Part A

Aim:

1. Design and analysis algorithms for quicksort

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Pick first element as pivot.
2. Pick last element as pivot (implemented below)
3. Pick a random element as pivot.

Procedure:

1. Design algorithm and find best, average and worst-case complexity
2. Implement algorithm in any programming language.
3. Paste output

Practice Exercise:

S.no	Query statement
1	Applying divide and conquer method to sort n elements using quick sort. a) Elements can be read from a file or can be generated using random number generator. b) Determine time required to sort the elements. c) Choose random element as pivot. d) Repeat experiment for different values of n and plot the graph of the time taken versus n.
2	Repeat the experiment 1 with pivot as first and last element
3	Find the best, worst and average case complexity for 1 and 2.

Instructions:

1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

1) **Pivot as random element**

Algorithm:

Input: n lists of random numbers

Output: sorted lists using quicksort with pivot as random element and plot of the time complexity

Algorithm:

Partition algorithm:

```
def partition(arr,l,h):
    i=l
    j=h
    p=l
    global com
    while i<j:
        while arr[p]>=arr[i] and i<h:
            i+=1
            com+=1
        while arr[p]<arr[j]:
            j-=1
            com+=1
        if i<j:
            arr[i],arr[j]=arr[j],arr[i]
            com+=1

    arr[j],arr[p]=arr[p],arr[j]
    return j
```

Find pivot algorithm:

```
def piv_find(arr,l,h):
    p=random.randint(l,h)
    arr[p],arr[l]=arr[l],arr[p]
    return partition(arr,l,h)
```

Quick sort algorithm:

```
def quicksort(arr,l,h):
    if l<h:
        m=piv_find(arr,l,h)
        quicksort(arr,l,m-1)
        quicksort(arr,m+1,h)
```

Program:

```
import random
import math
import time
import matplotlib.pyplot as plt

def partition(arr,l,h):
    i=l
    j=h
    p=l
```

