PDSP2024, Lecture 20, 29 October 2024

Searching and Sorting

Setup

- Use `time` library to time executions

In [1]:
```
import time
```

Naive search by scanning the list

In [2]:
```
def naivesearch(v,l):
  for x in l:
    if v == x:
      return(True)
  return(False)
```

Binary search

In [3]:
```
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]`

In [4]:
```
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, Tr
ue),(11, False),(12, True),(13, False),(14, True),(15, False),(16, True),(17, False),(18, True),(19, False),(20,
True),(21, False),(22, True),(23, False),(24, True),(25, False),(26, True),(27, False),(28, True),(29, False),(3
0, True),(31, False),(32, True),(33, False),(34, True),(35, False),(36, True),(37, False),(38, True),(39, Fals
e),(40, True),(41, False),(42, True),(43, False),(44, True),(45, False),(46, True),(47, False),(48, True),(49, F
alse),(50, True),
(0, True),(1, False),(2, True),(3, False),(4, True),(5, False),(6, True),(7, False),(8, True),(9, False),(10, Tr
ue),(11, False),(12, True),(13, False),(14, True),(15, False),(16, True),(17, False),(18, True),(19, False),(20,
True),(21, False),(22, True),(23, False),(24, True),(25, False),(26, True),(27, False),(28, True),(29, False),(3
0, True),(31, False),(32, True),(33, False),(34, True),(35, False),(36, True),(37, False),(38, True),(39, Fals
e),(40, True),(41, False),(42, True),(43, False),(44, True),(45, False),(46, True),(47, False),(48, True),(49, F
alse),(50, True),
```

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

In [5]:
```
l = list(range(0,200000,2))

starttime = time.perf_counter()
for i in range(3001,23000,2):
  v = naivesearch(i,l)
elapsed = time.perf_counter() - starttime
print()
print("Naive search", elapsed)

starttime = time.perf_counter()
for i in range(3001,23000,2):
  v = binarysearch(i,l)
```

```
elapsed = time.perf_counter() - starttime
print()
print("Binary search", elapsed)
```

```
Naive search 15.395366703996842

Binary search 2.4174999160022708
```

Selection sort

In [6]:
```
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

In [7]:
```
import random
random.seed(2024)
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)]
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    SelectionSort(tmplist)
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
random 0.42442894099804107
ascending 0.42093820399895776
descending 0.43981997100490844
```

Insertion sort, iterative

In [8]:
```
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

In [9]:
```
import random
random.seed(2024)
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)]
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    InsertionSort(tmplist)
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
random 0.643531706999056
ascending 0.00026845099637284875
descending 1.2660512360016583
```

Insertion sort, recursive

In [10]:
```
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)
```

In [11]:
```
import random
random.seed(2024)
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)]
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    ISort(tmplist)
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
Cell In[11], line 10
      8 tmplist = inputlists[k][:]
      9 starttime = time.perf_counter()
---> 10 ISort(tmplist)
     11 elapsed = time.perf_counter() - starttime
     12 print(k,elapsed)

Cell In[10], line 14, in ISort(L)
     12 if n < 1:
     13    return(L)
---> 14 L = Insert(ISort(L[:-1]),L[-1])
     15 return(L)

Cell In[10], line 14, in ISort(L)
     12 if n < 1:
     13    return(L)
---> 14 L = Insert(ISort(L[:-1]),L[-1])
     15 return(L)

    [... skipping similar frames: ISort at line 14 (2970 times)]

Cell In[10], line 14, in ISort(L)
     12 if n < 1:
     13    return(L)
---> 14 L = Insert(ISort(L[:-1]),L[-1])
     15 return(L)

RecursionError: maximum recursion depth exceeded
```

Setup

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

In [12]:
```
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

In [13]:
```
import random
random.seed(2024)

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)]
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    ISort(tmplist)
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
random 5.569181976999971
ascending 0.014761371996428352
descending 9.967413070000475
```

Merge sort

In [14]:
```
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)
```

In [15]:
```
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

In [16]:
```
mergesort([i for i in range(0,1000,2)]+[j for j in range (1,1000,2)])
```

Out[16]: [0,
1,
2,
3,
4,
5,
6,
7,
8,
9,
10,
11,
12,
13,
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,

85,
86,
87,
88,
89,
90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,
124,
125,
126,
127,
128,
129,
130,
131,
132,
133,
134,
135,
136,
137,
138,
139,
140,
141,
142,
143,
144,
145,
146,
147,
148,
149,
150,
151,
152,
153,
154,
155,
156,
157,
158,
159,
160,
161,
162,
163,
164,
165,
166,
167,
168,
169,
170,

171,
172,
173,
174,
175,
176,
177,
178,
179,
180,
181,
182,
183,
184,
185,
186,
187,
188,
189,
190,
191,
192,
193,
194,
195,
196,
197,
198,
199,
200,
201,
202,
203,
204,
205,
206,
207,
208,
209,
210,
211,
212,
213,
214,
215,
216,
217,
218,
219,
220,
221,
222,
223,
224,
225,
226,
227,
228,
229,
230,
231,
232,
233,
234,
235,
236,
237,
238,
239,
240,
241,
242,
243,
244,
245,
246,
247,
248,
249,
250,
251,
252,
253,
254,
255,
256,

257,
258,
259,
260,
261,
262,
263,
264,
265,
266,
267,
268,
269,
270,
271,
272,
273,
274,
275,
276,
277,
278,
279,
280,
281,
282,
283,
284,
285,
286,
287,
288,
289,
290,
291,
292,
293,
294,
295,
296,
297,
298,
299,
300,
301,
302,
303,
304,
305,
306,
307,
308,
309,
310,
311,
312,
313,
314,
315,
316,
317,
318,
319,
320,
321,
322,
323,
324,
325,
326,
327,
328,
329,
330,
331,
332,
333,
334,
335,
336,
337,
338,
339,
340,
341,
342,

343,
344,
345,
346,
347,
348,
349,
350,
351,
352,
353,
354,
355,
356,
357,
358,
359,
360,
361,
362,
363,
364,
365,
366,
367,
368,
369,
370,
371,
372,
373,
374,
375,
376,
377,
378,
379,
380,
381,
382,
383,
384,
385,
386,
387,
388,
389,
390,
391,
392,
393,
394,
395,
396,
397,
398,
399,
400,
401,
402,
403,
404,
405,
406,
407,
408,
409,
410,
411,
412,
413,
414,
415,
416,
417,
418,
419,
420,
421,
422,
423,
424,
425,
426,
427,
428,

429,
430,
431,
432,
433,
434,
435,
436,
437,
438,
439,
440,
441,
442,
443,
444,
445,
446,
447,
448,
449,
450,
451,
452,
453,
454,
455,
456,
457,
458,
459,
460,
461,
462,
463,
464,
465,
466,
467,
468,
469,
470,
471,
472,
473,
474,
475,
476,
477,
478,
479,
480,
481,
482,
483,
484,
485,
486,
487,
488,
489,
490,
491,
492,
493,
494,
495,
496,
497,
498,
499,
500,
501,
502,
503,
504,
505,
506,
507,
508,
509,
510,
511,
512,
513,
514,

515,
516,
517,
518,
519,
520,
521,
522,
523,
524,
525,
526,
527,
528,
529,
530,
531,
532,
533,
534,
535,
536,
537,
538,
539,
540,
541,
542,
543,
544,
545,
546,
547,
548,
549,
550,
551,
552,
553,
554,
555,
556,
557,
558,
559,
560,
561,
562,
563,
564,
565,
566,
567,
568,
569,
570,
571,
572,
573,
574,
575,
576,
577,
578,
579,
580,
581,
582,
583,
584,
585,
586,
587,
588,
589,
590,
591,
592,
593,
594,
595,
596,
597,
598,
599,
600,

601,
602,
603,
604,
605,
606,
607,
608,
609,
610,
611,
612,
613,
614,
615,
616,
617,
618,
619,
620,
621,
622,
623,
624,
625,
626,
627,
628,
629,
630,
631,
632,
633,
634,
635,
636,
637,
638,
639,
640,
641,
642,
643,
644,
645,
646,
647,
648,
649,
650,
651,
652,
653,
654,
655,
656,
657,
658,
659,
660,
661,
662,
663,
664,
665,
666,
667,
668,
669,
670,
671,
672,
673,
674,
675,
676,
677,
678,
679,
680,
681,
682,
683,
684,
685,
686,

687,
688,
689,
690,
691,
692,
693,
694,
695,
696,
697,
698,
699,
700,
701,
702,
703,
704,
705,
706,
707,
708,
709,
710,
711,
712,
713,
714,
715,
716,
717,
718,
719,
720,
721,
722,
723,
724,
725,
726,
727,
728,
729,
730,
731,
732,
733,
734,
735,
736,
737,
738,
739,
740,
741,
