1 Problem 1:

1. Write a Python program that extracts 1000 unique links fromTwitter.

Also note that you need to verify that the final target URI (i.e., the one that responds with a 200) is unique. You could have different shortened URIs for www.cnn.com.
For example,
http://cnn.it/1cTNZ3V
http://t.co/BiYdsGotTd

Below is the Python coded used to extract 1000 unique URIs and ensure the links are valid and unique.
I used an adapted code from Craig Addyman (http://www.craigaddyman.com/mining-all-tweets-with-python/)
I've included a the uniqueURI file in the githubb

Listing 1: extractTwitter.py

```
1  #extractTwitter.py
2  # Extracts 1000 URIs from timoreilly twitter page
3  # and prints its output to a file "uniqueURI"
4
5  from twython import Twython #http://twython.readthedocs.org/en/latest/
6      # used http://www.craigaddyman.com/mining-all-tweets-with-python/
7  import time
8  import requests
9  from bs4 import BeautifulSoup
10
11 REQUEST_TOKEN_URL = "https://api.twitter.com/oauth/request_token"
12 AUTHORIZE_URL = "https://api.twitter.com/oauth/authorize?oauth_token="
13 ACCESS_TOKEN_URL = "https://api.twitter.com/oauth/access_token"
14
15 CONSUMER_KEY = "K7ddHMoZcHXu0WyWdnp6wdQxf"
16 CONSUMER_SECRET = "OcvSpsGJ5e1f9GvHbT1y5TXLzwETVkL9wkd1WDzL3FHgC7WDzk"
17
18 ACCESS_KEY = "2539409241-FFZPA5lh2vZklUx3q5jASFesRVVXPxqfS6nY8QB"
19 ACCESS_SECRET = "zzQVXih1Gy2cR1WdMtcMhmXbWpi1S45Jc7vlLlkHFWwm9"
20
21 twitter = Twython(CONSUMER_KEY,CONSUMER_SECRET,ACCESS_KEY,ACCESS_SECRET)
22 lis = [] ## this is the latest starting tweet id
23
24 unique_URIs = []
25
26 ################################################
27 #  This function tests to ensure the link is
28 #       unique and ensure it's a valid site
29 ################################################
30
31 def testUniqueURL(link):
32   try:
33     r = requests.head(link, allow_redirects=True)
```

```
34      if r.status_code == 200: #ensure valid website
35
36          #tests to see if URI already saved in link
37          if r.url in unique_URIs:
38              return [False, ""]
39          else:
40              return [True, r.url]
41      return [False, ""]
42    except requests.ConnectionError:
43      return [False, ""]
44
45
46  ################################################
47  #  Extracts 1000 links from timoreilly's twitter
48  ################################################
49  def get1000links():
50      while True:
51          uri = ""
52      ## tweet extract method with the last list item as the max_id
53          user_timeline = twitter.get_user_timeline(screen_name="timoreilly",
54          count=200, include_retweets=False, max_id=lis[-1])
55
56          for tweet in user_timeline:
57              lis.append(tweet['id'])
58              soup = BeautifulSoup(Twython.html_for_tweet(tweet))
59              uri = soup.find('a', 'twython-url')
60              if uri:
61                  uri = uri.get('href')
62                  unique = testUniqueURL(uri)
63                  if unique[0]:
64                      unique_URIs.append(unique[1])
65                      print 1000 - len(unique_URIs)
66                      if len(unique_URIs) == 1000:
67                          return
68          print "Sleep for 5 minutes"
69          time.sleep(300) ## 5 minute rest between api calls
70  ################################################
71
72  user_timeline = twitter.get_user_timeline(screen_name="timoreilly",count=1)
73  #print user_timeline
74
75  #print user_timeline[0]['id_str']
76  lis = [(user_timeline[0]['id_str'])]
77
78  get1000links()
79
80  f = open('uniqueURI', 'w')
81
82  for link in unique_URIs:
```

```
83          f.write(link)
84          f.write('\n')
85
86  f.close()
```

2 Problem 2:

Download the TimeMaps for each of the target URIs. We'll use the mementoweb.org Aggregator, so for example:

URI-R = http://www.cs.odu.edu/
URI-T = http://mementoweb.org/timemap/link/http://www.cs.odu.edu/
You could use the cs.odu.edu aggregator:
URI-T = http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/http://www.cs.odu.edu/

But be sure to say which aggregator you use – they are likely to give different answers. Create a histogram of URIs vs. number of Mementos (as computed from the TimeMaps). For example, 100 URIs with 0 Mementos, 300 URIs with 1 Memento, 400 URIs with 2 Mementos, etc.

