n');
```

```
     1     5     9    13    17    21
    25    29    33    37    41    45
    49    53    57    61    65    69
    73    77    81    85    89    93
    97
```

Problem 5(b)

Show that if you multiply any 2 Hilbert numbers you will always get another Hilbert number

```
% A Hilbert number is defined as follows:
% H = 4*n + 1
% Thus when you multiply 2 Hilbert numbers you obtain a number that is of
% the form:
% H*H = (4*n + 1) * (4*n + 1)
% Which can be simplified as:
% H*H = 16*n^2 + 8*n + 1
% H*H = 4*(4*n^2 + 2*n) + 1
% Which is also in the form of a Hilbert number where the seed happens to
% be set as (4*n^2 + 2*n). This in turn proves that multiplying 2 Hilbert
% numbers will always yield another Hilbert number.
```

Problem 5(c)

1617 is a Hilbert number (show it). It can be completely factored as a product of Hilbert primes in two different ways, i.e, 1617 = H1XH2 = H3XH4. Find the values of H1,H2,H3,H4

```
multipleFour = 1617 - 1;
remainder = mod(multipleFour,4);
if (remainder == 0)
    seed = multipleFour / 4;
end
str = sprintf(['The value 1617 is a Hilbert number obtained from using the'...
    ' seed value %d.\n'], seed);
disp(str);

for j = 0:(404/2)
    remainder1 = mod(1617,(j*4 + 1));
    if (remainder1 == 0)
        quotient = 1617/(j*4 + 1);
        remainder2 = mod((quotient - 1),4);
        if (remainder2 == 0)
            str = sprintf(['Hilbert primes multiplied to obtain 1617'...
                ' are %d and %d.\n'], (j*4 + 1) ,quotient);
            disp(str);
        end
    end
end

% As the outputs of this program show, if we exclude the values of 1 and
% 1617, the values we obtain for the four Hilbert primes are H1 = 21,
% H2 = 77, H3 = 33, and H4 = 49.
```

```
The value 1617 is a Hilbert number obtained from using the seed value 404.

Hilbert primes multiplied to obtain 1617 are 1 and 1617.

Hilbert primes multiplied to obtain 1617 are 21 and 77.

Hilbert primes multiplied to obtain 1617 are 33 and 49.

Hilbert primes multiplied to obtain 1617 are 49 and 33.

Hilbert primes multiplied to obtain 1617 are 77 and 21.
```

Published with MATLAB® 7.10

Cameron Hocking
Brian Knox
Josh Hay

100

1)
FindPrimes(500)

ans =

1) 2
2) 3
3) 5
4) 7
5) 11
6) 13
7) 17
8) 19
9) 23
10) 29
11) 31
12) 37
13) 41
14) 43
15) 47
16) 53
17) 59
18) 61
19) 67
20) 71
21) 73
22) 79
23) 83
24) 89
25) 97
26) 101
27) 103
28) 107
29) 109
30) 113
31) 127
32) 131
33) 137
34) 139
35) 149
36) 151
37) 157
38) 163
39) 167
40) 173
41) 179
42) 181
43) 191
44) 193
45) 197
46) 199
47) 211
48) 223
49) 227
50) 229
51) 233
52) 239
53) 241
54) 251
55) 257
56) 263
57) 269
58) 271
59) 277
60) 281
61) 283
62) 293
63) 307
64) 311
65) 313
66) 317
67) 331
68) 337
69) 347
70) 349
71) 353
72) 359
73) 367
74) 373
75) 379
76) 383
77) 389
78) 397
79) 401
80) 409
81) 419
82) 421
83) 431
84) 433
85) 439
86) 443
87) 449
88) 457
89) 461
90) 463
91) 467
92) 479
93) 487
94) 491
95) 499

Matlab Code for Problem 1

```
function [retPrimes] = FindPrimes(upperBound)
% Find all prime numbers up the the value of "upperBound" using
% "The Sieve of Eratosthenes

lastSieve = floor(sqrt(upperBound));

primeArray = 1:1:upperBound; % Generate array of number that will be put
                             % through the sieve
sieveArray = primes(lastSieve); % Generate the array of prime numbers that
                                % will be used for the sieve
for count=1:length(sieveArray)

    % Start the sieve. For each prime number n in the array sieveArray, the
    % value at the index mn (skipping n), where m is an integer, will be
removed from
    % the array primeArray.
    removeValues =
sieveArray(count)+sieveArray(count):sieveArray(count):upperBound;

    primeArray(removeValues) = 0;
end

% remove the value of 1 from primeArray
primeArray(1) = 0;

% parse primeArray and only pull out values that are primes (not zero)
for count=1:length(primeArray)
    if(primeArray(count) == 0)
        continue
    else
        if ~exist('finalArray','var')
            finalArray = primeArray(count);
        else
            finalArray = [finalArray;primeArray(count)];
        end
    end
end

retPrimes = finalArray;
```

PrimeFactors(45)

ans =

3
3
5

>> PrimeFactors(129)

ans =

3
43

>> PrimeFactors(5254)

ans =

2
37
71

Matlab Code for Problem 2

