Lecture 06 - Sorting

February 14, 2025

[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
        # smallest_number = min(L)
        # smallest_index = L.index(smallest_number)

        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
```

[5]:
```
L = [random.randint(1,1_000_000) for n in range(10_000)]
```

[6]:
```
tt = time()
R1 = insertion_sort(L)
print(time()-tt)
```

```
1.9664602279663086
```

[7]:
```
tt = time()
R2 = sorted(L)
print(time()-tt)
```

```
0.0015079975128173828
```

[8]:
```
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:
            assert len(RS) > 0
            full_list.append(RS.pop(0))
    return full_list
```

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

```
0.02866673469543457
```

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

[10]: True

[11]:
```
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 6.9141387939453125e-06
4 3.814697265625e-06
        0.48
8 4.76837158203125e-06
        1.6666666666666667
16 1.1920928955078125e-05
        2.5
32 3.314018249511719e-05
        2.78
64 0.00011205673217773438
        3.381294964028777
128 0.0003907680511474609
        3.4872340425531916
256 0.001377105712890625
        3.5210494203782794
512 0.006947040557861328
        5.046958932594005
1024 0.02269291877746582
        3.2676302959555037
2048 0.08159685134887695
        3.5959148077711114
4096 0.3188202381134033
        3.9073148781243243
8192 1.33201003074646
        4.177965265147889
16384 5.31607723236084
        3.991025878304774
32768 21.62325406074524
        4.06752246500922
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[11], line 6
      3 L = [random.randint(1,1000000) for n in range(2**p)]
      5 tt = time()
----> 6 R = insertion_sort(L)
```

```
      7 times.append(time()-tt)
      9 print(f"{2**p} {time()-tt}")

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

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

KeyboardInterrupt:
```

[12]:
```
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 7.200241088867188e-05
10 1.7881393432617188e-05
        0.25597269624573377
100 0.0001499652862548828
        8.386666666666667
1000 0.001924753189086914
        12.81558028616852
10000 0.03138113021850586
        16.327130628954222
100000 0.9506680965423584
```

```
30.296270125291574
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[12], 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 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[8], line 31, in merge_sort(L)
     29 if RS:
     30     if LS[0] <= RS[0]:
---> 31         full_list.append(LS.pop(0))
     32     else:
     33         full_list.append(RS.pop(0))

KeyboardInterrupt:
```

[13]:
```
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 0.00015306472778320312
100 1.0013580322265625e-05
        0.058084772370486655
1000 0.00011086463928222656
        12.324324324324325
10000 0.001292286987304688
        11.859649122807017
100000 0.012674093246459961
```

```
        9.828032544378699
1000000 0.13305974006652832
        10.500188146754468
10000000 1.9103949069976807
        14.356965336821451
```

[ ]: