Lecture 06 - Sorting

February 17, 2023

[1]:
```
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
```

[2]:
```
import random
from time import time
import math
```

[3]:
```
# O(n^2)
L = [random.randint(1,1_000_000) for n in range(10_000)]
```

[4]:
```
tt = time()
R1 = insertion_sort(L)
print(time()-tt)
```

```
3.492711305618286
```

[5]:
```
tt = time()
R2 = sorted(L)
print(time()-tt)
```

```
0.001461029052734375
```

[6]:
```
R1 == R2
```

[6]:
```
True
```

[8]:
```
def merge_sort(L):

    # 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 LS" means the same as "if len(LS) > 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
```

[9]:
```
tt = time()
R3 = merge_sort(L)
print(time()-tt)
```

```
0.04400300979614258
```

[10]:
```
R3 == R2
```

[10]:
```
True
```

[14]:
```
from tqdm import tqdm
from time import sleep
```

[15]:
```
for i in tqdm(range(1000)):
    y = i+1
    sleep(0.1)
```

```
 22%|
| 224/1000 [00:23<01:20,  9.59it/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[15], line 3
      1 for i in tqdm(range(1000)):
      2     y = i+1
----> 3     sleep(0.1)

KeyboardInterrupt:
```

[11]:
```
times = []
for p in range(1,6):
    L = [random.randint(1,1000000) for n in range(10**p)]

    tt = time()
    R = insertion_sort(L)
    times.append(time()-tt)

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

```
10 2.3126602172851562e-05
100 0.0005078315734863281
        22.817204301075268
1000 0.04228687286376953
        83.57587181903864
10000 3.9788308143615723
        94.09979249836479
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[11], line 6
      3 L = [random.randint(1,1000000) for n in range(10**p)]
      5 tt = time()
----> 6 R = insertion_sort(L)
      7 times.append(time()-tt)
      9 print(f"{10**p} {time()-tt}")

Cell In[1], line 7, in insertion_sort(list_to_sort)
```

```
      3 new_list = []
      5 while len(L) > 0:
      6     # find the index of the smallest thing
----> 7     min_index = min(range(len(L)), key=lambda i : L[i])
      9     # remove it from L and add it to new_list
     10     new_list.append(L.pop(min_index))

Cell In[1], line 7, in insertion_sort.<locals>.<lambda>(i)
      3 new_list = []
      5 while len(L) > 0:
      6     # find the index of the smallest thing
----> 7     min_index = min(range(len(L)), key=lambda i : L[i])
      9     # remove it from L and add it to new_list
     10     new_list.append(L.pop(min_index))

KeyboardInterrupt:
```

[17]:
```
times = []
for p in range(1,7):
    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]}")
```

```
10 0.0032930374145507812
100 0.00024199485778808594
        0.08152969894222946
1000 0.0026509761810302734
        11.083832335329342
10000 0.038099050521850586
        14.38699801908878
100000 1.2326390743255615
        32.35682367225345
```

```
---------------------------------------------------------------------------
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[8], line 29, in merge_sort(L)
     27 if RS:
     28     if LS[0] <= RS[0]:
---> 29         full_list.append(LS.pop(0))
     30     else:
     31         full_list.append(RS.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 5.91278076171875e-05
100 1.0013580322265625e-05
        0.15416666666666667
1000 0.00010204315185546875
        11.432432432432432
10000 0.0014238357543945312
        14.018912529550827
100000 0.018959760665893555
        13.406913996627319
1000000 0.3307461738586426
        17.44864973648793
10000000 4.558649063110352
        13.782788598780295
```

[ ]: