Mortgages Lab

July 21, 2018

In [3]:
```
# A program which examines the costs of three kinds of loans
# Assignment Alla Topp, p. 130-134 of the textbook

# Mortgage base class
def findPayment(loan, r, m):
    """Assumes: loan and r are floats, m an int
       Returns the monthly payment for a mortgage of size
       loan at a monthly rate of r for m months"""
    return loan*((r*(1+r)**m)/((1+r)**m - 1))

class Mortgage(object):
    """Abstract class for building different kinds of mortgages"""
    def __init__(self, loan, annRate, months):
        """Create a new mortgage"""
        self.loan = loan
        self.rate = annRate/12.0
        self.months = months
        self.paid = [0.0]
        self.owed = [loan]
        self.payment = findPayment(loan, self.rate, months)
        self.legend = None #description of mortgage
    def makePayment(self):
        """Make a payment"""
        self.paid.append(self.payment)
        reduction = self.payment - self.owed[-1]*self.rate
        self.owed.append(self.owed[-1] - reduction)
    def getTotalPaid(self):
        """Return the total amount paid so far"""
        return sum(self.paid)
    def __str__(self):
        return self.legend

# Mortgage subclasses
class Fixed(Mortgage):
    def __init__(self, loan, r, months):
        Mortgage.__init__(self, loan, r, months)
        self.legend = 'Fixed, ' + str(r*100) + '%'
```

```
        class FixedWithPts(Mortgage):
            def __init__(self, loan, r, months, pts):
                Mortgage.__init__(self, loan, r, months)
                self.pts = pts
                self.paid = [loan*(pts/100.0)]
                self.legend = 'Fixed, ' + str(r*100) + '%, '\
                              + str(pts) + ' points'

        class TwoRate(Mortgage):
            def __init__(self, loan, r, months, teaserRate, teaserMonths):
                Mortgage.__init__(self, loan, teaserRate, months)
                self.teaserMonths = teaserMonths
                self.teaserRate = teaserRate
                self.nextRate = r/12.0
                self.legend = str(teaserRate*100)\
                              + '% for ' + str(self.teaserMonths)\
                              + ' months, then ' + str(r*100) + '%'
            def makePayment(self):
                if len(self.paid) == self.teaserMonths + 1:
                    self.rate = self.nextRate
                    self.payment = findPayment(self.owed[-1], self.rate,
                                               self.months - self.teaserMonths)
                Mortgage.makePayment(self)

        # Evaluate mortgage
        def compareMortgages(amt, years, fixedRate, pts, ptsRate,
                             varRate1, varRate2, varMonths):
            totMonths = years*12
            fixed1 = Fixed(amt, fixedRate, totMonths)
            fixed2 = FixedWithPts(amt, ptsRate, totMonths, pts)
            twoRate = TwoRate(amt, varRate2, totMonths, varRate1, varMonths)
            morts = [fixed1, fixed2, twoRate]
            for m in range(totMonths):
                for mort in morts:
                    mort.makePayment()
            for m in morts:
                print (m)
                print (' Total payments = $' + str(int(m.getTotalPaid())))

        compareMortgages(amt=200000, years=30, fixedRate=0.07,
                         pts = 3.25, ptsRate=0.05, varRate1=0.045,
                         varRate2=0.095, varMonths=48)
```

```
Fixed, 7.000000000000001%
 Total payments = $479017
Fixed, 5.0%, 3.25 points
 Total payments = $393011
```

```
4.5% for 48 months, then 9.5%
 Total payments = $551444
```

In [ ]:
```
# Results:
# We can see that a fixed rate of 7% mortgage with no points is $479017
# A fixed rate of 5% mortgage with points is $393011
# A mortgage with an initial teaser rate of 4.5% followed by a higher rate
# of 9.5% for the duration would be $551444
```

Recursive binary search

July 21, 2018

In [1]:
```
# p.157 of the textbook, Alla Topp
# Finger exercise: Why does the code use mid+1
# rather than mid in the second recursive call
```

In [16]:
```
def search(L,e):
    """Assumes L is a list, the elements of which are in
       ascending order.
       Returns True if e is in L and False otherwise"""

    def bSearch(L, e, low, high):
        #Decrements high-low
        if high == low:
            return L[low] == e
        mid = (low+high)//2
        if L[mid] == e:
            return True
        elif L[mid]>e:
            if low == mid: #nothing left to search
                return False
            else:
                return bSearch(L, e, low, mid-1)
        else:
            return bSearch(L, e, mid+1, high)

    if len(L) == 0:
        return False
    else:
        return bSearch(L, e, 0, len(L) - 1)
```

In [22]:
```
L = [1, 3, 11, 8, 0, 100, 28, 88, 7]
print(sorted(L))
```

```
[0, 1, 3, 7, 8, 11, 28, 88, 100]
```

In [23]:
```
print(L)
```

```
[1, 3, 11, 8, 0, 100, 28, 88, 7]
```

```
In [ ]: # We can see that bSearch function has two recursive calls
        # first call uses arguments which covers all the elements on the left of mid (mid -1)
        # and the second call covers all elements on the right (mid +1)
        # mid is just a center
```