CSE 1384

Lab 6: Quicksort

James Stevens

Jhs435

Ramesh Shrestha

Rs2401

Mrs. JK

TA: Delma

October 8, 2017

Problem Statement:

Test each Pivot method in Quicksort and compare them to each other as well as Insertion Sort

Design and Analysis:

N is the length of the list

C is the time it takes to make one comparison

Best Case:

Each time the list is divided in two so it will run log2(n) times with n times so

F(n) = n log2(n)

O(nlog2(n))

Worst Case:

F(n) = (n^2)/2 – n/2

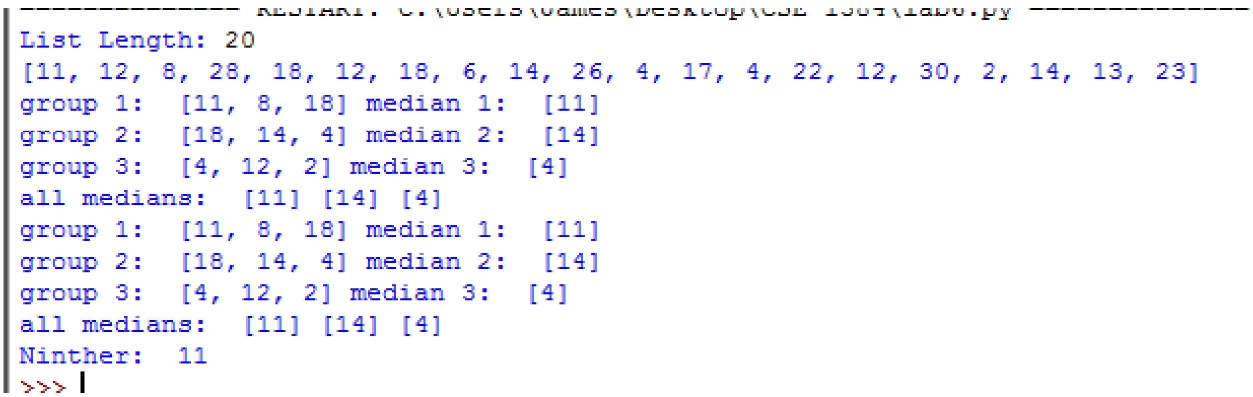
From ((n-1)(n-1+1))/2

Since it can only run n-1 times per number in the list

O(n^2)

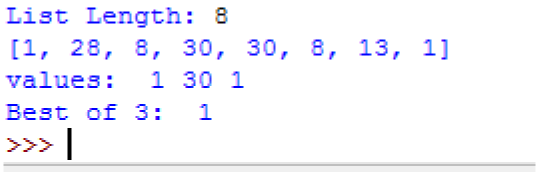
Screenshots:

Ninther Pivot:



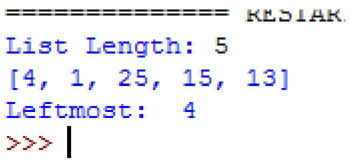
```
============== RESTART: C:\Users\James\Desktop\CSE 1384\lab6.py ==============
List Length: 20
[11, 12, 8, 28, 18, 12, 18, 6, 14, 26, 4, 17, 4, 22, 12, 30, 2, 14, 13, 23]
group 1:  [11, 8, 18] median 1:  [11]
group 2:  [18, 14, 4] median 2:  [14]
group 3:  [4, 12, 2] median 3:  [4]
all medians:  [11] [14] [4]
group 1:  [11, 8, 18] median 1:  [11]
group 2:  [18, 14, 4] median 2:  [14]
group 3:  [4, 12, 2] median 3:  [4]
all medians:  [11] [14] [4]
Ninther:  11
>>> 
```

Best of 3 Pivot:



