# sortedSets.py

# Firstly, recall our sets represented by (unsorted) lists.

set1 = [5, 3, 7, 0, 6, 2, 4]
set2 = [8, 0, 9, 5, 15, 10]

def intersect(s1,s2):
    if s1 == []:
        return []
    elif s1[0] in s2:
        return [s1[0]] + intersect(s1[1:],s2) # use it
    else:
        return intersect(s1[1:],s2) # lose it

def union(s1,s2):
    if s1 == []:
        return s2
    elif s1[0] in s2:
        return intersect(s1[1:],s2) # lose it
    else:
        return [s1[0]] + intersect(s1[1:],s2) # use it

def difference(s1,s2):
    """In s1 but not in s2."
    if s1 == []:
        return []
    elif s1[0] in s2:
        return difference(s1[1:],s2) # lose it
    else:
        return [s1[0]] + difference(s1[1:],s2) # use it

def member(e, s):
    """Our own version of in."
    if s == []:
        return False
    elif e == s[0]:
        return True
    else:
        return member(e,s[1:])

    # Equivalently

def member(e, s):
    """Our own version of in."
    """
    return s != [] and (e == s[0] or member(e,s[1:]))

# What are the costs of these operations,
# relative to the size of s, s1 and s2?

# Even if we implement these imperatively, the costs
# are roughly the same.

def intersect(s1,s2):
    result = []
    for e in s1:
        if e in s2:
            result.append(e)
    return result
```python
def union(s1, s2):
    result = s2
    for e in s1:
        if not e in s2:
            result.append(e)
    return result

def difference(s1, s2):
    """In s1 but not in s2."
    result = []
    for e in s1:
        if not e in s2:
            result.append(e)
    return result

def member(e, s):
    """Our own version of in."
    for i in s:
        if i == e:
            return True
    return False  # (if e is never found in s)

# Now, let's take advantage of sorted sets and consider the cost there.

sset1 = [0, 2, 3, 4, 5, 6, 7]
sset2 = [0, 5, 8, 9, 10, 15]

def intersectSorted(s1, s2):
    result = []
    i, j = 0, 0
    while i < len(s1) and j < len(s2):
        if s1[i] == s2[j]:
            result.append(s1[i])
            i += 1
            j += 1
        elif s1[i] < s2[j]:
            i += 1
        else:
            j += 1
    return result

# What's the cost of intersectSorted() in terms of the sizes of s1 and s2?
# How does that compare to the cost of intersect()?

# Let's do member() for sorted sets.

def memberSorted(e, s):
    for i in range(len(s)):
        if s[i] == e:
            return True
        elif s[i] > e:  # It can't be there (it would be out of order).
            return False
    return False

# Here's a better one. We keep dividing out search space in half.

def memberSorted(e, s):
```

# Keep dividing the search space in half.
low = 0
high = len(s)
while low < high:
    midpoint = (low + high) / 2
    if s[midpoint] == e:
        return True # found it!
    elif s[midpoint] < e:
        low = midpoint + 1 # look in upper half
    else:
        high = midpoint # look in lower half
return False # never found

## >>> memberSorted(0,sset1)
## True
## >>> memberSorted(43,sset1)
## False
## >>> memberSorted(5,sset1)
## True
## >>> memberSorted(5,[])
## False
## >>> memberSorted(-5,sset1)
## False
## >>>

# Now, let's measure the times for these and compare. Do these # measurements bear out our analyses?

import random
import time
import copy

def randomSet(size):
    bigSet = []
    for i in range(size):
        randomInt = random.randint(0,size*10)
        while randomInt in bigSet:
            randomInt = random.randint(0,size*10) # get another
        bigSet.append(randomInt)
    return bigSet

def timeIt(funcName, arg1, arg2):
    start = time.time()
    eval(funcName)(arg1, arg2)
    end = time.time()
    print funcName +"(arg1,arg2) =", end - start

big = 10
print "\nSet size =", big
bigset1 = randomSet(big)
bigset2 = randomSet(big)
timeIt("intersect", bigset1, bigset2)
bigset1Sorted = copy.copy(bigset1)
bigset1Sorted.sort()
bigset2Sorted = copy.copy(bigset2)
bigset2Sorted.sort()
timeIt("intersectSorted", bigset1Sorted, bigset2Sorted)

big = 1000
print "\nSet size =", big
bigset1 = randomSet(big)
bigset2 = randomSet(big)
timeIt("intersect", bigset1, bigset2)

bigset1Sorted = copy.copy(bigset1)
bigset1Sorted.sort()
bigset2Sorted = copy.copy(bigset2)
bigset2Sorted.sort()
timeIt("intersectSorted", bigset1Sorted, bigset2Sorted)

big = 100000
print "\nSet size =", big
bigset1 = randomSet(big)
bigset2 = randomSet(big)
timeIt("intersect", bigset1, bigset2)

bigset1Sorted = copy.copy(bigset1)
bigset1Sorted.sort()
bigset2Sorted = copy.copy(bigset2)
bigset2Sorted.sort()
timeIt("intersectSorted", bigset1Sorted, bigset2Sorted)

## >>> ============================== RESTART ==============================
## >>>
## ## Set size = 10
## ## intersect(arg1,arg2) = 3.2901763916e-05
## ## intersectSorted(arg1,arg2) = 3.09944152832e-05
## ##
## ## Set size = 1000
## ## intersect(arg1,arg2) = 0.0149459838867
## ## intersectSorted(arg1,arg2) = 0.000499963760376
## ##
## ## Set size = 100000
## ## intersect(arg1,arg2) = 150.630785942
## ## intersectSorted(arg1,arg2) = 0.0547950267792
## ## >>>