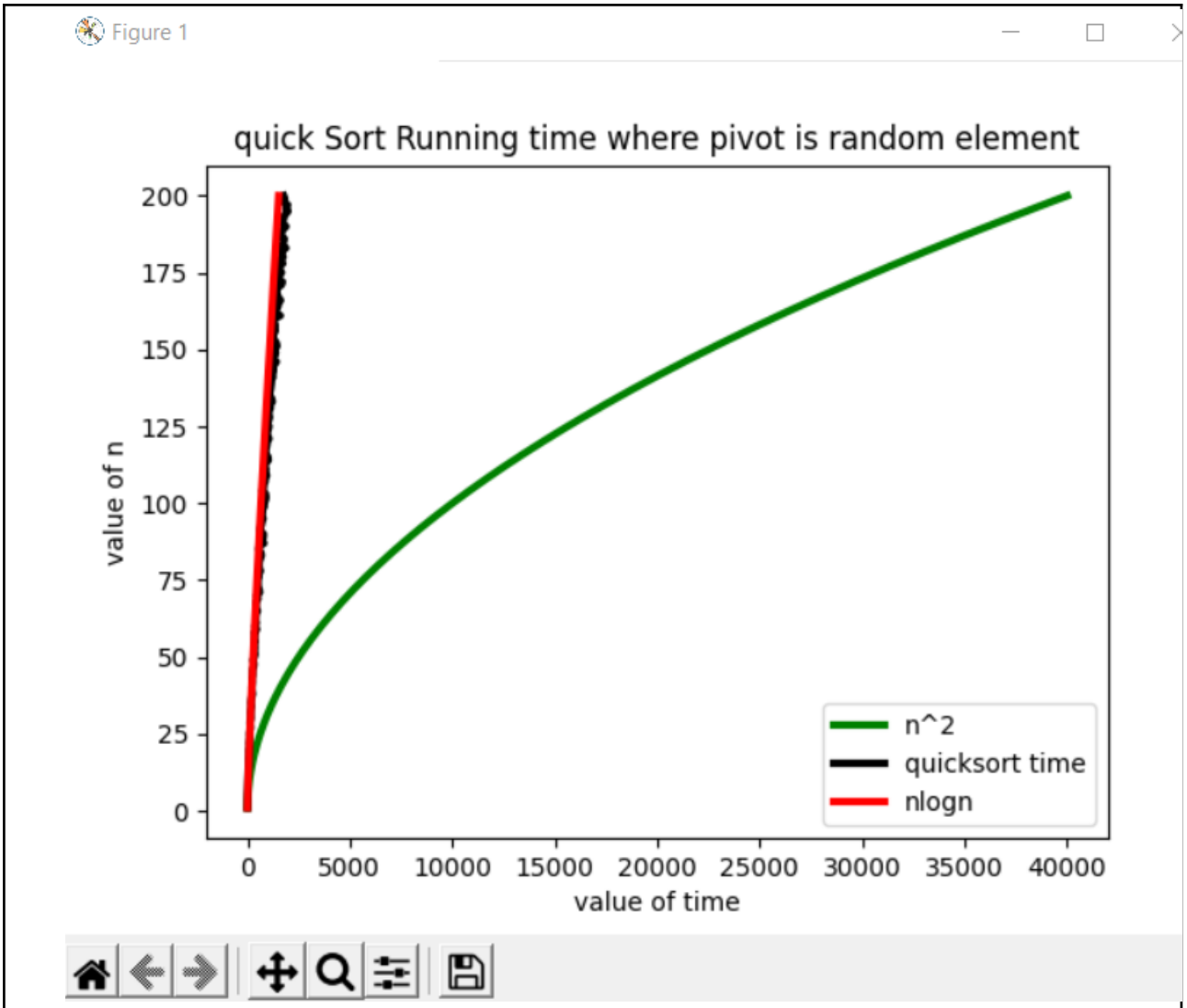
```
    global com
    while i<j:
        while arr[p]>=arr[i] and i<h:
            i+=1
            com+=1
        while arr[p]<arr[j]:
            j-=1
            com+=1
        if i<j:
            arr[i],arr[j]=arr[j],arr[i]
            com+=1

    arr[j],arr[p]=arr[p],arr[j]
    return j

def piv_find(arr,l,h):
    p=random.randint(l,h)
    arr[p],arr[l]=arr[l],arr[p]
    return partition(arr,l,h)


def quicksort(arr,l,h):
    if l<h:
        m=piv_find(arr,l,h)
        quicksort(arr,l,m-1)
        quicksort(arr,m+1,h)

comlist=[]
timeList=[]
n=int(input("number of lists:"))
for j in range(n):
    arr=[]
    for i in range(0,j):
        ran=random.randint(1,1000)
        arr.append(ran)
    com=1

    start = time.time()

    quicksort(arr,0,len(arr)-1)
    total=time.time()-start
    #print(arr)
    timeList.append(total*1000)
    comlist.append(com)
print("Average time taken to sort n lists using quick sort where pivot is
random element is",sum(timeList)/len(timeList),"seconds")
#print(arr)
```

```
n1=[*range(1,n+1,1)]
actual=[]
actual1=[]
for x in n1:
    actual.append(x*x)
    actual1.append(x*math.log(x,2))
#print(nn,n1)
plt.plot(actual,n1,color='green', linewidth=3,label='n^2')
plt.plot(comlist,n1,color='black', linewidth=3,label='quicksort time')
plt.plot(actual1,n1,color='red', linewidth=3,label='nlogn')

plt.title('quick Sort Running time where pivot is 1st element
')
plt.xlabel('value of time')
plt.ylabel('value of n')
plt.legend()
plt.show()
```

