Searching_And_Sorting

March 11, 2020

In [1]:
```
# tools
import time
import random
import matplotlib.pyplot as plt
# generate random data
def make_random(n):
    return [random.randint(1,1000) for i in range(0,n)]
# generate ascending sequence of data
def ascending(n):
    return [i for i in range(0,n)]
# geenrate descending sequence of data
def descending(n):
    return [n-i for i in range(0,n)]
# check if data x are sorted
def check(x):
    n = len(x)
    for i in range(1,n):
        assert x[i-1] <= x[i]
# measure the time used to apply some sorting algorithm "algo" to data "x"
def measure_time(algo,x):
    start = time.time()
    algo(x)
    end = time.time();
    check(x) # sanity
    return end - start
# plot time of some sorting algorithms
def show_time(n,scenario,sorts):
    durations = []
    m = len(sorts)
    plt.figure(figsize=(10,6))
    for k in range(0,m):
        durations.append([measure_time(sorts[k][0],scenario(i)) for i in range(1,n)])
        plt.plot(durations[k], label=sorts[k][1])
    plt.xlabel("Array length")
    plt.ylabel("Runtime (s)")
    plt.legend(bbox_to_anchor=(0.85, 1), loc='lower right', borderaxespad=0.)
    plt.show()
```

In [2]:
```
# original sorting algorithm ("selection sort" from last week)
def original_selection_sort(x):
    n = len(x)
    for i in range(0,n):
        for j in range(i,n):
            if x[j] < x[i]:
                x[i],x[j] = x[j],x[i]
    return x
```

In [3]:
```
# selection sort algorithm, slightly improved
def selection_sort (x):
    n = len(x)
    for i in range(0,n):
        v = x[i]
        k = i
        for j in range(i+1,n):
            if x[j] < x[k]:
                k = j
        if k != i:
            x[k],x[i] = x[i],x[k]
    return x
```

In [4]:
```
# insertion sort algorithm from case study
def original_insertion_sort(x):
    n = len(x)
    for i in range(1, n):
        for j in reversed(range(0, i)):
            if x[j+1] < x[j]:
                x[j+1], x[j] = x[j], x[j+1]
            else:
                break
```

In [5]:
```
# insertion sort algorithm (slightly improved)
def insertion_sort(x):
    n = len(x)
    for i in range(0, n):
        j = i;
        v = x[i];
        while j > 0 and v < x[j-1]:
            x[j] = x[j-1]
            j = j-1
        x[j] = v
    return x
```

