10-07 Lecture:

Announcements:

- True ups done by hopefully Wednesday afternoon.
- QOTD13 will be the hardest one.. Iteration
- Lab5 will have some iteration and recursion

```
return( ' '.join([c for c in w if c.lower() in 'aeiou' ] for w in S.split()
]))
```

Palindromes:

```
def palindrome(s):
    if len(s)<2:
        return(True)
    elif s[0]!=s[-1]:
        return(False)
    else:
        return(palindrome(s[1:-1]))
#these behave different. How? Recursion forumla is better because it has
chance of failing early, instead of finishing the whole amount of
charchters.
def palindrome(s):
    return(len(s)<2 or s[0]==s[-1] and palindrome(s[1:-1]))

def palindrome(s):    #racecar==racecar
    return(s == s[::-1]    #::?
```

Binary Search:

- Recursion is the natural way to express certain important algorithmic concepts, such as binary search
- Imagine a list of elements known to be in sorted order. Write a function to find the location of a particular element, or return failure if the element is not found
- On average, this should take time in line with scanning half of the elements, since you are not exploiting prior knowledge of where the target might be.
- Problem is similar to searching for a name in a large phone book.

- Instead, exploiting the fact book entries are sorted: open to center, if target found there, return it,
- else if book has no entries, return failure,
- Else if target is less than midpoint entry, discard upper half of book
- Else if target is greater than midpoint entry, discard lower half of book

```
def helper(L,x,lo,hi):
    mid = (lo + hi) //2
    if (L[mid] == x):
        return(mid)
    elif (hi-lo <2):
        return(-1)
    elif (L[mid] < x):
        return(helper(L,x,mid,hi))
    else:
        return(helper(L,x,lo,mid))

def binsearch(L,x):
    return(helper(L,x,0,len(L)))
#we can rewrite this slightly to hide helper function, and share some of
the arguments
```

```
def binsearch(L,x):
    def helper(L,x,lo,hi):
        mid = (lo + hi) //2
        if (L[mid] == x):
             return(mid)
        elif (hi-lo <2):
             return(-1)
        elif (L[mid] < x):
             return(helper(L,x,mid,hi))
        else:
            return(helper(L,x,lo,mid))
    return(helper(L,x,0,len(L)))
```

- Worst case analysis is a sort of back of the envelope algebra of expected run time
- Worst case analysis relates the run time of the algorithm to the size of the input
- It ignores constant factors and instead only distinguishes between structural differences in run times
- For example, we would say sequential search takes linear time, or time proportional to the size(length), of the input L

- So, if on avg we examine half of the elements of L to find our match, as L gets longer so does the time required to examine half of the elements of L.
- Note that the ½ does not really factor into the rate of growth, it is just a constant
- We could use “order of” notation to indicate linear time as O(N)
- Binary search excludes ½ of remaining elements of L each step
- It continues to divide L in half until it finds target or isolates location where the target would otherwise be found
- Log is always base 2
- Question: how many times can I(integer) divide N by 2 before I gets to 1
- Answer: $\log_2(N)$

● Seqsearch

- Which is faster?