Input & Output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week4> python .\4a.py
number of lists:200
time list is  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0004043579101562, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0008811950683594, 0.0, 0.0, 0.9996891021728516, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0001659
393310547, 0.0, 0.0, 0.0, 0.0, 0.9999275207519531, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9996891021728516, 0.0, 0.0, 0.9989738464355469, 0.0, 0.0, 0.0, 0.0, 1.000404357910156
2, 0.0, 0.9999275207519531, 0.0, 0.0, 0.0, 0.0, 0.9992122650146484, 0.0, 0.0, 1.0106563568115234, 0.0, 0.0, 1.0008811950683594, 0.0, 0.0, 0.99945068359375, 0.0, 0.0, 1.00088119
50683594, 0.0, 0.0, 0.0, 0.0, 1.0004043579101562, 0.0, 0.9891986846923828, 0.0, 0.9996891021728516, 0.0, 1.0001659393310547, 0.0, 1.0004043579101562, 0.0, 0.0, 1.000642776489257
8, 0.0, 0.0, 0.0, 0.0, 1.0118484497070312, 0.0, 0.0, 0.0, 1.0001659393310547, 0.0, 0.9987354278564453, 0.0, 0.9999275207519531, 0.0, 0.9999275207519531, 0.0, 1.0001659393310547,
 0.0, 1.0004043579101562, 0.0, 1.0001659393310547, 0.0, 0.99945068359375, 0.0, 1.0004043579101562, 0.0, 0.99945068359375, 0.0, 0.0, 0.0, 0.98848342895507 81, 0.99945068359375, 0.
9989738464355469, 0.0, 0.0, 1.0001659393310547, 0.9999275207519531, 0.0, 0.9999275207519531, 0.9999275207519531, 0.0, 0.0, 0.9999275207519531, 0.0, 1.001119613647461, 0.0, 0.0,
1.0006427764892578, 0.9996891021728516, 0.9999275207519531, 0.0, 0.9996891021728516, 1.0004043579101562, 1.0004043579101562, 0.0, 0.0, 0.9984970092773438, 1.0001659393310547, 0.
0, 0.9999275207519531, 1.0004043579101562, 1.0001659393310547, 0.0, 0.0, 0.9999275207519531, 1.0004043579101562, 0.0, 0.0, 1.0006427764892578, 0.9996891021728516, 1.000642776489
2578, 0.0, 0.9999275207519531, 0.9999275207519531, 1.001119613647461, 0.9992122650146484, 0.9987354278564453, 0.0, 0.0, 0.9999275207519531, 1.0004043579101562, 0.999927520751953
1, 0.0, 0.0, 0.9999275207519531]
Average time taken to sort n lists using quick sort where pivot is random element is 0.32500267028808594 seconds
```

Plot:



2) **Pivot as first element**

Algorithm:

Input: n lists of random numbers

Output: sorted lists using quicksort with pivot as first element and plot of the time complexity

Algorithm:

Partition algorithm:

```
def partition(arr,l,h)
    p=i=l
    j=h
    global com
    while i<j:
        while arr[p]>=arr[i] and i<h:
            i+=1
            com+=1
        while arr[p]<arr[j]:
```

```
        j-=1
        com+=1
    if i<j:
        arr[i],arr[j]=arr[j],arr[i]
        com+=1

  arr[j],arr[p]=arr[p],arr[j]
  return j
```

Quick sort algorithm:

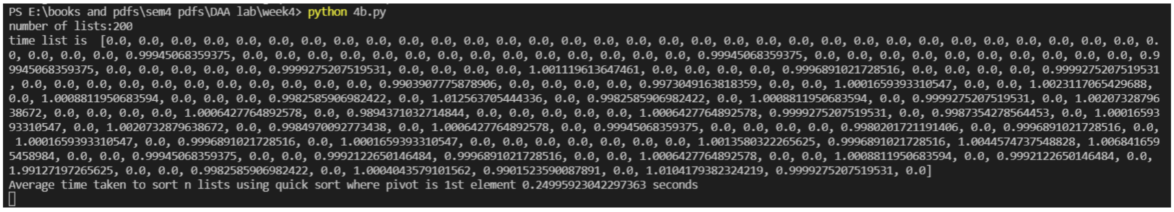
```
def quicksort(arr,l,h):
  if l<h:
    m=partition(arr,l,h)
    quicksort(arr,l,m-1)
    quicksort(arr,m+1,h)
```

Program:

