Homework1

September 22, 2021

[1]:

```python
########## Problem 1

##### part (a)
def divides(a,b):
    #We use that b%a == 0 if and only if a divides b
    return b%a==0

##### part (b)
def getDivisors(a):

    #first give an empty list of divisors
    listOfDivisors = []

    #Check each integer and if it is a divisor add it to the list
    for i in range(1,a+1):
        if divides(i,a):
            listOfDivisors.append(i)
    return listOfDivisors

#I mentioned you could do this in square root of a steps, here's how (either␣
↪one counts for credit).
def getDivisorsFaster(a):
    listOfDivisors = []
    for i in range(1,int(math.sqrt(a)+1)):
        if divides(i,a):
            listOfDivisors.append(i)
            listOfDivisors.append(a/i)
    return listOfDivisors

##### part (c)
def getCommonDivisors(a,b):
    #you could call getDivisors and compare lists, but it's a bit more␣
↪efficient I think to just populate a new list at once.
    listOfCommonDivisors = []
    for i in range(1,min(a,b)+1):
        if divides(i,a) and divides(i,b):
            listOfCommonDivisors.append(i)
```

```
    return listOfCommonDivisors

##### part (d)

#because getCommonDivisors returns a sorted list, we just need the top (or␣
↪-1st) element of the list.  If it were not sorted, you would have to sort␣
↪the list or at least go through it looking for the top element.
def findGCDSlow(a,b):
    return getCommonDivisors(a,b)[-1]

#a few tests
print("Divisors of 24:",getDivisors(24))
print("Divisors of 24:",getDivisorsFaster(24)) #notice this isn't sorted at the␣
↪moment
print("Common Divisors of 16 and 24:",getCommonDivisors(16,24))
print("gcd(2024,748) =",findGCDSlow(2024,748))
```

```
Divisors of 24: [1, 2, 3, 4, 6, 8, 12, 24]
Divisors of 24: [1, 24, 2, 12, 3, 8, 4, 6]
Common Divisors of 16 and 24: [1, 2, 4, 8]
gcd(2024,748) = 44
```

[3]:
```
########## Problem 2

##### part (a)
def divisionWithRemainder(a,b):
    #The trick here is to find the remainder first.  Then it becomes a simple␣
↪division problem.
    r = a%b
    #then solve a = bq+r for r
    q = (a-r)//b #Try to use // for integer division to avoid float errors
    return [q,r]

##### part(b)
def findGCDFast(a,b):
    while(b>0): #If the remainder hasn't yet become 0
        qr = divisionWithRemainder(a,b)
        #replace (a,b) with (b,r)
        a = b
        b = qr[1]
    #once we break out our remainder b=0, so the one before it is in position a.
    return a

#Notice we never really used q above.  This suggests the next recursive␣
↪algorithm.  That said, since we often work with huge numbers I would suggest␣
↪trying to avoid recursion if possible since you might exceed recursion depth.
def findGCDRecursive(a,b):
```

```
    #first find the remainder
    r = a%b
    if r==0:
        return b
    else:
        return findGCDRecursive(b,r)

#a few tests
print("divide 25 by 3: [q,r]=",divisionWithRemainder(25,3))
print("gcd(2024,748) =",findGCDFast(2024,748))
print("recursively gcd(2024,748) =",findGCDRecursive(2024,748))
```

```
divide 25 by 3: [q,r]= [8, 1]
gcd(2024,748) = 44
recursively gcd(2024,748) = 44
```

[12]:

```
########## Problem 3
#here we will need to remember q
def extendedGCD(a,b):
    #Each remainder can be computed in terms of a and b.  These placeholders␣
 ↪save the coefficients in the previous 2 remainders
    u0 = 1
    v0 = 0
    u1 = 0
    v1 = 1
    while(b>0): #If the remainder hasn't yet become 0
        #then do division with remainder
        qr = divisionWithRemainder(a,b)
        #replace (a,b) with (b,r)
        a = b
        b = qr[1]
        #compute the new cofficients
        u = u0 - qr[0]*u1
        v = v0 - qr[0]*v1
        #and shift them coefficients:
        u0 = u1
        u1 = u
        v0 = v1
        v1 = v
    #once we break out our remainder b=0, so the one before it is in position a.
 ↪  Therefore we want the coefficients associated to a as well, which are u0␣
 ↪and v0
    return [a,u0,v0]

#The algorithm outlined in the book does essentially this, except it doesn't␣
 ↪remember the v's noting that you can find them at the end.
```

```python
#a quick test
gcd,u,v = extendedGCD(2024,748)
print("[gcd,u,v] =",[gcd,u,v])
print("au + bv =",2024*u + 748*v)
```

```
[gcd,u,v] = [44, -7, 19]
au + bv = 44
```

[8]:

```python
########## Problem 4

##### part(a)
print("gcd(527,1258)")
print(findGCDSlow(527,1258))
print(findGCDFast(527,1258))
print(extendedGCD(527,1258))
print("\n")

##### part(b)
print("gcd(1056,228)")
print(findGCDSlow(1056,228))
print(findGCDFast(1056,228))
print(extendedGCD(1056,228))
print("\n")


##### part(c)
print("gcd(163961,167181)")
print(findGCDSlow(163961,167181))
print(findGCDFast(163961,167181))
print(extendedGCD(163961,167181))
print("\n")


##### part(d)
print("gcd(3892394,239847)")
print(findGCDSlow(3892394,239847))
print(findGCDFast(3892394,239847))
print(extendedGCD(3892394,239847))
print("\n")


##### part(e)
print("gcd(32715482947251,649917361940562)")
#print(findGCDSlow(32715482947251,649917361940562)) #Doesn't run in time
print(findGCDFast(32715482947251,649917361940562))
print(extendedGCD(32715482947251,649917361940562))
print("\n")
```

```python
##### part(a)
print("gcd(57993692894873334328961928359215776,375993729939672871359928438912)")
#print(findGCDSlow(57993692894873334328961928359215776,375993729939672871359928438912))
 ↪ Doesn't run in time
print(findGCDFast(57993692894873334328961928359215776,375993729939672871359928438912))
print(extendedGCD(57993692894873334328961928359215776,375993729939672871359928438912))
print("\n")
```

```
gcd(527,1258)
17
17
[17, -31, 13]


gcd(1056,228)
12
12
[12, 8, -37]


gcd(163961,167181)
7
7
[7, 4517, -4430]


gcd(3892394,239847)
1
1
[1, 59789, -970295]


gcd(32715482947251,649917361940562)
3
[3, 53354937663485, -2685776154786]


gcd(57993692894873334328961928359215776,375993729939672871359928438912)
32
[32, -1774150622414444425938744743, 273646973746237751512454653606925]
```

```
[0]:
```