Searching and Sorting

Setup

- Set up `Timer` class to time executions

```
import time

class TimerError(Exception):
    """A custom exception used to report errors in use of Timer class"""

class Timer:
    def __init__(self):
        self._start_time = None
        self._elapsed_time = None

    def start(self):
        """Start a new timer"""
        if self._start_time is not None:
            raise TimerError("Timer is running. Use .stop()")
        self._start_time = time.perf_counter()

    def stop(self):
        """Save the elapsed time and re-initialize timer"""
        if self._start_time is None:
           raise TimerError("Timer is not running. Use .start()")
        self._elapsed_time = time.perf_counter() - self._start_time
        self._start_time = None

    def elapsed(self):
        """Report elapsed time"""
        if self._elapsed_time is None:
           raise TimerError("Timer has not been run yet. Use .start()")
        return(self._elapsed_time)

    def __str__(self):
        """print() prints elapsed time"""
        return(str(self._elapsed_time))
```

Naive search by scanning the list

```
def naivesearch(v,l):
  for x in l:
    if v == x:
      return(True)
  return(False)
```

Binary search

```
def binarysearch(v,l):
  if l == []:
    return(False)

  m = len(l)//2

  if v == l[m]:
    return(True)

  if v < l[m]:
    return(binarysearch(v,l[:m]))
  else:
    return(binarysearch(v,l[m+1:]))
```

Checking correctness on input `[0,2,...,50]`

```
l = list(range(0,51,2))

for i in range(51):
  print((i,naivesearch(i,l)),end=",")
```

```
print()

for i in range(51):
  print((i,binarysearch(i,l)),end=",")
print()
```

```
(0, True),(1, False),(2, True),(3, False),(4, True),(5, False),(6, True),(7, False),(8, True),(9, False),(10, True),(11, False),(12, T
(0, True),(1, False),(2, True),(3, False),(4, True),(5, False),(6, True),(7, False),(8, True),(9, False),(10, True),(11, False),(12, T
```

Performance comparison across 10^4 worst case searches in a list of size 10^5

- Looking for odd numbers in a list of even numbers

```
l = list(range(0,100000,2))
t = Timer()
t.start()
for i in range(3001,13000,2):
  v = naivesearch(i,l)
t.stop()
print()
print("Naive search", t)
t.start()
for i in range(3001,13000,2):
  v = binarysearch(i,l)
t.stop()
print()
print("Binary search", t)
```

```
Naive search 9.183804485999985

Binary search 1.3049918499999933
```

Selection sort

```
def SelectionSort(L):
   n = len(L)
   if n < 1:
      return(L)
   for i in range(n):
      # Assume L[:i] is sorted
      mpos = i
      # mpos is position of minimum in L[i:]
      for j in range(i+1,n):
        if L[j] < L[mpos]:
           mpos = j
      # L[mpos] is the smallest value in L[i:]
      (L[i],L[mpos]) = (L[mpos],L[i])
      # Now L[:i+1] is sorted
   return(L)
```

Selection sort performance is more or less the same for all inputs

```
import random
random.seed(2021)
inputlists = {}
inputlists["random"] = [random.randrange(100000) for i in range(5000)]
inputlists["ascending"] = [i for i in range(5000)]
inputlists["descending"] = [i for i in range (4999,-1,-1)]
t = Timer()
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    t.start()
    SelectionSort(tmplist)
    t.stop()
    print(k,t)
```

```
random 1.0969957959999874
ascending 1.1075799240000492
descending 1.174061538999979
```

Insertion sort, iterative

```
def InsertionSort(L):
   n = len(L)
   if n < 1:
      return(L)
   for i in range(n):
      # Assume L[:i] is sorted
      # Move L[i] to correct position in L[:i]
      j = i
      while(j > 0 and L[j] < L[j-1]):
        (L[j],L[j-1]) = (L[j-1],L[j])
        j = j-1
      # Now L[:i+1] is sorted
   return(L)
```

Insertion sort preformance

- On already sorted input, performance is very good
- On reverse sorted input, performance is worse than selection sort

```
import random
random.seed(2021)
inputlists = {}
inputlists["random"] = [random.randrange(100000) for i in range(5000)]
inputlists["ascending"] = [i for i in range(5000)]
inputlists["descending"] = [i for i in range (4999,-1,-1)]
t = Timer()
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    t.start()
    InsertionSort(tmplist)
    t.stop()
    print(k,t)
```

```
random 2.206825952000031
ascending 0.0010471530000017992
descending 4.247259635999967
```

