Lecture 06 - Sorting

February 23, 2024

[2]:
```
def insertion_sort(list_to_sort): # O(n^2)
    L = list(list_to_sort)
    new_list = []

    while len(L) > 0:
        # find the index of the smallest thing
        min_index = min(range(len(L)), key=lambda i : L[i])

        # remove it from L and add it to new_list
        new_list.append(L.pop(min_index))

    return new_list
```

[3]:
```
import random
from time import time
import math
```

[7]:
```
L = [random.randint(1,1_000_000) for n in range(10_000)]
```

[8]:
```
tt = time()
R1 = insertion_sort(L)
print(time()-tt)
```

```
2.065708875656128
```

[9]:
```
tt = time()
R2 = sorted(L)
print(time()-tt)
```

```
0.0016009807586669922
```

[10]:
```
def merge_sort(L):
    # assume L is a list of numbers
    # output will be a sorted list of those numbers

    # base case: a list of length 1 is already sorted
    if len(L) <= 1:
        return L
```

```
    # split the list (roughly) in half
    mid_point = math.ceil(len(L)/2)
    left_half = L[:mid_point]
    right_half = L[mid_point:]

    # recursively apply the function to the two halfs (divide)
    LS = merge_sort(left_half)
    RS = merge_sort(right_half)

    # time to conquer
    full_list = []

    # while the two halves still have something left
    while len(LS) + len(RS) > 0:
        # either LS[0] or RS[0] is the smallest of all elements
        # in LS and RS. Find it, remove it from its list, and
        # add it to full_list
        #       "if [list]" means "if the list is non-empty".
        #       same as "if len([list]) > 0"
        if LS:
            if RS:
                if LS[0] <= RS[0]:
                    full_list.append(LS.pop(0))
                else:
                    full_list.append(RS.pop(0))
            else:
                full_list.append(LS.pop(0))
        else:
            full_list.append(RS.pop(0))
    return full_list
```

[11]:
```
tt = time()
R3 = merge_sort(L)
print(time()-tt)
```

```
0.03760099411010742
```

[12]:
```
R3 == R2
```

[12]:
```
True
```

[ ]:
```
times = []
for p in range(1,20):
    L = [random.randint(1,1000000) for n in range(2**p)]

    tt = time()
    R = insertion_sort(L)
```

```
    times.append(time()-tt)

    print(f"{2**p} {time()-tt}")
    if len(times) > 1:
        print(f"\t{times[-1]/times[-2]}")
```

```
2 0.0018496513366699219
4 1.0967254638671875e-05
        0.007599059164103492
8 1.0967254638671875e-05
        0.9761904761904762
16 2.7894973754882812e-05
        2.7560975609756095
32 7.009506225585938e-05
        2.566371681415929
64 0.00024700164794921875
        3.5586206896551724
128 0.0007770061492919922
        3.1540697674418605
256 0.004185199737548828
        5.388940092165899
512 0.013364315032958984
        3.194629724645117
1024 0.02763819694519043
        2.068543997715795
2048 0.0884389877319336
        3.200025881033516
4096 0.3295559883117676
        3.7263992494560982
8192 1.3519058227539062
        4.102263938818018
16384 5.438715219497681
        4.022998426362198
32768 21.997645139694214
        4.0446419130747975
65536 88.00860595703125
        4.000819673357127
131072 510.3064298629761
        5.798369826028475
```

[17]:
```
times = []
for p in range(0,7):
    L = [random.randint(1,1000000) for n in range(10**p)]

    tt = time()
    R2 = merge_sort(L)
    times.append(time()-tt)
```

```
    print(f"{10**p} {time()-tt}")
    if len(times) > 1:
        print(f"\t{times[-1]/times[-2]}")
```

```
1 0.011921167373657227
10 2.9802322387695312e-05
        0.0026050637271787806
100 0.00019311904907226562
        6.694214876033058
1000 0.0025758743286132812
        13.302469135802468
10000 0.026812076568603516
        10.435730858468677
100000 0.8233866691589355
        30.71299746542754
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[17], line 6
      3 L = [random.randint(1,1000000) for n in range(10**p)]
      5 tt = time()
----> 6 R2 = merge_sort(L)
      7 times.append(time()-tt)
      9 print(f"{10**p} {time()-tt}")

Cell In[10], line 16, in merge_sort(L)
     14 # recursively apply the function to the two halfs (divide)
     15 LS = merge_sort(left_half)
---> 16 RS = merge_sort(right_half)
     18 # time to conquer
     19 full_list = []

Cell In[10], line 15, in merge_sort(L)
     12 right_half = L[mid_point:]
     14 # recursively apply the function to the two halfs (divide)
---> 15 LS = merge_sort(left_half)
     16 RS = merge_sort(right_half)
     18 # time to conquer

Cell In[10], line 15, in merge_sort(L)
     12 right_half = L[mid_point:]
     14 # recursively apply the function to the two halfs (divide)
---> 15 LS = merge_sort(left_half)
     16 RS = merge_sort(right_half)
     18 # time to conquer
```

```
    [… skipping similar frames: merge_sort at line 15 (1 times)]

Cell In[10], line 15, in merge_sort(L)
     12 right_half = L[mid_point:]
     14 # recursively apply the function to the two halfs (divide)
---> 15 LS = merge_sort(left_half)
     16 RS = merge_sort(right_half)
     18 # time to conquer

Cell In[10], line 33, in merge_sort(L)
     31         full_list.append(LS.pop(0))
     32     else:
---> 33         full_list.append(RS.pop(0))
     34 else:
     35     full_list.append(LS.pop(0))

KeyboardInterrupt:
```

[18]:
```
times = []
for p in range(1,8):
    L = [random.randint(1,1000000) for n in range(10**p)]

    tt = time()
    R3 = sorted(L)
    times.append(time()-tt)

    print(f"{10**p} {time()-tt}")
    if len(times) > 1:
        print(f"\t{times[-1]/times[-2]}")
```

```
10 0.00019884109497070312
100 1.2636184692382812e-05
        0.060753341433778855
1000 0.00010323524475097656
        8.58
10000 0.0015521049499511719
        15.135198135198134
100000 0.014345884323120117
        9.26520868627753
1000000 0.1295619010925293
        9.032896158513273
10000000 1.8113889694213867
        13.980638928249387
```

[ ]: