sorting

February 11, 2026

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

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

```
2.3219053745269775
```

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

```
0.0008757114410400391
```

[6]:
```
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
```

[7]:
```
tt = time()
R3 = merge_sort(L)
print(time()-tt)
```

```
0.023035049438476562
```

[8]:
```
R3 == R2
```

[8]:
```
True
```

[9]:
```
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 5.245208740234375e-06
4 3.0994415283203125e-06
        0.5294117647058824
8 3.814697265625e-06
        1.7777777777777777
16 1.0967254638671875e-05
        2.625
32 3.075599670410156e-05
        3.0714285714285716
64 9.989738464355469e-05
        3.248062015503876
128 0.000370025634765625
        3.704057279236277
256 0.0014729499816894531
        3.9780927835051547
512 0.006085872650146484
        4.133138969873664
1024 0.030744075775146484
        5.052786268516342
2048 0.09838628768920898
        3.2003924397186223
4096 0.38050079345703125
        3.867499981824747
8192 1.5638058185577393
        4.109896486039401
16384 6.210069894790649
        3.9711307981959627
32768 24.940457344055176
        4.016132701196906
65536 100.17861104011536
        4.016711595898216
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[9], 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 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:
```

[10]:
```
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 3.314018249511719e-05
10 1.4066696166992188e-05
        0.43703703703703706
100 0.00010395050048828125
        7.389830508474576
1000 0.0014007091522216797
        13.474770642201834
10000 0.024042129516601562
        17.162212765957445
100000 0.9299261569976807
        38.68346094338874
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[10], 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[6], 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:
```

[11]:
```
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 0.0002009868621826172
100 6.9141387939453125e-06
        0.029940119760479042
1000 6.604194641113281e-05
        10.92
10000 0.0007967948913574219
        12.227106227106226
100000 0.010386943817138672
        13.047633313361294
1000000 0.13788390159606934
        13.278534199710696
10000000 1.866286039352417
        13.535305713100014
```

[ ]: