Assignment 1: DS8001
Mezbah Uddin
ID: 500793378
October 17, 2016

Question 1

Pseudo Code	cost	Times
input :n	c1	1
If(n<10) ..	c2	1
count=0	c3	1
s=0	c4	1
while(True)	c5	n+2
for(i= 5 to count)	c6	60+(n-10)/2[18+(n-11)]
s=s+i	c7	48+(n-10)/2[16+(n-11)]
if(count>n)	c8	n+2
count+count+1	c9	n+1
return s	c10	1

$T(n) = (c1+c2+c3+c4+c10) + c6\ [60+(n-10)/2[18+(n-11)]] + c7[48+(n-10)/2[16+(n-11)]]+ (n+2)(c5+c8) + c9(n+1)$

$T(n)$ is $O[n^2]$

Best case is when $n<10$, $T(n)$ is just a constant and worst case is $O[n^2]$.

Assumption: the for loop first iterates reversely, then goes forward. That is 5 to 0 (7 times), 5 to 1,....5 to 5 and then 5 to 6 , 5 to 7 ... like that.

Question 2

T(n) has n^2 and log(log(n)) as two terms apart from constants. n^2 dominates log(log(n)) for larger n (as n approaches to infinity), therefore O(n), which gives us the upper bound is $O[n^2]$ and is also same as $\Theta(n^2)$.

Question 3:

Python Code for the greedy heuristic

```
Input for the program are f(subsets), x (universal set) and w(weight)
l=[]
for a,b in zip(f,w):
        ratio=len(a)/b
        l.append(ratio)

# algorithm will choose the subsets with higher ratio. For a given a weight a set with a higher coverage(covers unique points of x) is better.


import pandas as pd


p=pd.DataFrame({‘subset’:f,’ratio’:l,'setx':[u,u,u,u})

# creating data frame combining the lists


j=[]
for x in f:
        z=list(set(u)-set(x))
        j.append(z)
p[‘uncovered']=j

# creating an uncovered column in dataframe p, which will the compute the uncovered part of x, for each subset.



g=p.sort_values('ratio',ascending=False)
```

dataframe creates a cumulative or sum of a subset column after choosing each by the higher ratio. The following creates a definition to get a unique elements of a the sum of subset from each row. Unique elements are important because it may happen that additional subsets may not add as much unique elements as its length. Algorithm does this by creating a function,f2 and applying it to the cumulative subset column of the data frame.

```
def f2(seq):
  # order preserving
  checked = []
  for e in seq:
     if e not in checked:
        checked.append(e)
  return checked

g['cov_pct_cum']=g.subset.cumsum().apply(f2)
```

now we can to find the length of the unique elements that each cumulative subsets. We can do this until we have found the full set x.

```
g['char_length']=g['cov_pct_cum'].str.len()
```

at the last step, the algorithm does a query to find out the subsets based on ratios and adding unique elements until it entirely covers x.

```
print(g[g.char_length<=len(x)])
```

Using this sets
f=[[1,3,6],[9,0],[8,3,6,1],[1,3,7]]
x=[1,3,6,8,9,0,7]
w=[2,2,2,4]

output of the algorithm

	ratio	setx	subset	uncovered	cov_pct_cum	char_length
2	2.00	[1, 3, 6, 8, 9, 0, 7]	[8, 3, 6, 1]	[0, 9, 7]	[8, 3, 6, 1]	4
0	1.50	[1, 3, 6, 8, 9, 0, 7]	[1, 3, 6]	[0, 8, 9, 7]	[8, 3, 6, 1]	4
1	1.00	[1, 3, 6, 8, 9, 0, 7]	[9, 0]	[8, 1, 3, 6, 7]	[8, 3, 6, 1, 9, 0]	6
3	0.75	[1, 3, 6, 8, 9, 0, 7]	[1, 3, 7]	[0, 8, 6, 9]	[8, 3, 6, 1, 9, 0, 7]	7

(b) worst case of the algorithm would be X is very large set, therefore each for loop would run to check each entry of the subset exist in X. Therefore worst case would be a very large X and small F's. So, the big O is O[X]

(c) Worst case would be large number of subsets and each subsets containing few unique elements of X. In this case, the algorithm wouldn't be very efficient because in the worst case, no matter what the ratio is, it is left with choosing all the subsets to cover X.
Assuming F set contains z subsets, the worst case would be $O[F^2]$, quadratic t(n).

Question 4

R script

```
a<-2147483647
count<-0
for(x in seq(1,50,by=1)){
 z<-(2^x-1)
 count=count+1
 if(z==a){
  print(z)
  print("This is integer$max")
  print(paste0("its"," 2^",count,"-1"))
 }
 }
```

Output:

```
[1] 2147483647
[1] "This is integer$max"
[1] "its 2^31-1"
```

Question 5

The statement is False.

Solution for the $Ax=b$ is $x^* = A^{-1} b$.

Now if we choose b to be double precision rather than a single precision, b would contain more floating point (decimal points) and so as norm of δb. And this will lead to a higher $\kappa(A)$ than $\kappa(A)$ with single precision floating points.

And a higher $\kappa(A)$ would result in larger ratio, therefore larger variability in predicting x.

so, double precision $\kappa(A)$ > single precision $\kappa(A)$.

Question 6

if-else condition block will provide the least efficient solution, as it checks the condition each time once and stores nothing in the memory to better match the next condition.

Switch has a better efficiency than multiple if-else block as it stores a index in memory to check which condition/s is/are true. O in this case is just O(n)

However, I would recommend a binary search tree than switch, because it checks conditions faster, as it stores items in the memory and allow fast lookup using lookup tables . And it has O(log(n)). As n approaches to infinity n goes faster to infinity than log(n). Therefore, it is optimal to choose binary search tree based on efficiency.