In [6]:
```
x = make_random(10)
print(x)
insertion_sort(x)
```

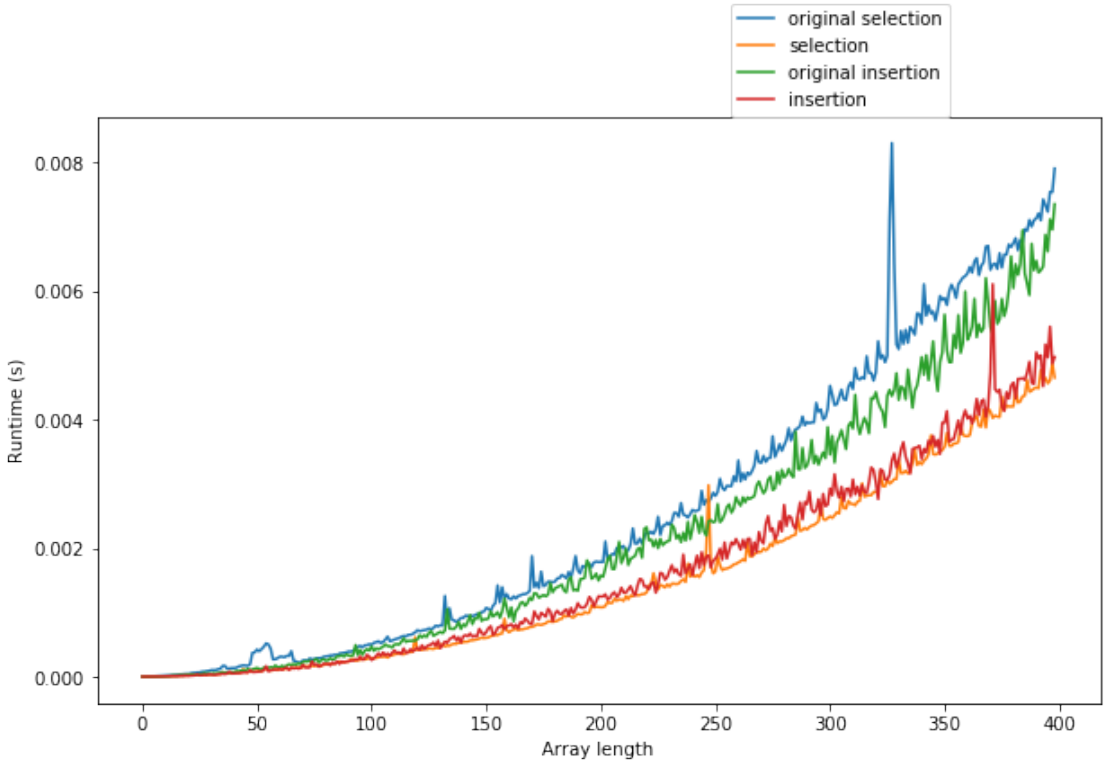
```
        check(x)
        print(x)
```

```
[864, 575, 159, 797, 850, 781, 655, 103, 685, 908]
[103, 159, 575, 655, 685, 781, 797, 850, 864, 908]
```

```
In [7]: n=4000
        print("random")
        print("original selection",measure_time(original_selection_sort,make_random(n)))
        print("selection",measure_time(selection_sort,make_random(n)))
        print("original insertion",measure_time(original_insertion_sort,make_random(n)))
        print("insertion",measure_time(insertion_sort,make_random(n)))
        print("ascending")
        print("original selection",measure_time(original_selection_sort,ascending(n)))
        print("selection",measure_time(selection_sort,ascending(n)))
        print("original insertion",measure_time(original_insertion_sort,ascending(n)))
        print("insertion",measure_time(insertion_sort,ascending(n)))
        print("descending")
        print("original selection",measure_time(original_selection_sort,descending(n)))
        print("selection",measure_time(selection_sort,descending(n)))
        print("original insertion",measure_time(original_insertion_sort,descending(n)))
        print("insertion",measure_time(insertion_sort,descending(n)))
```

```
random
original selection 0.7029609680175781
selection 0.5312433242797852
original insertion 0.8479001522064209
insertion 0.6985256671905518
ascending
original selection 0.5212631225585938
selection 0.5313122272491455
original insertion 0.0017833709716796875
insertion 0.0005598068237304688
descending
original selection 1.1382269859313965
selection 0.5119519233703613
original insertion 1.5960867404937744
insertion 1.3777172565460205
```

```
In [8]: show_time(400,make_random,[
            [original_selection_sort,"original selection"],
            [selection_sort,"selection"],
            [original_insertion_sort,"original insertion"],
            [insertion_sort,"insertion"]])
```




```
In [9]: def linear_search(x,b):
            n = len(x)
            m = n
            while m > 0 and b < x[m-1]:
                m = m - 1
            return m
        # right limit r exclusive
        def binary_search_r(x, l, r, b):
            if l==r: # not found
                return l
            assert(l<r)
            m = (l+r) // 2
            if b < x[m]:
                return binary_search_r(x,0,m,b)
            elif b > x[m]:
                return binary_search_r(x,m+1,r,b)
            else: # b == x[m] found
                return m
        def binary_search(x,b):
            return binary_search_r(x,0,len(x),b)

In [10]: print(binary_search([1,3,5],2))
```

1

```
In [11]: def time_search(x,algo):
             start = time.time()
             for i in range(0,10000):
                 b = algo(x,i)
             end = time.time()
             return end-start
```

```
In [12]: x = ascending(10000)
         print("linear",time_search(x,linear_search))
         print("binary",time_search(x,binary_search))
```

```
linear 4.819830656051636
binary 0.10728621482849121
```

```
In [13]: def insert_linear(x,b):
             p = linear_search(x,b)
             x.insert(p,b)
             return x
         def linear_insertion_sort (x):
             res = []
             n = len(x)
             for i in range(0,n):
                 insert_linear(res,x[i])
             for i in range(0,n):
                 x[i] = res[i]
         def insert_binary(x,b):
             p = binary_search(x,b)
             x.insert(p,b)
             return x
         def binary_insertion_sort (x):
             res = []
             n = len(x)
             for i in range(0,n):
                 insert_binary(res,x[i])
             for i in range(0,n):
                 x[i] = res[i]
```

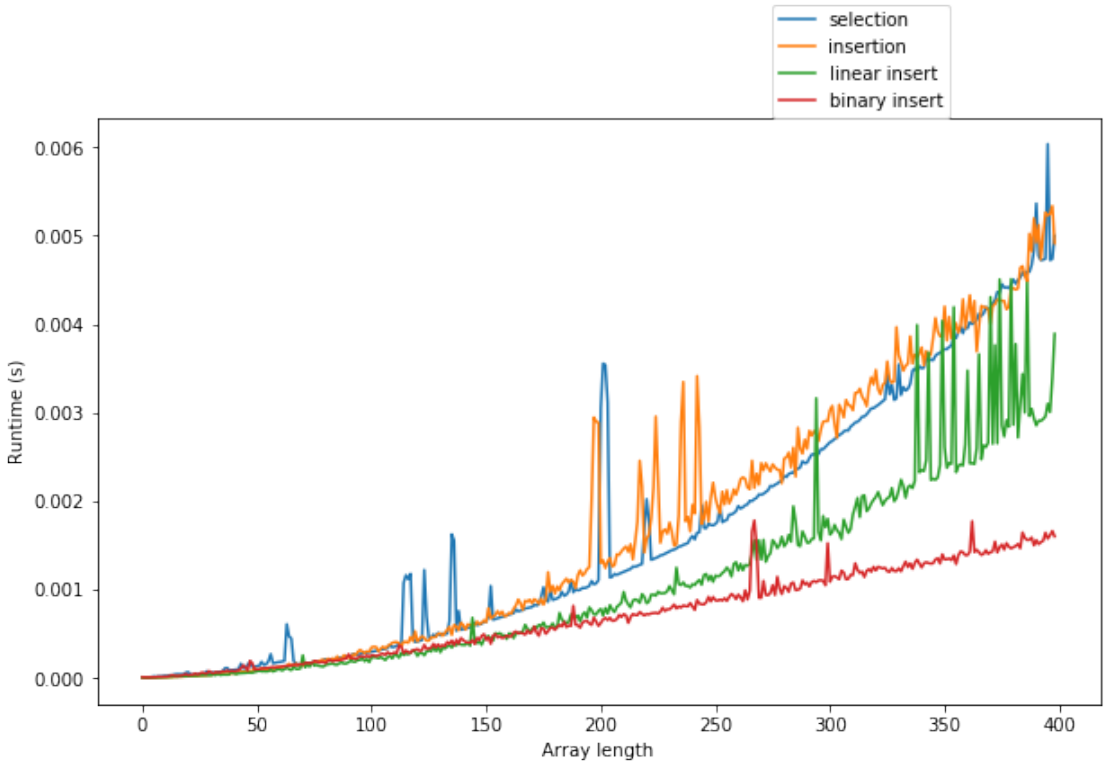
```
In [14]: print(insert_linear([1,3,5,11,17,23],333))
         print(insert_binary([1,3,5,11,17,23],333))
```

```
[1, 3, 5, 11, 17, 23, 333]
[1, 3, 5, 11, 17, 23, 333]
```

```
In [15]: data = [10,18,2,8,20]
         binary_insertion_sort(data)
         print(data)
```

```
[2, 8, 10, 18, 20]
```

```
In [16]: show_time(400,make_random,[
             [selection_sort,"selection"],
             [insertion_sort,"insertion"],
             [linear_insertion_sort,"linear insert"],
             [binary_insertion_sort,"binary insert"]
         ])
```




```
In [17]: n=8000
         print("random")
         print("binary_insertion",measure_time(binary_insertion_sort,make_random(n)))
         print("linear_insertion",measure_time(linear_insertion_sort,make_random(n)))
         print("ascending")
         print("binary_insertion",measure_time(binary_insertion_sort,ascending(n)))
         print("linear_insertion",measure_time(linear_insertion_sort,ascending(n)))
         print("descending")
         print("binary_insertion",measure_time(binary_insertion_sort,descending(n)))
         print("linear_insertion",measure_time(linear_insertion_sort,descending(n)))

random
binary_insertion 0.06321191787719727
```