See: http://en.wikipedia.org/wiki/Histogram
Note that the TimeMaps can span multiple pages.

I used two Bash scripts to solve this problem.

The first one downloads the TimeMaps for each URI using memento.web.org. It will also check to see if if there is a next page and extract the data from that page as well.

Listing 2: getMementos.py

```
1  #!/bin/bash
2  # gets mementos for all links
3  # execute: cat uniqueURI | ./getMementos
4
5  counter=1
6  while read line
7  do
8      echo "$counter"
9      link="1"
10     while [ 1 ]
11     do
12         #get the mementos from mementoweb.org
13         wget "http://mementoweb.org/timemap/link/$link/$line" -O - >> "
               $counter"
14         link=$(($link+999))
15
16         ret=`grep "http://mementoweb.org/timemap/link/$link/" $counter`
17
18
19         #if there is a link to a second page, repeat and extract the mementos
               from that page
20         # if no second page, break
21         if [[ $ret -eq "" ]]; then
```

```
22                  break
23              fi
24          done #continue to next URI
25          echo $counter
26
27          ((counter++))
28  done
```

The second one uses "grep" to find each line with mementos in it and uses "wc -l" to count each line for the number of occurances.

Listing 3: numMementos.py

```
1  #!/bin/bash
2  # read each memento file, extract the number of mementos, output in a file
3
4  rm memento.dat
5
6  for (( i=1; i<=1000; i++ ))
7  do
8      numMen=$(cat $i | grep 'rel="memento";' | wc -l)
9      printf "$numMen\n" >> memento.dat
10 done
```