```
function [retFactors] = PrimeFactors(inValue)
% Finds the prime factors of a given number

% If input is already a prime number, it is impossible to find any prime
% factors for it. Check for this condition
testPrimes = primes(inValue);
test = (inValue == testPrimes);
test = max(test);

% The highest possible number that can be multiplied by an integer and
% equal the given value is inValue/2. This will be used to find the upper
% bound of the array of prime numbers to be used
highestPrime = floor(inValue/2);

% Generate an array of prime numbers up to inValue/2
primeArray = primes(highestPrime);

% Variable to keep track of the current comparative value. Starts as
% inValue. Will decrease by inValue/n, n = some prime number, every
% iteration of the loop
currentFactor = inValue;
```

```matlab
% To find the prime factors, the value of currentFactor mod n, n = some
% prime number, will be computed continuously until the result is 0. When
% the result is 0, the divisor is a prime factor of the number. The loop
% will start again with currentValue = currentValue/prime divisor until
% currentValue is 1. When this occurs, all the prime factors are found.
while(1)
    if(currentFactor == 1)
        break;
    end

    if(test == 1)
        break;
    end

    % Starting with the smallest prime and going up to the highest
    % prime <= inValue/2, determine if the prime is a factor of the number
    tempIndex = 1;
    for i=1:length(primeArray)
        modValue = mod(currentFactor,primeArray(i));
        % The tested prime is a factor. Stop checking and add the value to
        % the array factorArray
        if(modValue == 0)
            tempIndex = i;
            break;
        end
    end

    if ~exist('factorArray','var')
        factorArray = primeArray(tempIndex);
    else
        factorArray = [factorArray;primeArray(tempIndex)];
    end

    % Update currentFactor and start the loop over until currentFactor == 1
    currentFactor = currentFactor/primeArray(tempIndex);
end

if(test == 1)
    retFactors = 'Input argument was a prime number';
elseif(inValue == 1)
    retFactors = 'Input argument was a prime number';
else
    retFactors = factorArray;
end
end
```

3) [ret1,ret2] = H1P3;
    1681

    1763

>> ret1

ret1 =

  41
  41

>> ret2

ret2 =

  41
  43

Matlab Code for Problem 3

```
function [ret1,ret2] = H1P3();

n = [41,42];

eulerNumbers = n.^2-n+41;

disp(eulerNumbers(1));
disp(eulerNumbers(2));

ret1 = PrimeFactors(eulerNumbers(1));
ret2 = PrimeFactors(eulerNumbers(2));

end
```

4)
Mersenne number #1, 1, decomposition: Already a prime number
Mersenne number #2, 3, decomposition: Already a prime number
Mersenne number #3, 7, decomposition: Already a prime number
Mersenne number #4, 15, decomposition: 3 5
Mersenne number #5, 31, decomposition: Already a prime number
Mersenne number #6, 63, decomposition: 3 3 7
Mersenne number #7, 127, decomposition: Already a prime number
Mersenne number #8, 255, decomposition: 3 5 17
Mersenne number #9, 511, decomposition: 7 73

Matlab Code for Problem 4

```
% this script finds the prime decomposition of the first 9 Mersenne numbers

clc
clear

mersenne_decomp = struct('number', {}, 'decomposition', {});

for ii = 1:9
    mersenne_nums(ii) = (2^ii)-1;

    decomp = factor(mersenne_nums(ii));

    if length(decomp) == 1  % has no decomposition and is already a prime #
        decomp = 'Already a prime number';
    end

    mersenne_decomp(ii).number = mersenne_nums(ii);
    for i2 = 1:length(decomp)
        mersenne_decomp(ii).decomposition(i2) = decomp(i2);
    end

    %print out first 9 mersenne nums and respective decompositions
    str=fprintf('Mersenne number #%i, %i, decomposition: ', ii,
mersenne_nums(ii));
    disp(mersenne_decomp(ii).decomposition);

end
```

5) hilbert numbers less than 100:

1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49
53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97 ✓

a) Hilbert Primes - a hilbert number that is not divisible by a smaller hilbert number other than 1 (hilbert primes are not necessarily prime numbers)

5, 9, 13, 17, 21, , 29, 33, 37, 41, , 49
53, 57, 61, , 69, 73, 77, , 89, 93, 97 ✓

b) $(4n+1)(4m+1)$ where $n \neq m$

$16mn + 4m + 4n + 1 = 4x + 1$

$16mn + 4m + 4n = 4x$ ✓

$4mn + m + n = x$ ✓ divisible by 4

ex: $(4n+1)(4m+1) = 4(4mn + m + n) + 1$

$n = 3$
$m = 6$ ✓

$(13)(25) = 4(4(3)(6) + 6 + 3) + 1$
$325 = 325$

c) $1617 = 4m + 1 \Rightarrow m = 404$ ✓

$1617 = 33 \times 49$ $H_1 = 33$ $H_2 = 49$
$1617 = 21 \times 77$ $H_3 = 21$ $H_4 = 77$