```
import random
import math
import matplotlib.pyplot as plt
import time
def partition(arr,l,h):
    p=i=l
    j=h
    global com
    while i<j:
        while arr[p]>=arr[i] and i<h:
            i+=1
            com+=1
        while arr[p]<arr[j]:
            j-=1
            com+=1
        if i<j:
            arr[i],arr[j]=arr[j],arr[i]
            com+=1

    arr[j],arr[p]=arr[p],arr[j]
    return j



def quicksort(arr,l,h):
    if l<h:
        m=partition(arr,l,h)
```

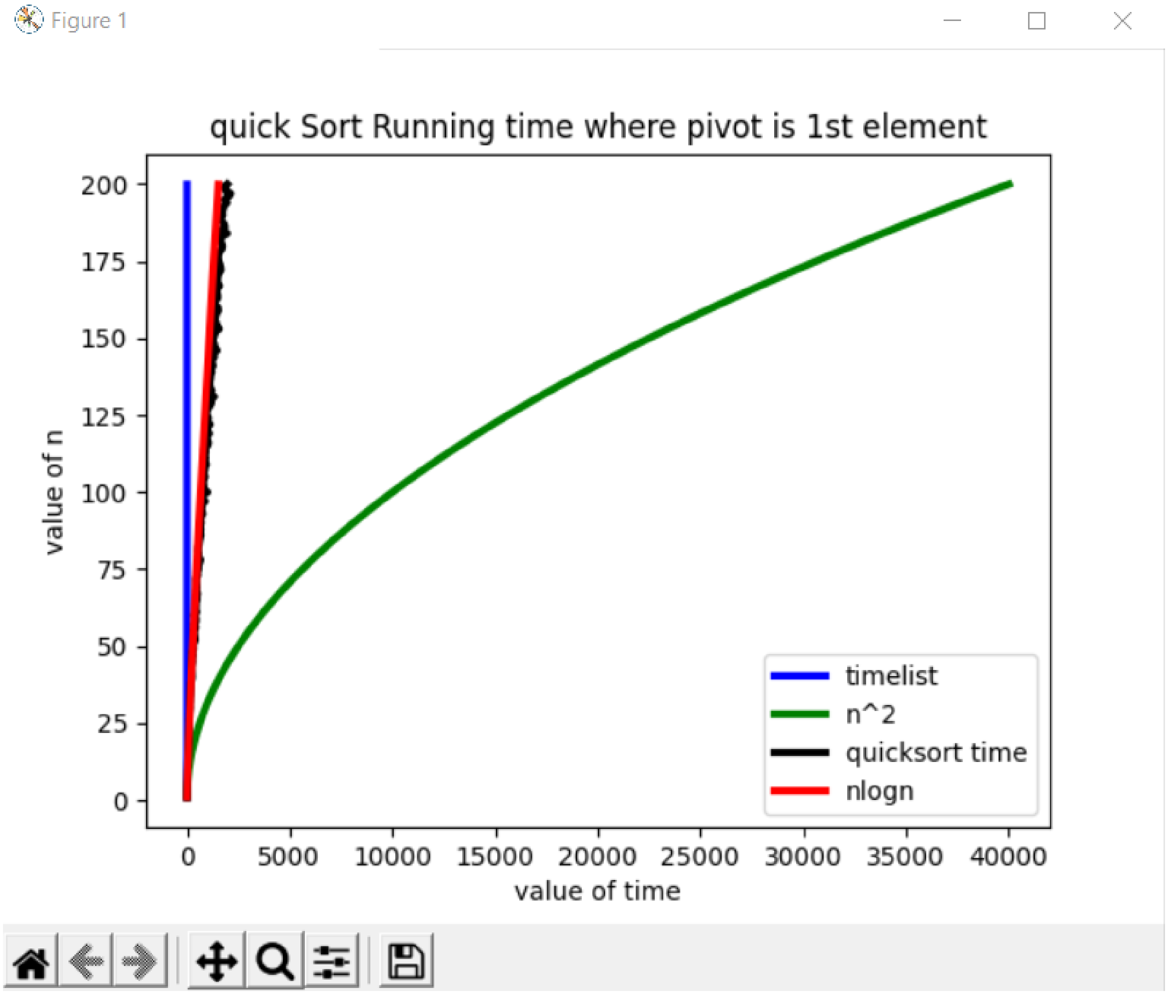
```
        quicksort(arr,l,m-1)
        quicksort(arr,m+1,h)

comlist=[]
timeList=[]
n=int(input("number of lists:"))
for j in range(n):
    arr=[]
    for i in range(0,j):
        ran=random.randint(1,1000)
        arr.append(ran)
    com=1

    start = time.time()

    quicksort(arr,0,len(arr)-1)
    total=time.time()-start
    #print(arr)
    timeList.append(total*1000)
    comlist.append(com)
print("Average time taken to sort n lists using quick sort where pivot is 1st
element",sum(timeList)/len(timeList),"seconds")
    #print(arr)

n1=[*range(1,n+1,1)]
nn=[]
nn1=[]
for x in n1:
    nn.append(x*x)
    nn1.append(x*math.log(x,2))
#print(nn,n1)
plt.plot(timeList,n1,color='blue', linewidth=3,label='timelist')

plt.plot(nn,n1,color='green', linewidth=3,label='n^2')
plt.plot(comlist,n1,color='black', linewidth=3,label='quicksort time')
plt.plot(nn1,n1,color='red', linewidth=3,label='nlogn')

plt.title('quick Sort Running time where pivot is 1st element')
plt.xlabel('value of time')
plt.ylabel('value of n')
plt.legend()
plt.show()
```

Input and Output:



```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week4> python 4b.py
number of lists:200
time list is  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.99945068359375, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.99945068359375, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9
9945068359375, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9999275207519531, 0.0, 0.0, 0.0, 0.0, 1.001119613647461, 0.0, 0.0, 0.0, 0.0, 0.9996891021728516, 0.0, 0.0, 0.0, 0.0, 0.9999275207519531
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9903907775878906, 0.0, 0.0, 0.0, 0.0, 0.9973049163818359, 0.0, 0.0, 1.0001659393310547, 0.0, 0.0, 1.0023117065429688,
0.0, 1.0008811950683594, 0.0, 0.0, 0.0, 0.9982585906982422, 0.0, 1.012563705444336, 0.0, 0.9982585906982422, 0.0, 1.0008811950683594, 0.0, 0.9999275207519531, 0.0, 1.00207328796
38672, 0.0, 0.0, 0.0, 0.0, 1.0006427764892578, 0.0, 0.9894371032714844, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0006427764892578, 0.9999275207519531, 0.0, 0.9987354278564453, 0.0, 1.00016593
93310547, 0.0, 1.0020732879638672, 0.0, 0.9984970092773438, 0.0, 1.0006427764892578, 0.0, 0.99945068359375, 0.0, 0.0, 0.0, 0.0, 0.9980201721191406, 0.0, 0.9996891021728516, 0.0,
 1.0001659393310547, 0.0, 0.9996891021728516, 0.0, 1.0001659393310547, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0013580322265625, 0.9996891021728516, 1.0044574737548828, 1.006841659
5458984, 0.0, 0.0, 0.99945068359375, 0.0, 0.0, 0.9992122650146484, 0.9996891021728516, 0.0, 0.0, 1.0006427764892578, 0.0, 0.0, 1.0008811950683594, 0.0, 0.9992122650146484, 0.0,
1.99127197265625, 0.0, 0.0, 0.9982585906982422, 0.0, 1.0004043579101562, 0.9901523590087891, 0.0, 1.0104179382324219, 0.9999275207519531, 0.0]
Average time taken to sort n lists using quick sort where pivot is 1st element 0.24995923042297363 seconds
```

Plot:



2) Pivot as last element

Algorithm:

Input: n lists of random numbers

Output: sorted lists using quicksort with pivot as last element and plot of the time complexity

Algorithm:

Partition algorithm:

def partition(arr,l,h):

```
    i=l
    p=j=h
    global com
    while i<j:
        while arr[p]>=arr[i] and i<h:
            i+=1
            com+=1
        while arr[p]<arr[j]:
            j-=1
            com+=1
        if i<j:
            arr[i],arr[j]=arr[j],arr[i]
            com+=1

    arr[j],arr[p]=arr[p],arr[j]
    return j
```

Quick sort algorithm:

