# CIS 122 Fall 2015 Project 7 read, search files, estimating times
# Due Monday, Feb 29, 11:59 PM

**Introduction to linear and binary searching**

This project focuses on estimating how long it takes to search several medium size files, all in a sorted order.

**Linear search**

A "smart" linear search will stop searching once it encounters an item that is past the spot it is looking for.

For example, looking for "Aaron" can stop when it encounters "Abagail". Abagail is beyond Aaron and the file is sorted by name, so search cannot find Aaron.

**Binary Search ("Divide and Conquer")**

Binary repeatedly splits the list of data items into two halves, looking only in the half that could contain the item searched for.

With a file of some thousands of names, a "binary" search splits the list to check in half over and over until it either finds the item, or concludes it is not in the list.

Results of searching a list of 12,398 items for "jordan":

| Try: | Found: | Note: |
|---|---|---|
| 6198 | khristy | Use earlier half next (about 6000 items) |
| 3098 | dia | Use later half next (about 3000 items) |
| 4648 | haskel | later |
| 5423 | jillayne | later |
| 5810 | kally | earlier |
| 5616 | jonnie | later |
| 5713 | judy | earlier |
| 5664 | josphine | earlier |
| 5640 | josef | earlier |
| 5628 | jorge | earlier |
| 5622 | jordan | Can't stop yet; need to find earliest jordan |
| 5619 | jonpaul | later |
| 5620 | jonquil | later |
| 5621 | jonthan | later |
| 5622 | jordan | Got it! |

Does binary search really save you much time when you are using a fast modern computer?

Counting the various attempts done by the binary search will reveal how many probes of the data list it takes.

By contrast, a linear search would start with the first item in the list and continue asking "is this is?" until it has either reached "jordan" at location 5622, or come to a word alphabetically past "jordan" (meaning jordan is not in the list).

We will use "microseconds" as a unit of measurement of time.
1 million microseconds equals 1 second.

**P7_comments1.py**
**5 points**

Note that this part does not involve programming; you will submit a series of Python language comments giving your observations.

Answer the following question, using approximate values.
The data is sorted in order.

If a binary search of about 12,000 items takes **b units of time**, roughly how many units of time would a binary search need to search about 78,000 items?

If a linear search of about 12,000 items takes **n units of time**, roughly how many units of time would a linear search need to search about 78,000 items?

**P7_estimates.py**
**5 points**

Comment on this statement:

"Sometimes, a linear search can beat a binary search."

Again, assuming you have written a program to do both kinds of searches, for which, if any, of these names might that statement above be true?

a) Search for **zoe**
b) Search for **melissa**
c) Search for **aaron**
d) Search for **king**

Explain your answer.

**P7_timing1.py**
**12 points**

You will work with short sorted files.

Remember to do this at the beginning of your program:

```
import time
```

Issues:
a) When timing processes that take little time, be sure to avoid timing non-essentials that take lots of time.
**print(***anything***)** takes some 250,000 microseconds (1/4th of a second) to print. Don't time any sequence of statements that include a print call.

b)
```
start = time.time()  # works on Macs, iffy
                     # on Windows
```

Instead, use this:

```
start = time.clock( ) # OK on Windows, Macs
```

Your task – read the file **yob_97_short.txt**
Each line of text resembles this

```
Ashley,F,1997,20893,3
```

name, F for Female, year, number of girls with this name this year, rank (1 is most popular name for girls, in this case Ashley is the 3rd most popular name for girls in 1997).

You need at least two search functions

**linear_start** given a **names** list (each entry is a sub-list) and a **search_name** to search for, will return the **index** of the first name matching the search name.

**binary_start** given a **names** list (each entry is a sub-list) and a **search_name** to search for, will return the **index** of the first name matching the search name.

Both of these functions return **-1** when no match is found.

You'll need to decide on how to retrieve all the names that match the search_name; start with the index returned by either the linear or binary search.

Insert timing into the beginning and end points of your linear and binary search start functions.

Print the time used in microseconds.

Here's an example of some timings using a similar file:

```
Choose your search method
Q Quit
M Match name
B Binary search
Type the option you want: m
Name to look for: angelina

Linear search took      182.0 microseconds
Year Name ......... MF Number  ..Rank
1967 Angelina       F     422     460

Choose your search method
Q Quit
M Match name
B Binary search
Type the option you want: b
Name to look for: angelina

Binary search took       25.0 microseconds
Year Name ......... MF Number  ..Rank
1967 Angelina       F     422     460


Choose your search method
Q Quit
M Match name
B Binary search
Type the option you want: m
Name to look for: zori

Linear search took    3,582.0 microseconds
```

What happens when you test with **aaron** and **zoe?**

**At the end of your program, it should print your conclusions about**

**linear search times - what affects them**

**binary search times - what affects them**

**Here is a "standard" binary search;** you must modify it so that it extracts a name from a sublist in the main list and makes its comparisons using that name.

```
def binary_search(item, the_list):
    """ str, list -> int
        Do binary search for item in the_list
        if in list, return index of first matching item in list
        else return -1
    """
##    debug = False
    word = ''
    mid = 0
    left = 0
    right = len(the_list) - 1
##    if debug: print("look for", item)
    while left <= right:
        mid = (left + right) // 2  # integer result //
        word = the_list[mid]
        if debug: print("mid", mid, word, left, right)
        if word < item:
            # Search just the right area (highest) ; adjust left
            left = mid + 1
##            if debug: print("left now  ", left, mid, "right still", right)
        else:
            # Search just the left area (lowest); adjust right
            right = mid - 1
##            if debug: print("left still", left, mid, "right now  ", right)
        #end if
        if debug: input("\nPress Return to continue\n")
    #end while

    if left >= 0 and left < len(the_list) and the_list[left] == item:
##        if debug: print("First item found", item, '**', left, mid, right)
        return left
    else:
##        if debug: print("Not found", word, item, '**', left, mid, right)
        return -1
    #end if
#end def
```

Rubric 12 points total

2 points - linear search returns index of first matching name, or -1 if no match found

Tested with names **aaron, anne, melissa, tom, zoe**

2 points - binary search returns index of first matching name, or - 1 if no match found.

Tested with names **aaron, anne, melissa, tom, zoe**

2 points - you have correctly modified binary search to get matching data from the name item in the sublist.

2 points - You print your conclusions about what factors influence the speed of a linear search

2 points - You print your conclusions about what factors influence the speed of a linear search.

2 point - Modify the program to work with the file **`yob_97.txt`** instead of the short test file. Remember not to print the entire file!

*You should only need to change the filename in your program; everything else should work fine, but you will have a larger list of names; some 27,000 instead of around 50.*

*Submit both the short and full versions of your program.*

**P7_timings2.py**

**8 points**

Copy your program from P7_timings1.py and modify it to work with the file **`yob_1950_2010.txt`**

This change will bring your list of names up to about 1.2 million.

Each name will often show up in each of the four years; your data is sorted by name, and within name, by F and M (gender), and withing gender by year.

"**Back of the envelope calculations**"
This means crude estimates (the kind you might do in pencil on a scrap of paper or the back of a small envelop).

Quick crude estimates
Roughly, 30,000 names in 1997 data.
How many doublings of 30,000 will get you to a point where the result exceeds 1.2 million?

Let n be about 30,000.

How many times larger , roughly, is 1,200,000?

Searching for a name starting with a z such as **zoe** is close to a worst case for a linear search. How many times slower, roughly, would you expect your linear search to be?

**Coffee break time?**
Will a linear search for zoe or zelda slow your machine down to where you notice a long pause, or slow it down to where you can get a cup of coffee while you wait for the search to complete?

**Binary is better?**
The theory is that binary should usually run far, far more quickly than a linear search, but does it run into some unforeseen limitations?

**Scaling**

**a) Performance**

Is linear search still viable for a machine with a single user?

Would linear search on this million–plus names work well for a server with hundreds of users seeking data on trends in names?

Same questions, but for binary searches.

**b) Display and analysis of results**

Many names such as Jordan have 60 years of data for both F female and M male results, giving over 120 lines of data.

Without some additional analysis the results can feel more like a flood of data.
Some questions: which year was the name at its best rank – 1 is most popular, 3000 is 3000th in popularity?

What year was it least popular?

How many girls had this name over the 60 year time span? How many boys?

Some names look like data entry errors, someone typing the wrong gender. Did parents really name their boy Melissa, or their girl Sam? You could consider suppressing a gender with less than 5% of the number of the opposite sex in a given year.

Could you explore the data to detect cultural trends? One year, the Beatles released a song about Michelle. Do you see effects on the popularity of that name?

Rubric

**1 points**
Run your program on the yob_1950_2010.txt file.

Answer the following question with print statements at the end of your program (after user selects Quit):

**2 points**
What kind of variation in timings do you find for **linear** searches for:

**aaron**
**jane**
**max**
**paul**
**zoe**

**1 points**
Compared to searches on smaller files, how does linear search scale?

Gets slower faster than the file grows
Gets slower in proportion to file size
Gets slower, but at a slower rate than the increase in file size.

**1 point**
How can you characterize the change in speed of binary searches compared to changes in file sizes?

**1 point**
For strictly personal use by one person at a time, is linear search fast enough?

**1 point**
If your file grew to 10 times its current size, would search time become an issue?

**1 point**
If you were using your program on a web server with 10 or more users per second, which kind of search would you need?

**Bonus** 4 points

For this large **`yob_1950_2010.txt`** file, provide some **summary** data such as

```
1964 Jacqueline    F  11,973     37 Best rank
1964 Jacqueline    F  11,973     37 Highest number, best rank
2010 Jacqueline    F   1,805    171 Lowest number, worst rank
```

Ask for a range of years to print such as 1960-1965.