**Time-Complexities**

In this document, we will analyze the time complexity of every function in our project, determining the complexity of every step and adding them up.

1. def openbookfile(name):

import csv O(1)

with open(name) as csv\_file: O(1)

x=csv.reader(csv\_file) O(n)

d=tuple() O(1)

F=[] O(1)

for y in x: O(m)

for z in y: O(p)

d+=(z,) O(1)

F.append(d) O(1)

d=tuple() O(1)

bookdict={} O(1)

for x in F: O(m)

bookdict[x[0]]=[x[2].split(','),x[1],x[3]] O(1)

return bookdict O(1)

if we add up the complexities, it would give, . (where n is the length no. of rows in csv file, m is the no. of columns and p is the no. of comma separated elements in one cell.).

1. def userdataload(name):

import csv O(1)

with open(name) as csv\_file: O(1)

x=csv.reader(csv\_file) O(n)

d=tuple() O(1)

F=[] O(1)

for y in x: O(m)

for z in y: O(p)

d+=(z,) O(1)

F.append(d) O(1)

d=tuple() O(1)

bookdict2={} O(1)

for x in F: O(m)

if x!=(): O(1)

final={} O(1)

for y in range(len(x)): O(p)

if y!=0: O(1)

M=x[y].split(',') O(1)

o=M[1] O(1)

final[M[0]]=eval(o) O(1)

bookdict2[x[0]]=final O(1)

return bookdict2 O(1)

Adding up the complexities would give us, .

1. def WeightedEdge\_Create(datadict):

final\_lst=[] O(1)

alrdy\_traversed = [] O(1)

for person in datadict.keys(): O(V)

alrdy\_traversed.append(person) O(1)

books\_read=datadict[person] O(1)

for neighbor in datadict.keys(): O(V)

weight=0 O(1)

if neighbor in alrdy\_traversed: O(1)

continue O(1)

for n\_books in datadict[neighbor].keys(): O(v)

if n\_books in books\_read.keys(): O(v)

currentchoice=books\_read[n\_books] O(1)

neighborchoice=datadict[neighbor][n\_books] O(1)

if currentchoice==neighborchoice: O(1)

weight+=1 O(1)

else: O(1)

weight-=1 O(1)

if weight != 0: O(1)

final\_lst.append((person, neighbor, weight)) O(1)

return final\_lst O(1)

Summing up would give Time complexity as, where V is the number of vertices of main dictionary and v is the no. of vertices of nested dictionary.

1. def GetMeADuo(G,name):

Final=[] O(1)

a=G[name] O(1)

t=0 O(1)

for x in a: O(n)

if x[1]>t: O(1)

Final.append(x[0]) O(1)

t=x[1] O(1)

if Final==[]: O(1)

print('You have no match :c') O(1)

else: O(1)

print('Your highest duo score is with ',end='') O(1)

print(Final[-1]+'.') O(1)

return Final O(1)

the time complexity of this function would be O(n) only.

1. def recommend\_genre(bookdata,name,c):

global genre\_output O(1)

try: O(1)

userbooks=c[name] O(1)

except: O(1)

genre\_output = 'User record not found!' O(1)

return O(1)

for x,y in userbooks.items(): O(V\*E)

if y==True: O(1)

genre=bookdata[x][0] O(1)

author=bookdata[x][1] O(1)

price=bookdata[x][2] O(1)

rec=[] O(1)

for gen in genre: O(g)

for i,j in bookdata.items(): O(v\*e)

if gen in j[0] and i not in rec and i!=x: O(w\*r)

rec.append((i,j[2])) O(1)

for x in rec: O(K)

genre\_output+= x[0] + ' for $' + x[1] + '\n' O(1)

return O(1)

The total time complexity would be, Where V are the vertices of main dictionary, E are the Edges of main dictionary, g is the length of genre of initial book of the user, v and e are the vertices and edges of Book data dictionary and lastly, w and r are the length of genre of each book in book data dictionary, whereas r is the length of rec list.

1. def save\_new\_entries(user\_name, user\_books):

global new\_user\_data O(1)

new\_user\_data=[user\_name] O(1)

for x in user\_books: O(n)

new\_user\_data.append(x) O(1)

for this function, complexity would be simply .

1. def New\_User(datafile,new\_user\_data):

import csv O(1)

with open(datafile, 'a') as csvFile: O(1)

writer = csv.writer(csvFile) O(1)

writer.writerow(new\_user\_data) O(n)

Time complexity would be, O(n).

1. def Add\_to\_records(datafile, user\_name, user\_books):

global new\_user\_data O(1)

save\_new\_entries(user\_name, user\_books) O(n)

New\_User(datafile,new\_user\_data) O(n)

The complexity for this function would be, .

1. def top\_picks(G, name):

global user\_data O(1)

links = G[name] O(1)

counter = 0 O(1)

toplist = [] O(1)

while counter <2: O(E)

maxval = 0 O(1)

max\_index = None O(1)

for person in range(len(links)): O(E)

if links[person][1] > maxval: O(1)

maxval = links[person][1] O(1)

max\_index = person O(1)

if max\_index: O(1)

toplist.append(links.pop(max\_index)[0]) O(1)

counter +=1 O(1)

else: O(1)

break O(1)

recommendation = [] O(1)

for connection in toplist: O(2)

read\_books = user\_data[connection] O(1)

for book in read\_books.keys(): O(V)

if book not in user\_data[name].keys() and read\_books[book]: O(V)

recommendation.append(book) O(1)

return recommendation O(1)

The complexity for this function would be, . Where E would be length of value of user links dictionary and V is the number of users of library.

1. def show\_top\_picks(G, name, book\_data):

global top\_picks\_display O(1)

top\_picks\_display = '' O(1)

try: O(1)

picks = top\_picks(G, name)

for entry in picks: O(1)

top\_picks\_display+= entry+'\t Genres: '+str(', '.join(book\_data[entry][0]))+'\n' O(1)

except: O(1)

top\_picks\_display = 'User record not found!' O(1)

return O(1)

The complexity would be same as the top\_picks function (as it is basically calling that function), hence it would .