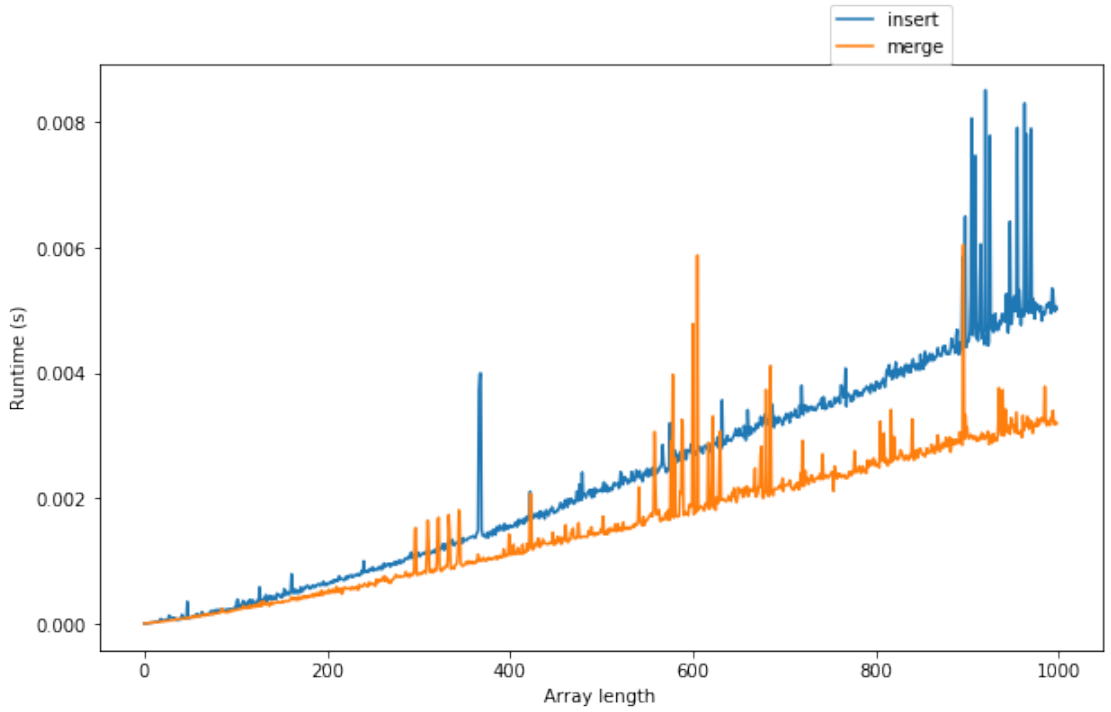
```
linear_insertion 1.679685115814209
ascending
binary_insertion 0.027215957641601562
linear_insertion 0.0034923553466796875
descending
binary_insertion 0.034285783767700195
linear_insertion 3.082965850830078
```

```
In [18]: def merge(x,l,m,r):
             result = []
             i = l
             j = m
             while i < m and j < r:
                 if x[i] < x[j]:
                     result.append(x[i])
                     i = i + 1
                 else:
                     result.append(x[j])
                     j = j + 1
             while i < m:
                 result.append(x[i])
                 i = i + 1
             while j < r:
                 result.append(x[j])
                 j = j + 1
             for i in range(l,r):
                 x[i] = result[i-l]

         # recursively sort data in x[l:r], (l included, r excluded!)
         def merge_sort_r(x,l,r):
             assert(0 <= l and l <= r)
             assert(r <= len(x))
             if r - l > 1: # more than one element
                 m = (l+r) // 2
                 merge_sort_r(x,l,m)
                 merge_sort_r(x,m,r)
                 merge(x,l,m,r)

         def merge_sort(x):
             merge_sort_r(x,0,len(x))
```

```
In [19]: x = make_random(80)
         merge_sort(x)
         check(x)
         print(x)
```

```
[26, 33, 44, 54, 61, 117, 119, 143, 165, 171, 219, 226, 233, 236, 237, 242, 256, 266, 267, 271
```