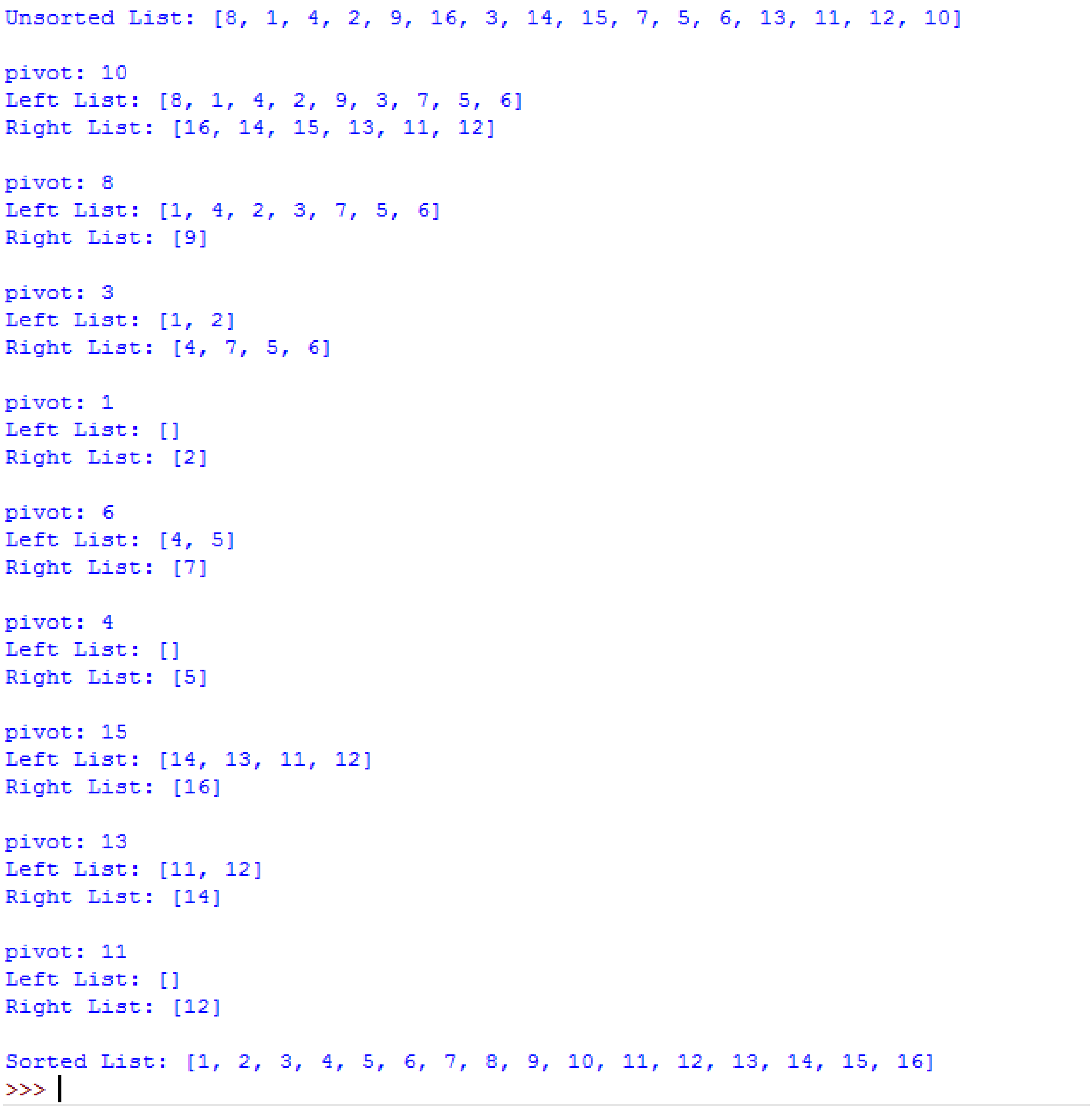
```
List Length: 8
[1, 28, 8, 30, 30, 8, 13, 1]
values:  1 30 1
Best of 3:  1
>>> 
```

Leftmost Pivot:



```
============== RESTAR
List Length: 5
[4, 1, 25, 15, 13]
Leftmost:  4
>>> 
```

Quicksort Test:



```
Unsorted List: [8, 1, 4, 2, 9, 16, 3, 14, 15, 7, 5, 6, 13, 11, 12, 10]

pivot: 10
Left List: [8, 1, 4, 2, 9, 3, 7, 5, 6]
Right List: [16, 14, 15, 13, 11, 12]

pivot: 8
Left List: [1, 4, 2, 3, 7, 5, 6]
Right List: [9]

pivot: 3
Left List: [1, 2]
Right List: [4, 7, 5, 6]

pivot: 1
Left List: []
Right List: [2]

pivot: 6
Left List: [4, 5]
Right List: [7]

pivot: 4
Left List: []
Right List: [5]

pivot: 15
Left List: [14, 13, 11, 12]
Right List: [16]

pivot: 13
Left List: [11, 12]
Right List: [14]

pivot: 11
Left List: []
Right List: [12]

Sorted List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
>>> 
```

Timing Tables:

Unsorted List

Leftmost Pivot selection Method

List Length n	Single sort Time in sec	Estimated C value in sec
5000	0.005565	2.52e-10
10000	0.0223	3.023 e-10
15000	0.0523	2.42 e-10
20000	0.0901	2.071e-10
25000	0.14118	2.07e-10

Calculated C: 2.2476e-10 sec

Best of 3 pivot selection method

List Length n	Single sort Time in Sec	Estimated C value in sec
5000	0.005144	2.51e-10
10000	0.0209	2.006e-10
15000	0.0478	2.06e-10
20000	0.08279	2.16e-10
25000	0.1306	1.996e-10