Insertion sort, recursive

```
def Insert(L,v):
   n = len(L)
   if n == 0:
     return([v])
   if v >= L[-1]:
     return(L+[v])
   else:
     return(Insert(L[:-1],v)+L[-1:])

def ISort(L):
   n = len(L)
   if n < 1:
      return(L)
   L = Insert(ISort(L[:-1]),L[-1])
   return(L)
```

```
import random
random.seed(2021)
inputlists = {}
inputlists["random"] = [random.randrange(100000) for i in range(5000)]
inputlists["ascending"] = [i for i in range(5000)]
inputlists["descending"] = [i for i in range (4999,-1,-1)]
t = Timer()
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    t.start()
    ISort(tmplist)
    t.stop()
    print(k,t)
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-11-01b9ac69e2fd> in <module>()
      9     tmplist = inputlists[k][:]
     10     t.start()
---> 11     ISort(tmplist)
     12     t.stop()
     13     print(k,t)

[ 1 frames ]
... last 1 frames repeated, from the frame below ...

<ipython-input-10-c3c8a390eb84> in ISort(L)
     12     if n < 1:
     13         return(L)
```

Setup

- Set recursion limit to maxint, $2^{31} - 1$
  - This is the highest value Python allows

```
import sys
sys.setrecursionlimit(2**31-1)
```

Recursive insertion sort is slower than iterative

- Input of 2000 (40%) takes more time than 5000 for iterative
  - Overhead of recursive calls
- Performance pattern between unsorted, sorted and random is similar

```
import random
random.seed(2021)

inputlists = {}
inputlists["random"] = [random.randrange(100000) for i in range(2000)]
inputlists["ascending"] = [i for i in range(2000)]
inputlists["descending"] = [i for i in range (1999,-1,-1)]
t = Timer()
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    t.start()
    ISort(tmplist)
    t.stop()
    print(k,t)
```

```
random 12.900060098999973
ascending 0.028535271000009743
descending 20.618901792999964
```

Merge sort

```
def merge(A,B):
  (m,n) = (len(A),len(B))
  (C,i,j,k) = ([],0,0,0)
  while k < m+n:
    if i == m:
      C.extend(B[j:])
      k = k + (n-j)
    elif j == n:
      C.extend(A[i:])
      k = k + (n-i)
    elif A[i] < B[j]:
      C.append(A[i])
      (i,k) = (i+1,k+1)
    else:
      C.append(B[j])
      (j,k) = (j+1,k+1)
  return(C)
```

```
def mergesort(A):
  n = len(A)

  if n <= 1:
     return(A)

  L = mergesort(A[:n//2])
```

```
    R = mergesort(A[n//2:])

    B = merge(L,R)

    return(B)
```

- A simple input to check correctness

```
mergesort([i for i in range(0,1000,2)]+[j for j in range (1,1000,2)])
```

```
941,
942,
943,
944,
945,
946,
947,
948,
949,
950,
951,
952,
953,
954,
955,
956,
957,
958,
959,
960,
961,
962,
963,
964,
965,
966,
967,
968,
969,
970,
971,
972,
973,
974,
975,
976,
977,
978,
979,
980,
981,
982,
983,
984,
985,
986,
987,
988,
989,
990,
991,
992,
993,
994,
995,
996,
997,
998,
999]
```

- Perfomance on large inputs, 10^6, random and sorted

```
import random
random.seed(2021)
inputlists = {}
inputlists["random"] = [random.randrange(100000000) for i in range(1000000)]
inputlists["ascending"] = [i for i in range(1000000)]
inputlists["descending"] = [i for i in range (999999,-1,-1)]
t = Timer()
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    t.start()
    mergesort(tmplist)
    t.stop()
    print(k,t)
```

```
random 10.106079193000028
ascending 5.113605829999983
descending 5.000102768000033
```

```
def quicksort(L,l,r):  # Sort L[l:r]
  if (r - l <= 1):
    return
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper+1
    else:  # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      # Shift both segments
      (lower,upper) = (lower+1,upper+1)
  # Move pivot between lower and upper
  (L[l],L[lower-1]) = (L[lower-1],L[l])
  lower = lower-1
  # Recursive calls
  quicksort(L,l,lower)
··quicksort(L,lower+1,upper)
  return(L)
```

```
qlist = [1,3,5,0,2,4,17,2,-5,6,4,3]
qnew = quicksort(qlist,0,12)
print(qnew,qlist)
```

```
[-5, 0, 1, 2, 2, 3, 3, 4, 4, 5, 6, 17] [-5, 0, 1, 2, 2, 3, 3, 4, 4, 5, 6, 17]
```

```
import random
random.seed(2021)
inputlists = {}
inputlists["random"] = [random.randrange(100000000) for i in range(1000000)]
inputlists["ascending"] = [i for i in range(10000)]
inputlists["descending"] = [i for i in range (9999,-1,-1)]
t = Timer()
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    t.start()
    quicksort(tmplist,0,len(tmplist))
    t.stop()
    print(k,t)
```

```
random 5.946581722000019
ascending 6.36157859299999
descending 11.775665258999993
```