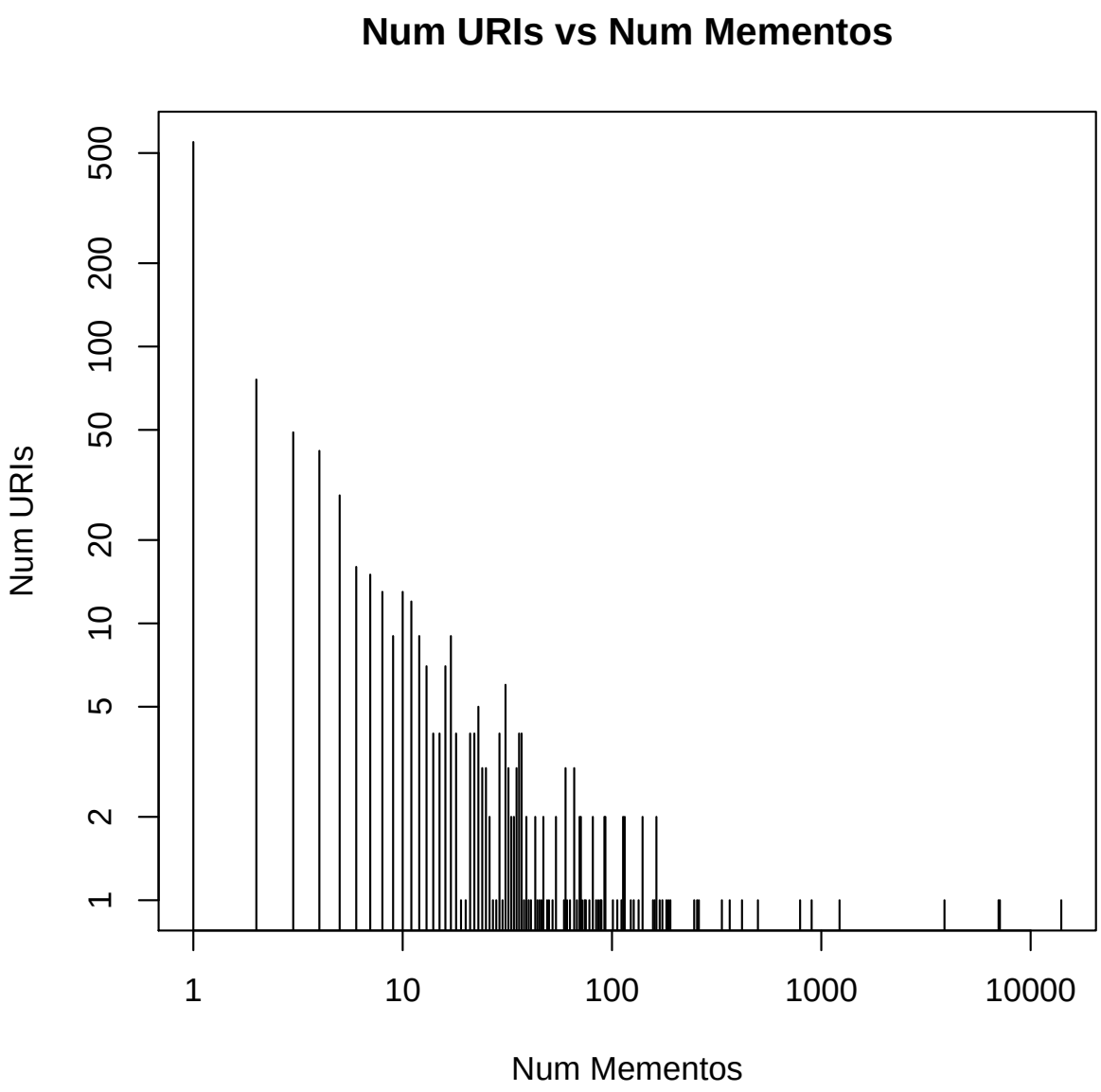

Figure 1: Above is a histograph: URIs vs. number of Mementos. I had to use a log log scale to shore the relationship between the two. This shows that most URI have a few mementos. This appears to be because twitter is a social media site and in this case Tim O'Reilly was posting recent websites. The website with the more mementos was www.healthcare.gov

3 Problem 3:

Estimate the age of each of the 1000 URIs using the "Carbon Date" tool:

http://ws-dl.blogspot.com/2013/04/2013-04-19-carbon-dating-web.html

Note: you'll have better luck downloading and installing the tool rather than using the web

service (which will run slowly and likely be unreliable).

For URIs that have more than 0 Mementos and an estimated creation date, create a graph with age (in days) on one axis and number of mementos on the other.

This was probably the hardest part of the assignment. Not the problem itself, but executing. I ended up modifying the local.py to add threading and executed 10 links at a time. This reduced run time from 12+ hours to about 45 minutes Below is the modified code.