Calculated C: 2.0678e-10sec

Ninther pivot selection method

List Length n	Single sort Time in Sec	Estimated C value in sec
5000	0.00521	2.123e-10
10000	0.0210	2.523e-10
15000	0.0485	2.423e-10
20000	0.0855	2.1023e-10
25000	0.1350	1.893e-10

Calculated C: 2.123e-10sec

Insertion sort

List Length n	Single sort Time in Sec	Estimated C value in sec
5000	0.00176	7.995e-11
10000	0.00695	7.125e-11
15000	0.0157	7.025e-11
20000	0.0277	7.325e-11
25000	0.0451	6.925e-11

Calculated C:7.025e-11sec

SORTED LIST

Leftmost Pivot selection Method

List Length n	Single sort Time in sec	Estimated C value in sec
5000	0.00523	2.147e-10
10000	0.0205	3.356e-10
15000	0.0496	2.006e-10
20000	0.0864	2.006e-10
25000	0.134	1.8906e-10

Calculated C:2.46e-10sec

Best of 3 pivot selection method

List Length n	Single sort Time in Sec	Estimated C value in sec
5000	0.00508	2.08e-10
10000	0.0206	2.58e-10
15000	0.0464	2.36e-10
20000	0.0841	2.37e-10
25000	0.130	1.78e-10

Calculated C:2.08 e-10sec

Ninther pivot selection method

List Length n	Single sort Time in Sec	Estimated C value in sec
5000	0.0503	2.84e-10
10000	0.0201	2.83e-10
15000	0.0485	2.757e-10
20000	0.0831	2.064e-10
25000	0.128	1.962e-10

Calculated C: 2.064 e-10sec

Insertion sort

List Length n	Single sort Time in Sec	Estimated C value in sec
5000	0.00071	2.013e-11

10000	0.0029	2.112e-11
15000	0.00613	2.317e-11
20000	0.01096	2.912e-11
25000	0.0179	2.011e-11

Calculated C:2.812e-11secs

Graphs:

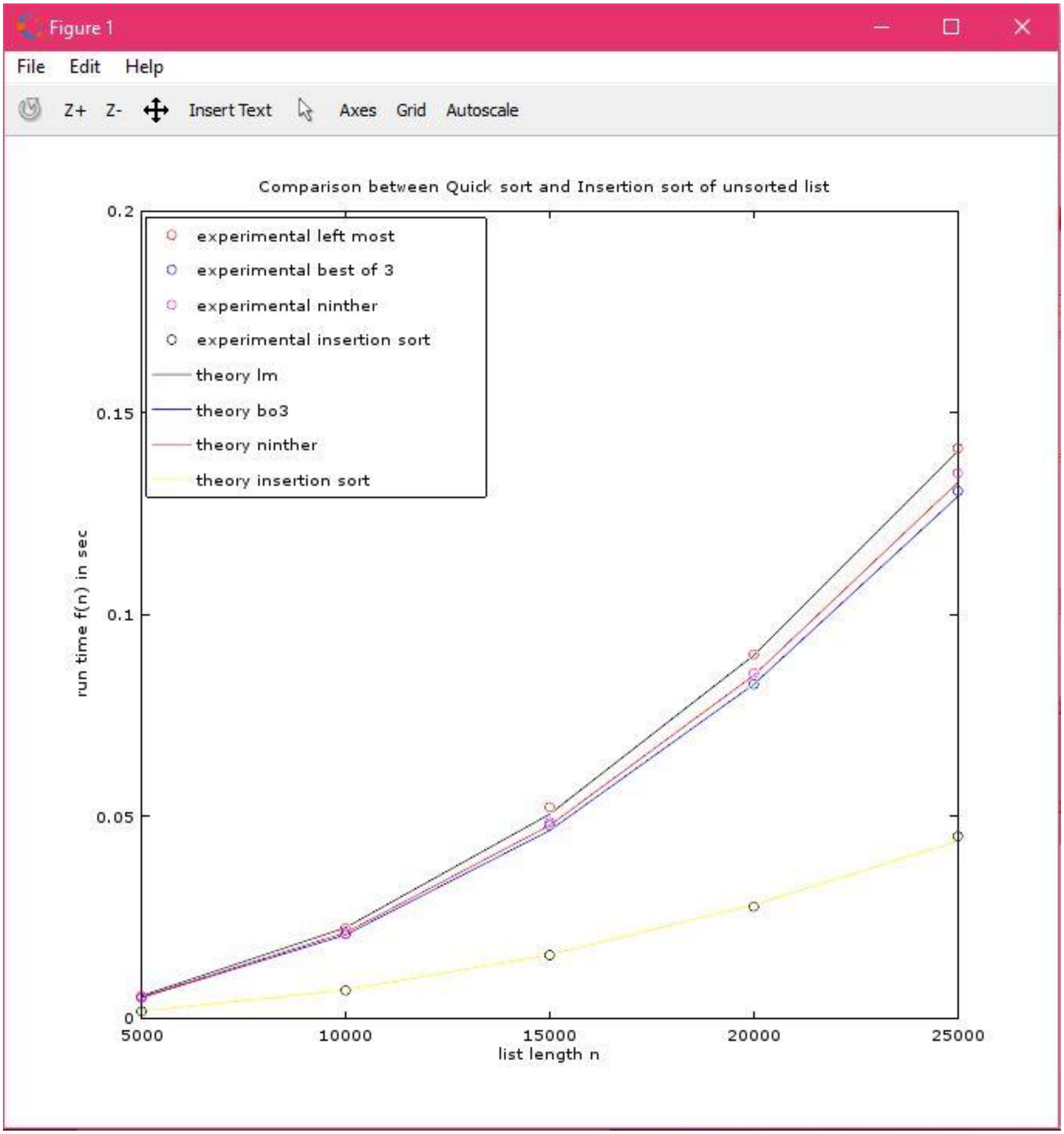


Figure 1
File
Edit
Help
Z+
Z-
Insert Text
Axes
Grid
Autoscale
Comparison between Quick sort and Insertion sort of unsorted list
experimental left most
experimental best of 3
experimental ninther
experimental insertion sort
theory lm
theory bo3
theory ninther
theory insertion sort
run time f(n) in sec
0.2
0.15
0.1
0.05
0
5000
10000
15000
20000
25000
list length n