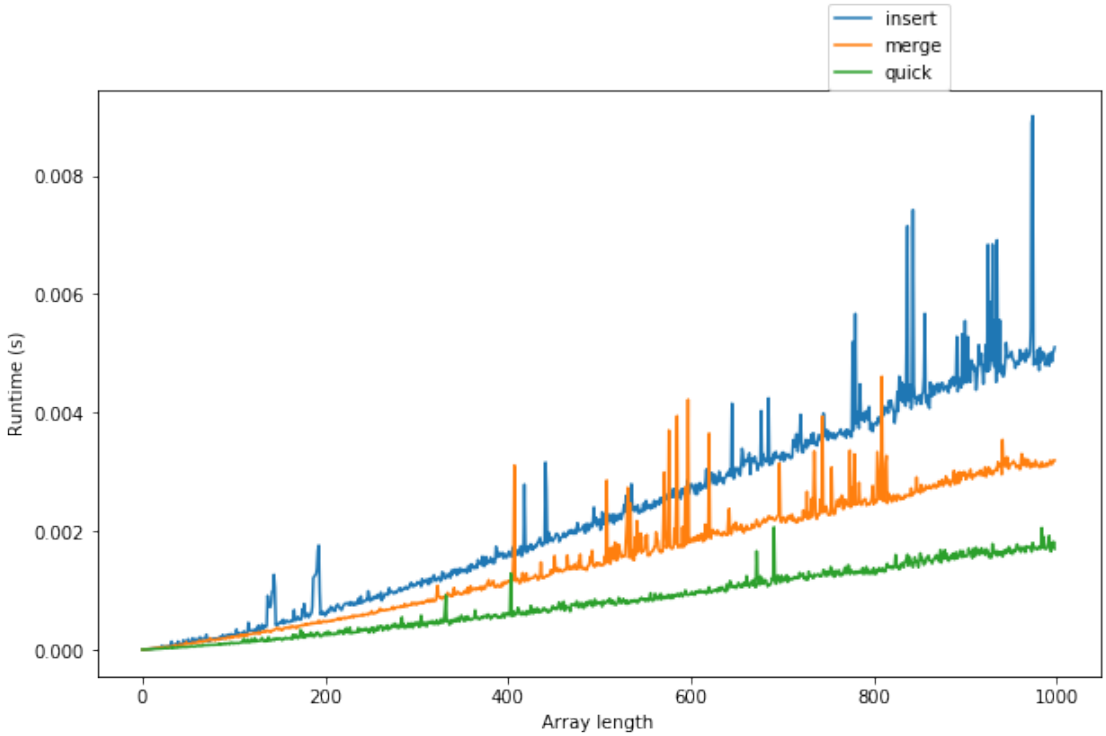
In [20]:
```
show_time(1000,make_random,[[binary_insertion_sort,"insert"],[merge_sort,"merge"]])
```



In [21]:
```
n=100000
print("random")
print("binary_insertion",measure_time(binary_insertion_sort,make_random(n)))
print("merge",measure_time(merge_sort,make_random(n)))
print("ascending")
print("binary_insertion",measure_time(binary_insertion_sort,ascending(n)))
print("merge",measure_time(merge_sort,ascending(n)))
print("descending")
print("binary_insertion",measure_time(binary_insertion_sort,descending(n)))
print("merge",measure_time(merge_sort,descending(n)))
```

```
random
binary_insertion 1.7975482940673828
merge 0.5363290309906006
ascending
binary_insertion 0.4638535976409912
merge 0.47109174728393555
descending
binary_insertion 2.719980478286743
merge 0.4664645195007324
```

```
In [22]: def partition(x,l,r,p):
             r = r-1
             while (l < r):
                 while x[l] < p:
                     l = l + 1
                 while x[r] > p:
                     r = r - 1
                 x[l],x[r] = x[r],x[l]
                 if x[l] == x[r]:
                     l = l + 1
             return l-1

In [23]: def quick_sort_r(x,l,r):
             if r-l > 1:
                 m = partition(x,l,r,x[(l+r)//2])
                 quick_sort_r(x,l,m)
                 quick_sort_r(x,m+1,r)

         def quick_sort(x):
             quick_sort_r(x,0,len(x))

In [24]: x = make_random(100)
         quick_sort(x)
         check(x)
         print(x)

[20, 50, 54, 57, 61, 62, 76, 81, 106, 117, 128, 132, 135, 138, 138, 155, 187, 189, 192, 206, 2

In [25]: show_time(1000,make_random,[[binary_insertion_sort,"insert"],[merge_sort,"merge"],[qu
```




```
In [26]: n=100000
         print("random")
         print("binary_insertion",measure_time(binary_insertion_sort,make_random(n)))
         print("merge",measure_time(merge_sort,make_random(n)))
         print("quick",measure_time(merge_sort,make_random(n)))
         print("ascending")
         print("binary_insertion",measure_time(binary_insertion_sort,ascending(n)))
         print("merge",measure_time(merge_sort,ascending(n)))
         print("quick",measure_time(merge_sort,make_random(n)))
         print("descending")
         print("binary_insertion",measure_time(binary_insertion_sort,descending(n)))
         print("merge",measure_time(merge_sort,make_random(n)))
         print("quick",measure_time(merge_sort,descending(n)))
```

```
random
binary_insertion 1.7695677280426025
merge 0.5400090217590332
quick 0.5045046806335449
ascending
binary_insertion 0.46944761276245117
merge 0.4671595096588135
quick 0.535001277923584
descending
```

```
binary_insertion 2.6789934635162354
merge 0.5006849765777588
quick 0.4471285343170166
```

```
In [27]: def natural_merge_sort(x):
             n = len(x)
             l = n
             while l != 0:
                 r = 0
                 while r < n:
                     # build intervals [l,m) and [m,r)
                     l = r # previous right border, valid element
                     m = l
                     while m < n-1 and x[m] <= x[m+1]:
                         m = m + 1
                     m = m + 1 # first element not in ascending sequence
                     if m < n:
                         r = m # valid element
                         while r < n-1 and x[r] <= x[r+1]:
                             r = r + 1
                         r = r + 1 # first element not in ascending sequence
                         merge(x,l,m,r)
                     else: # sweep done
                         r = n
```

```
In [28]: n=100000
         print("random")
         print("natural_merge",measure_time(natural_merge_sort,make_random(n)))
         print("merge",measure_time(merge_sort,make_random(n)))
         print("quick",measure_time(merge_sort,make_random(n)))
         print("ascending")
         print("natural_merge",measure_time(natural_merge_sort,ascending(n)))
         print("merge",measure_time(merge_sort,ascending(n)))
         print("quick",measure_time(merge_sort,make_random(n)))
         print("descending")
         print("natural_merge",measure_time(natural_merge_sort,descending(n)))
         print("merge",measure_time(merge_sort,make_random(n)))
         print("quick",measure_time(merge_sort,descending(n)))
```

```
random
natural_merge 0.6969892978668213
merge 0.5101075172424316
quick 0.5071032047271729
ascending
natural_merge 0.01383352279663086
merge 0.43948960304260254
quick 0.497849702835083
```

```
descending
natural_merge 0.6574206352233887
merge 0.5415761470794678
quick 0.45767951011657715
```