**WEEK-2**

**R-3.8** :

**4n logn+2n**

4n logn+2n <= c nlogn for c<= 1

4n logn = O(nlogn)

**2n**

2n <= c \* 2n for c<=1

2n = O(2n)

**2logn**

2logn = n <= c\*n for c<=1

2logn = O(n)

**210**

210 <= c for c<= 1

210 = O(1)

**3n+100logn**

3n+100logn <= c\*n for c<= 1

3n+100logn = O(n)

**4n**

4n <= c\*n for c>= 4

4n = O(n)

**N2+10n**

N2+10n <= cn2 for c>1

N2+10n = O(n2)

**N3**

N3 <= c n3 for c >= 1

N3 = O(n3)

**Nlogn**

Nlogn <= cnlogn for c>= 1

Nlogn = O(nlogn)

O(1)[210] < O(n)[2logn] < O(n)[3n+100logn] < O(n)[4n] < O(nlogn)[nlogn] < O(nlogn)[4nlogn+2n] < O(n)[n2+10n] < O(n3)[n3] < O(2n)[2n]

**C-3.35** :

If we assume that 3 sets are mathematical sets (elements vary between each set), just add 3 sets and arrange them, then break the list in sequence and check if there are 3 times the same thing. Time complexity is controlled by filtering, which is O (n log n). The complexity of the auxiliary space reaches O (n).

**C-3.42** :

* Consider the sum of the maximum number of visits each friend can make without visiting their maximum number of times.
* The number is one more than the total number of visits each friend can make while still being able to make one more allowed visit, that is, the sum where each friend i visits i−1times. In other words, the minimum value for C such that Bob should know that one of his friends has seen his/her maximum allowed number of times is n(n−1)/2+1.

**C-3.57** :

import time

import numpy as np

import matplotlib.pylab as plt

def do\_sorted(n):

bignum = 1E6

A = np.random.randint(0,bignum,n)

A = sorted(A)

plot\_n = []

plot\_walltime = []

for j in np.linspace(1E3,1E5,50):

n = int(j)

start = time.clock()

do\_sorted(n)

end = time.clock()

print("Sorting array of size {0:d} took {1:.6f}".format(n,(end-start)))

plot\_n.append(n)

plot\_walltime.append(end-start)

plt.plot(plot\_n,plot\_walltime,'o-')

plt.show()

**R-4.1** :

findMax(s, n):

if(n==1):

Return s[0]

else:

Return max(s[n-1], findMax(a, n-1))

Time Complexity :- O(n)

Space Complexity :- O(n)

**R-4.3** :

power(2,5) — return 2\*16 = 32

power( 2,4) — return 2\*8 = 16

power(2,3) — return 2\*4 = 8

power(2,2) — return 2\*2 = 4

power(2,1) — return 2\*1 = 2

power(2,0) — return 1

**C-4.9** :

**Function to find the minimum**

Def findMin(s,n):

if(n==1):

Return s[0]

Return min(s[n-1], findMin(s, n-1))

**Function to find maximum**

Def findMax(s,n):

if(n==1):

Return s[0]

Return max(s[n-1], findMax(s, n-1))