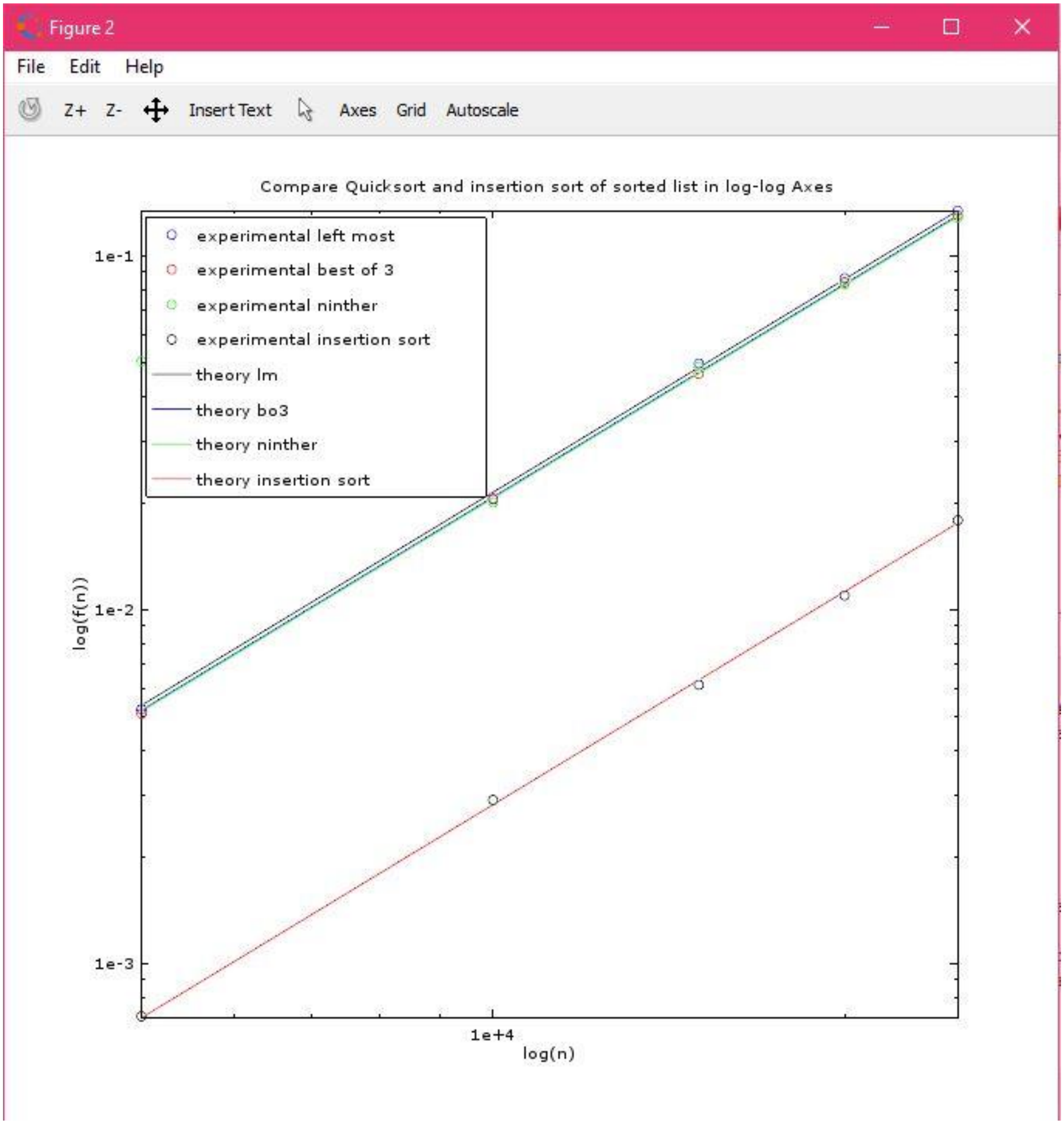


Figure 2
File
Edit
Help
Z+
Z-
Insert Text
Axes
Grid
Autoscale
Compare Quicksort and insertion sort of sorted list in log-log Axes
○ experimental left most
○ experimental best of 3
○ experimental ninther
○ experimental insertion sort
theory lm
theory bo3
theory ninther
theory insertion sort
1e-1
1e-2
1e-3
log(f(n))
1e+4
log(n)

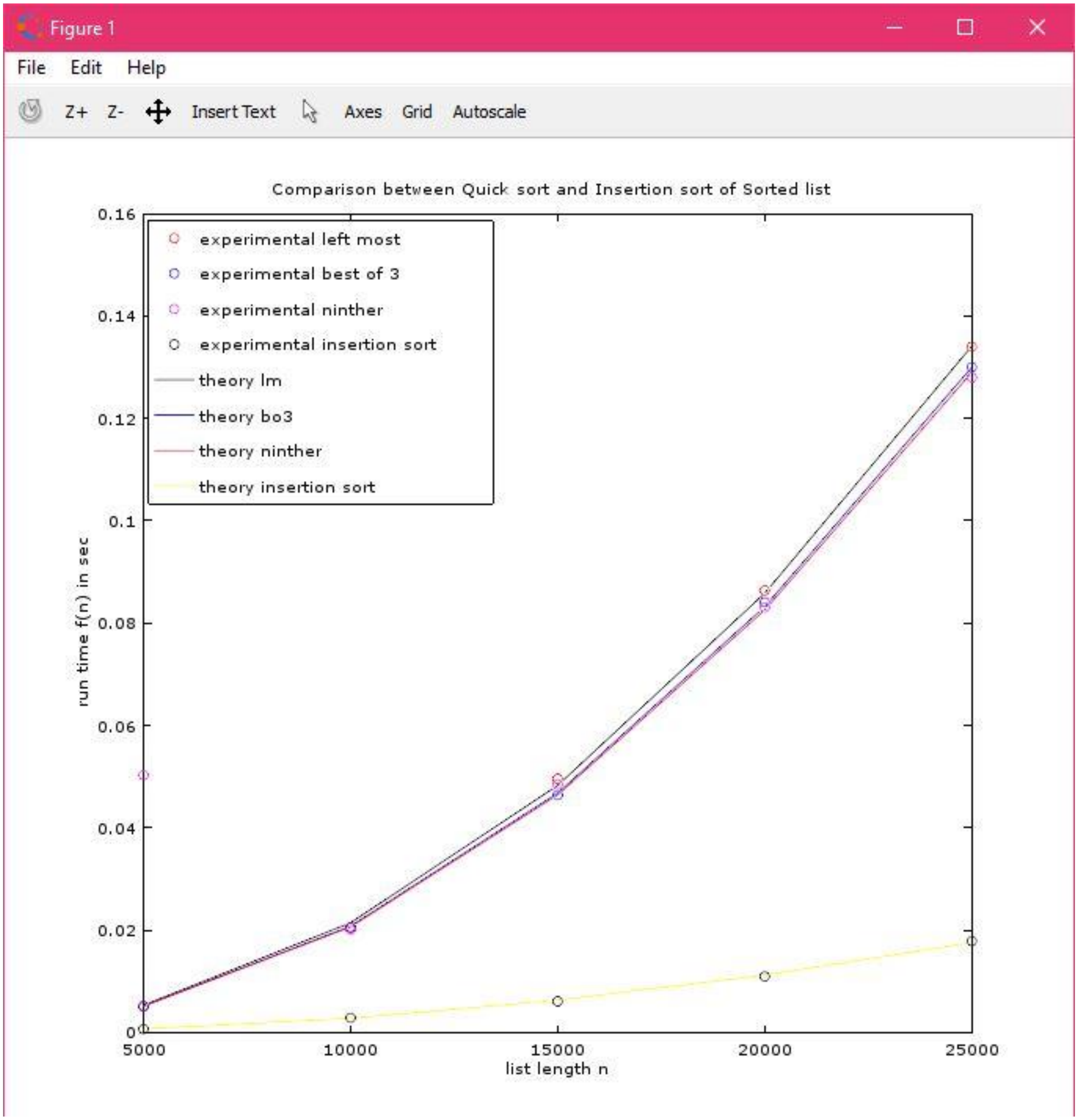


Figure 1
File
Edit
Help
Z+
Z-
Insert Text
Axes
Grid
Autoscale
Comparison between Quick sort and Insertion sort of Sorted list
experimental left most
experimental best of 3
experimental ninther
experimental insertion sort
theory lm
theory bo3
theory ninther
theory insertion sort
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
run time f(n) in sec
5000
10000
15000
20000
25000
list length n