Listing 4: local.py

```
#adapted local.py
#used Mathew Chaney code to help figure out how to use Threads via futures.
    This significantly reduced run times.

import json
from ordereddict import OrderedDict
import json as simplejson

import Queue
import threading
import futures
import urllib2

import re

from getBitly import getBitlyCreationDate
from getArchives import getArchivesCreationDate
from getGoogle import getGoogleCreationDate
from getBacklinks import *
from getLowest import getLowest
from getLastModified import getLastModifiedDate
from getTopsyScrapper import getTopsyCreationDate
from htmlMessages import *
from pprint import pprint

from threading import Thread
import Queue
import datetime

import os,sys, traceback


NUMTHREADS = 10 # the amount of threads you want to run! Don't use too many
    otherwise your computer will lock up


def cd(url, fname, backlinksFlag = False):
    fn = 'estCreation2/'+str(fname)
```

```
37      fout = open(fn, 'w+')
38      print 'Getting Creation dates for: ' + url
39
40      threads = []
41      outputArray =['','','','','','']
42      now0 = datetime.datetime.now()
43
44
45      lastmodifiedThread = Thread(target=getLastModifiedDate, args=(url,
            outputArray, 0))
46      bitlyThread = Thread(target=getBitlyCreationDate, args=(url, outputArray,
            1))
47      googleThread = Thread(target=getGoogleCreationDate, args=(url, outputArray
            , 2))
48      archivesThread = Thread(target=getArchivesCreationDate, args=(url,
            outputArray, 3))
49
50      if( backlinksFlag ):
51          backlinkThread = Thread(target=getBacklinksFirstAppearanceDates, args
                =(url, outputArray, 4))
52
53      topsyThread = Thread(target=getTopsyCreationDate, args=(url, outputArray,
            5))
54
55
56      # Add threads to thread list
57      threads.append(lastmodifiedThread)
58      threads.append(bitlyThread)
59      threads.append(googleThread)
60      threads.append(archivesThread)
61
62      if( backlinksFlag ):
63          threads.append(backlinkThread)
64
65      threads.append(topsyThread)
66
67
68      # Start new Threads
69      lastmodifiedThread.start()
70      bitlyThread.start()
71      googleThread.start()
72      archivesThread.start()
73
74      if( backlinksFlag ):
75          backlinkThread.start()
76
77      topsyThread.start()
78
79
```

```
80      # Wait for all threads to complete
81      for t in threads:
82          t.join()
83
84      # For threads
85      lastmodified = outputArray[0]
86      bitly = outputArray[1]
87      google = outputArray[2]
88      archives = outputArray[3]
89
90      if( backlinksFlag ):
91          backlink = outputArray[4]
92      else:
93          backlink = ''
94
95      topsy = outputArray[5]
96
97      #note that archives["Earliest"] = archives[0][1]
98      try:
99          lowest = getLowest([lastmodified, bitly, google, archives[0][1],
                backlink, topsy]) #for thread
100     except:
101        print sys.exc_type, sys.exc_value , sys.exc_traceback
102
103
104
105     result = []
106
107     result.append(("URI", url))
108     result.append(("Estimated Creation Date", lowest))
109     result.append(("Last Modified", lastmodified))
110     result.append(("Bitly.com", bitly))
111     result.append(("Topsy.com", topsy))
112     result.append(("Backlinks", backlink))
113     result.append(("Google.com", google))
114     result.append(("Archives", archives))
115     values = OrderedDict(result)
116     r = json.dumps(values, sort_keys=False, indent=2, separators=(',', ': '))
117
118     now1 = datetime.datetime.now() - now0
119
120
121     #print "runtime in seconds: "
122     #print now1.seconds
123     #print r
124     fout.write( 'runtime in seconds:  ' +  str(now1.seconds) + '\n' + r + '\n'
            )
125
126     fout.close()
```

```
127     print str(fname) + "␣complete**************************\n"
128     return r
129
130
131
132 f=open("uniqueURI", 'r')
133 theurls = f.readlines()
134 count = 1
135
136
137 q = Queue.Queue()
138
139 with futures.ThreadPoolExecutor(max_workers=NUMTHREADS) as executor:
140     for u in theurls:
141         urifutures = executor.submit(cd, u, count)
142         count = count+1
143
144     for future in futures.as_completed(urifutures):
145                 try:
146                     data = future.result()
147                 except Exception as exc:
148                      print "{}␣generated␣an␣exception:␣{}".format(u, exc)
```

The next part I used python to print the estimated days alive per link based on the estimated creation date from local.py. Sites without an estimated creation date are listed as "0"

Listing 5: creationdate.py

```
1  # open each creation file and pull the creation date
2  # create a file called "creation_date"
3  # file output:
4  #     <fileNumber> '\t' <age in days>
5
6  import time
7  import calendar
8
9  # returns the number of days based on a date (Day Month Year)
10 #  received.
11 def age((year, month, day)):
12     days = year*365                  # years, roughly
13     days = days + (year+3)//4        # plus leap years, roughly
14     days = days - (year+99)//100     # minus non-leap years every century
15     days = days + (year+399)//400    # plus leap years every 4 centirues
16     for i in range(1, month):
17         if i == 2 and calendar.isleap(year):
18             days = days + 29
19         else:
20             days = days + calendar.mdays[i]
21     days = days + day
22     return days
23
```

```
24 fout = open('creation_date', 'w+')
25
26 for x in range(1,1001): #open every file
27     try:
28         f = open(str(x), 'r')
29     except:
30         print "ERROR"
31         continue
32
33     date = f.readlines()
34     #print x
35
36     if not date:
37         fout.write(str(x) + "\t" + "0\n")
38         continue
39
40     sub = "\"Estimated Creation Date\": "
41
42     for string in date:
43         if sub in string:
44             age = string.split('"')[3]
45             if (age == ""):
46                 fout.write(str(x) + "\t0\n")
47             else:
48                 age=age.split("T")[0]
49                 age=age.split("-")
50
51                 year = int(age[0])
52                 month = int(age[1])
53                 day = int(age[2])
54
55                 estCreation = (year, month, day)
56                 daysalive =  age(estCreation)
57
58                 todaydate = time.localtime()[:3]
59                 todaydate = age(todaydate)
60
61                 fout.write(str(x) + "\t" + str(todaydate-daysalive)+"\n") #
                       prints age to the file
62
63
64 fout.close()
```