742,
743,
744,
745,
746,
747,
748,
749,
750,
751,
752,
753,
754,
755,
756,
757,
758,
759,
760,
761,
762,
763,
764,
765,
766,
767,
768,
769,
770,
771,
772,

773,
774,
775,
776,
777,
778,
779,
780,
781,
782,
783,
784,
785,
786,
787,
788,
789,
790,
791,
792,
793,
794,
795,
796,
797,
798,
799,
800,
801,
802,
803,
804,
805,
806,
807,
808,
809,
810,
811,
812,
813,
814,
815,
816,
817,
818,
819,
820,
821,
822,
823,
824,
825,
826,
827,
828,
829,
830,
831,
832,
833,
834,
835,
836,
837,
838,
839,
840,
841,
842,
843,
844,
845,
846,
847,
848,
849,
850,
851,
852,
853,
854,
855,
856,
857,
858,

859,
860,
861,
862,
863,
864,
865,
866,
867,
868,
869,
870,
871,
872,
873,
874,
875,
876,
877,
878,
879,
880,
881,
882,
883,
884,
885,
886,
887,
888,
889,
890,
891,
892,
893,
894,
895,
896,
897,
898,
899,
900,
901,
902,
903,
904,
905,
906,
907,
908,
909,
910,
911,
912,
913,
914,
915,
916,
917,
918,
919,
920,
921,
922,
923,
924,
925,
926,
927,
928,
929,
930,
931,
932,
933,
934,
935,
936,
937,
938,
939,
940,
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

In [17]:
```
import random
random.seed(2023)
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)]
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    mergesort(tmplist)
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
random 3.2394319730010466
ascending 1.6956631510038278
descending 1.6907014559983509
```

Quicksort

In [18]:
```
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
```

Small input to check correctness

In [19]:
```
qlist = [1,3,5,0,2,4,17,2,-5,6,4,3]
quicksort(qlist,4,8)
print(qlist)
```

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

Quicksort performance

- Random input of size 10^6
- Sorted inputs of size 2×10^4

In [20]:
```
import random
random.seed(2024)
inputlists = {}
inputlists["random"] = [random.randrange(100000000) for i in range(1000000)]
inputlists["ascending"] = [i for i in range(15000)]
inputlists["descending"] = [i for i in range (14999,-1,-1)]
for k in inputlists.keys():
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    quicksort(tmplist,0,len(tmplist))
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
random 2.5654859520000173
ascending 5.371246255002916
descending 8.383171867004421
```

Randomized quicksort

- Choose the pivot position uniformly at random between `l` and `r-1`
- Swap pivot to `L[l]` and proceed as usual

In [21]:
```
import random
def quicksortrand(L,l,r):  # Sort L[l:r]
  if (r - l <= 1):
    return(L)

  # Choose a random postion between l and r-l as pivot, swap with L[l]
  randpivot = random.randrange(r-l)
  (L[l],L[l+randpivot]) = (L[l+randpivot],L[l])

  # Rest is same as before
  (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
  quicksortrand(L,l,lower)
  quicksortrand(L,lower+1,upper)
  return(L)
```

In [22]:
```
import random
random.seed(2024)
inputlists = {}
inputlists["random"] = [random.randrange(100000000) for i in range(1000000)]
inputlists["ascending"] = [i for i in range(15000)]
inputlists["descending"] = [i for i in range (14999,-1,-1)]
inputlists["ascendingbig"] = [i for i in range(1000000)]
inputlists["descendingbig"] = [i for i in range (999999,-1,-1)]
for k in inputlists.keys():
```

```
    tmplist = inputlists[k][:]
    starttime = time.perf_counter()
    quicksortrand(tmplist,0,len(tmplist))
    elapsed = time.perf_counter() - starttime
    print(k,elapsed)
```

```
random 2.9289210839997395
ascending 0.03626769800030161
descending 0.02561152499401942
ascendingbig 2.2861892690052628
descendingbig 2.3142642469974817
```