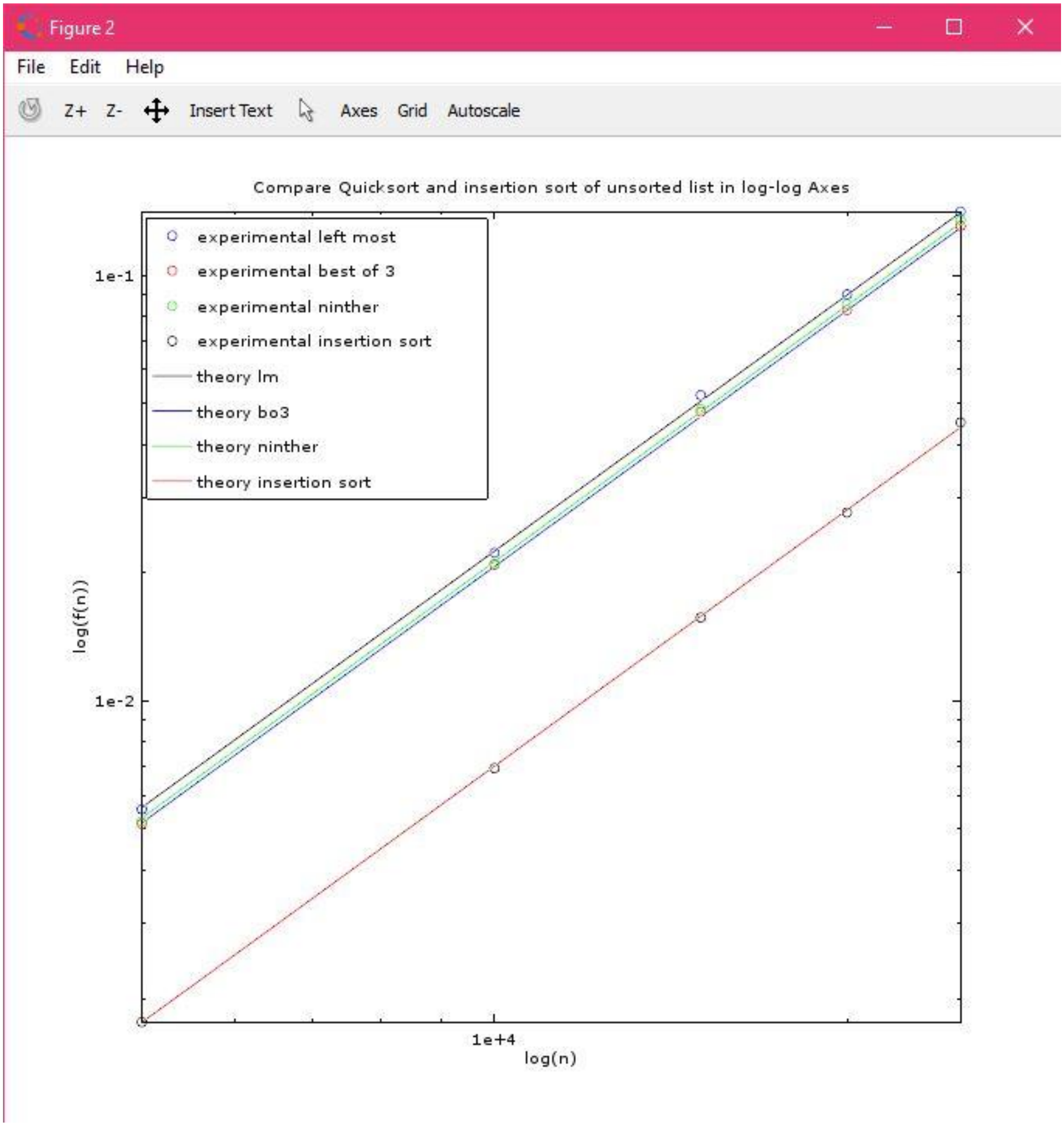


Figure 2
File
Edit
Help
Z+
Z-
Insert Text
Axes
Grid
Autoscale
Compare Quicksort and insertion sort of unsorted list in log-log Axes
experimental left most
experimental best of 3
experimental ninther
experimental insertion sort
theory lm
theory bo3
theory ninther
theory insertion sort
1e-1
1e-2
log(f(n))
1e+4
log(n)

Conclusions:

1) Quicksort because it has to take fewer steps at best to sort the list
2) Best of 3 because it doesn’t take as long as Ninther and can be used on a greater variety of lists
3) Ninther because it’s more likely to find the middle value and find the parts of the list that are sorted already