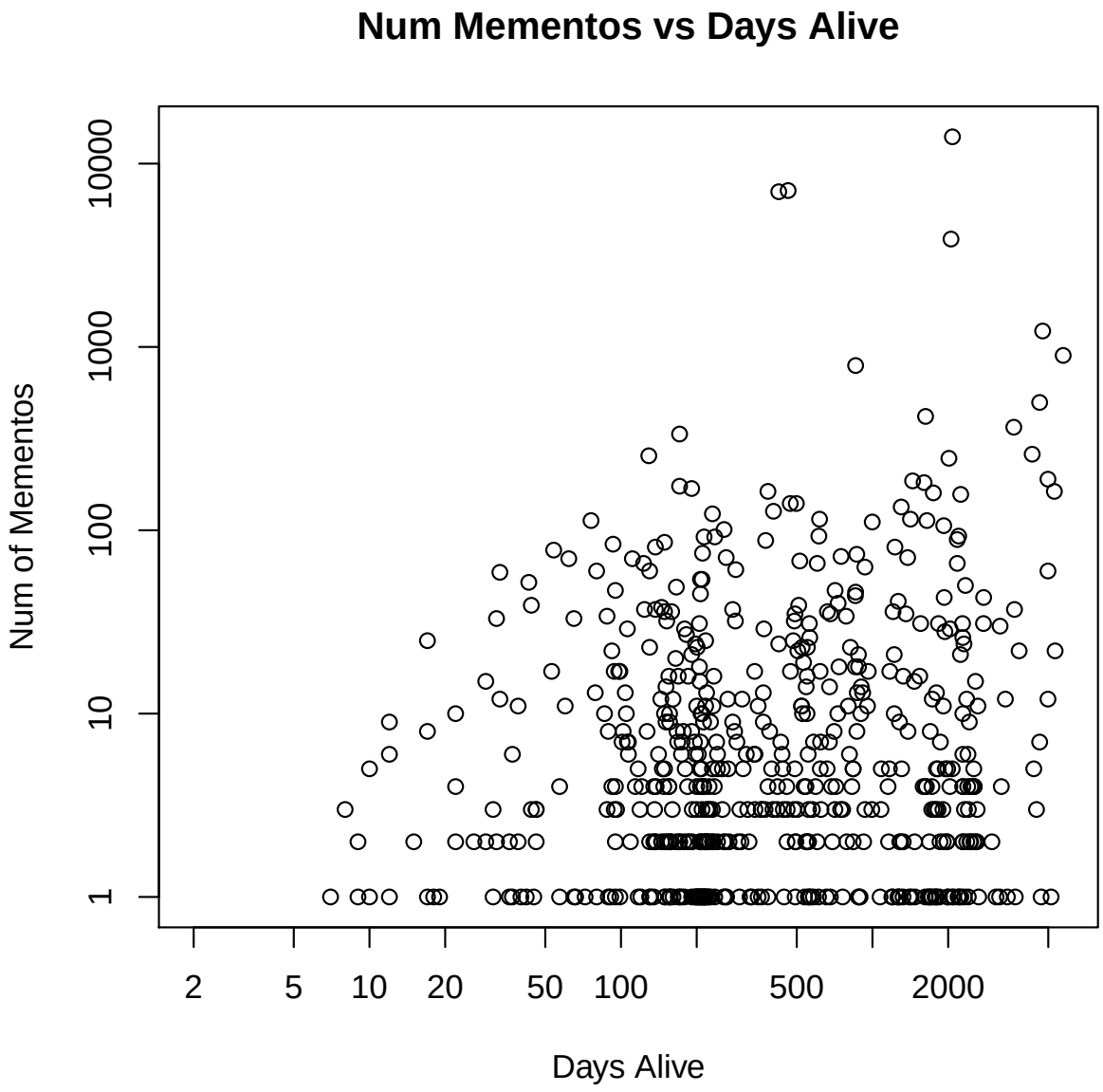

Figure 2: Above is a plot to show the relations between the nmber of mementos vs days alive. Again, in order to see the relationship, I had to use a log-log scale. As expected, the longer ago the website was created the more mementos were created.