Code Appendix:

```
from random import randint

from time import time




# function: Pivot_Leftmost


# parameters: any list


# returns: the first value in the list


# purpose: find the first value of the list to use as a pivot


def Pivot_Leftmost(List):
   return List[0]


# -------------------------------------------------------------------------
# function: Pivot_Best_of_3


# parameters: any list with at least 3 values
# returns: a pivot using best of 3 method

# purpose: find a pivot using best of 3 method, which picks 3 evenly distributed
# values and finds the median

def Pivot_Best_of_3(List):

   # finds the 3 values to use
   n = len(List)- 1
   start = 0
   mid = n//2

   #print("values: ", List[start], List[mid], List[n])

   # since it is only 3 values, removes the highest and lowest
   myList=[List[start], List[mid], List[n]]
   myList.remove(max(myList))
   myList.remove(min(myList))

   # sets the remaining value to a variable
   pivot= myList[0]


   return pivot

# --------------------------------------------------------------------------
# function: Pivot_Ninther

# parameters: any list with at least 9 values
# returns: a pivot using the Ninther method

# purpose: find a pivot using Ninther method which picks the median
# of 3 sets of evenly distributed medians

def Pivot_Ninther(List):

   # creates a value to use to find the distance between each value to find the 9 numbers
   dist = len(List)//9

   # creates a group of the first 3 evenly distributed numbers
   group1 = [List[0], List[dist], List[dist*2]]

#    group1_start = [List[0], List[dist], List[dist*2]]

   # removes the top and bottom value of each set to find the median of the set

   group1.remove(max(group1))
   group1.remove(min(group1))

#    print("group 1: ", group1_start, "median 1: ", group1)

   group2 = [List[dist*3], List[dist*4], List[dist*5]]

#    group2_start = [List[dist*3], List[dist*4], List[dist*5]]

   group2.remove(max(group2))
   group2.remove(min(group2))
```

```
#    print("group 2: ", group2_start, "median 2: ", group2)

    group3 = [List[dist*6], List[dist*7], List[dist*8]]

##    group3_start= [List[dist*6], List[dist*7], List[dist*8]]

    group3.remove(max(group3))
    group3.remove(min(group3))

##    print("group 3: ", group3_start, "median 3: ", group3)
##    print("all medians: ", group1, group2, group3)

    # creates a list of the medians of the 3 groups
    medianmed = group1 + group2 + group3

    medianmed.remove(max(medianmed))
    medianmed.remove(min(medianmed))

    # returns the remaining value which is the median of the medians
    return medianmed[0]

# -------------------------------------------------------------------

def quick_sort_bo3(List):

    # if the list is less than 2 values
    if len(List) <2:
```

```
    # return it as is
    return List


# finds a pivot using best of 3 method
pivot = Pivot_Best_of_3(List)

# print("pivot:",pivot)

# makes 3 empty lists for the left and right bins and the pivots
leftList=[]
rightList=[]
pivotList=[]

# for each value in the list, assigns it to a bin using its size compared to the pivot
for val in List:

    if val < pivot:

        leftList.append(val)

    elif val== pivot:

        pivotList.append(val)


    else:
        rightList.append(val)
    #print("Left List:",leftList)
    #print("Right List:",rightList)
    #print("")

    # recursively runs using the left and right bins
    q= quick_sort_bo3(leftList)

    w= quick_sort_bo3(rightList)

    # returns a sorted list or sorted bin
    return q+ pivotList+w

# -------------------------------------------------------------------------

# parameters: an unsorted List

# returns: a sorted List or List segment to previous recursions

# purpose: sort a list recursively by dividing the list into bins and
# recombining while finding pivots using the leftmost item in the list

def quick_sort_leftmost(List):

    # if the list is one or fewer values, return as is
    if len(List) <2:

        return List
```

```
# finds the pivot using leftmost value in list
pivot = Pivot_Leftmost(List)

#print("pivot",pivot)

# creates 3 empty lists for the bins and pivots
leftList=[]
rightList=[]
pivotList=[]

# moves values into left or right bins based on whether the value is larger
# or smaller than the pivot or if it is the same as the pivot
for val in List:

    if val < pivot:

        leftList.append(val)

    elif val== pivot:

        pivotList.append(val)



    else:
        rightList.append(val)
# print("Left list",leftList)
# print("Right List",rightList)

# uses the best of the method for other recursions
```

```
    q= quick_sort_bo3(leftList)

    w= quick_sort_bo3(rightList)

   # print("left bin: ", q, "right bin:", w)

    # returns sorted list
    return q+ pivotList+w

# ----------------------------------------------------------------------------

# parameters: an unsorted list ; a pivot found using Ninther method

# returns: a sorted list

# purpose: sorts a list recursively and finds pivots using Ninther method

def quick_sort_ninther(List, Pivot):

    # if the list has 1 or fewer values
    if len(List) <2:

        # return the list as is
        return List

    # creates 3 empty lists to hold bins and pivots
    leftList=[]
    rightList=[]
    pivotList=[]
```

```
 # for each
 for val in List:

     # if value is smaller than the pivot
     if val < Pivot:

         # put it into the left bin
         leftList.append(val)

     # if the value is the same as the pivot
     elif val== Pivot:

         # move it into the pivot list
         pivotList.append(val)



     else:

         # if the value is larger than the pivot, put it in the right bin
         rightList.append(val)

 # recursively uses the function on the left and right bins
 q= quick_sort_bo3(leftList)

 w= quick_sort_bo3(rightList)

# print("left bin: ", q, "right bin:", w)
```

```
    # returns the sorted list
    return q+ pivotList+w

# ---------------------------------------------------------------------------

# parameters: an unsorted list

# returns: a sorted list

# purpose: sorts by checking every value in an unsorted list and comparing
#          it to the last value in a sorted list
def insertion_sort(List):


    n = len(List)-1
    sortedList = []
    # repeats for as long as the list is
    for i in range(1, n):

        # picks the value to compare
        value = List[i]

        # compares the value with every number currently in the list
        for j in range(len(sortedList)):

            #if the value is lower than a value, inserts it before the current list value
            if value < sortedList[j]:

                sortedList.insert(j, value)
                break

            # if it's greater than any value in the list, places it at the end
            elif value >= sortedList[len(sortedList)-1]:

                sortedList.append(value)

                break

    # returns the sorted list
    return sortedList


# ---------------------------------------------------------------------------------------------------



n= int(input("enter list size: "))

# creates a random list of specified length using randint function
randList=[]
for i in range(1, n+1):
    randList.append(randint(1,100000))
print("List Length", len(randList))


#creates a sorted list of specified length
sortedList = []
for i in range(1, n+1):
    sortedList.append(i)

# timing code sets

start_time1 = time()
for rep1 in range(10000):
    leftmost=quick_sort_leftmost(randList)
stop_time1 = time()

total_time1 = stop_time1 - start_time1
C = total_time1/(10000*n**2)

print("Unsorted List")
print("\nLeftmost Quicksort")
print("single time:",total_time1/10000)
print("est C:", C)
print("\n")

start_time2 = time()
for rep1 in range(10000):
    bo3=quick_sort_bo3(randList)
stop_time2 = time()

total_time2 = stop_time2 - start_time2
C = total_time2/(10000*n**2)

print("Best of 3 Quicksort")
print("single time:",total_time2/10000)
print("est C:", C)
print("\n")


start_time3 = time()

nintherPivot = Pivot_Ninther(randList)

for rep1 in range(10000):


ninthersort=quick_sort_ninther(randList, nintherPivot)

stop_time3 = time()

total_time3 = stop_time3 - start_time3

C = total_time3/(10000*n**2)


print("Ninther Quicksort")

print("single time:",total_time3/10000)

print("est C:", C)

print("\n")


start_time4 = time()

for rep1 in range(10000):

insort= insertion_sort(randList)

stop_time4 = time()

total_time4 = stop_time4 - start_time4

C = total_time4/(10000*n**2)


print("Insertion Sort")

print("single time:",total_time4/10000)

print("est C:", C)

print("\n\n")


# -------------------------------------------------------
# sorted list
start_time_1 = time()

for rep1 in range(10000):
    sortedlist=quick_sort_leftmost(sortedList)

stop_time_1 = time()
total_time_1 = stop_time_1 - start_time_1
C = total_time_1/(10000*n**2)

print("SORTED LIST")
print("\nLeftmost Quicksort")
print("single time:",total_time_1/10000)
print("est C:", C)
print("\n")

start_time_2 = time()

for rep1 in range(10000):
    bo3sorted=quick_sort_bo3(sortedList)

stop_time_2 = time()
total_time_2 = stop_time_2 - start_time_2
C = total_time_2/(10000*n**2)

print("Best of 3 Quicksort")
print("single time:",total_time_2/10000)
print("est C:", C)
print("\n")
```

```
start_time_3 = time()
nintherPivot = Pivot_Ninther(sortedList)
for rep1 in range(10000):
    quick_sort_ninther(sortedList, nintherPivot)

stop_time_3 = time()
total_time_3 = stop_time_3 - start_time_3
C = total_time_3/(10000*n**2)

print("Ninther Quicksort")
print("single time:",total_time_3/10000)
print("est C:", C)
print("\n")

start_time_4 = time()
for rep1 in range(10000):
    insorted=insertion_sort(sortedList)

stop_time_4 = time()
total_time_4 = stop_time_4 - start_time_4
C = total_time_4 /(10000*n**2)

print("Insertion Sort")
print("single time:",total_time_4/10000)
print("est C:", C)
print("\n")
```