```
def quicksort(arr,l,h):
    if l<h:
        m=partition(arr,l,h)
        quicksort(arr,l,m-1)
        quicksort(arr,m+1,h)
```

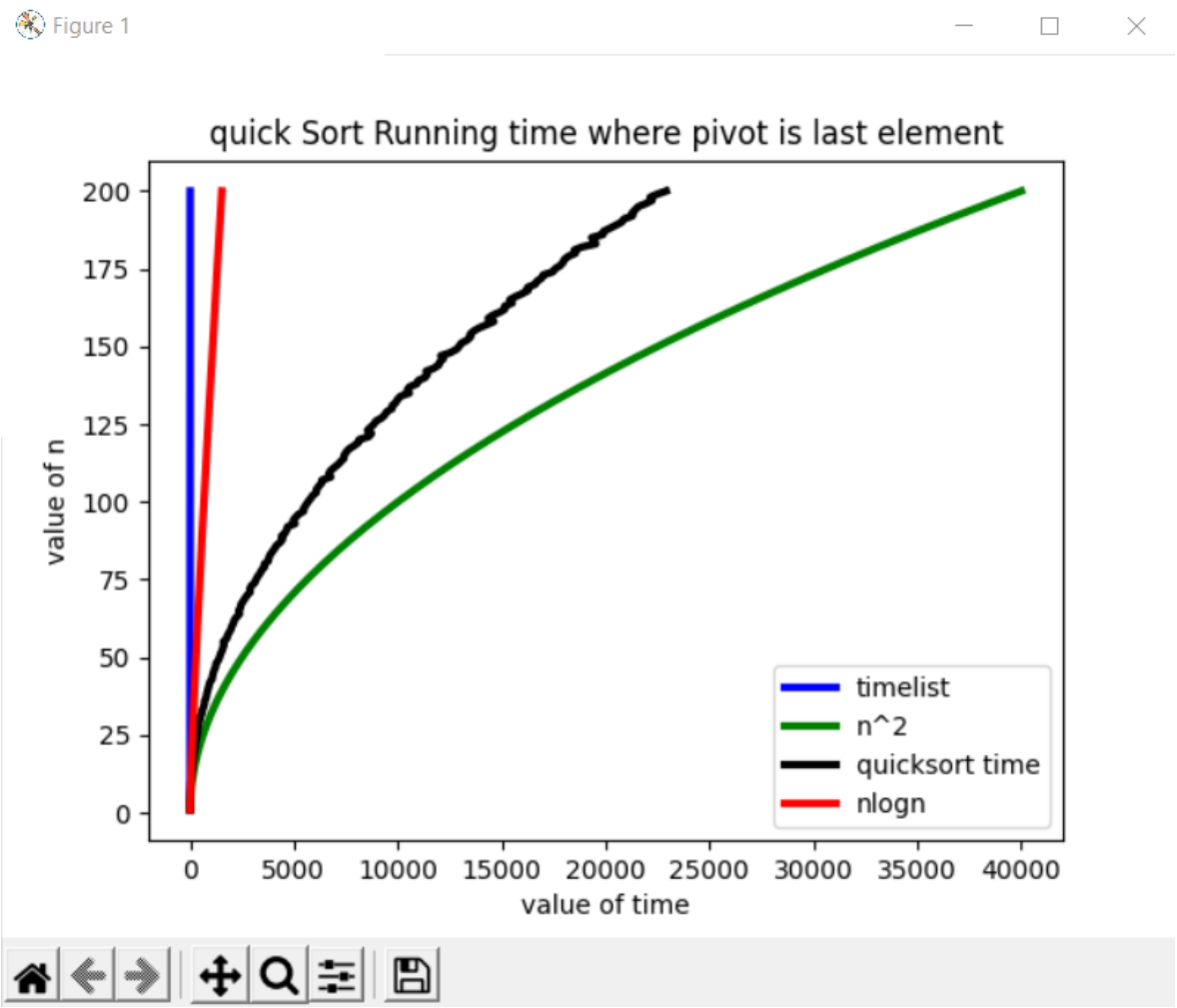
```
import random
import math
import matplotlib.pyplot as plt
import time
def partition(arr,l,h):
    i=l
    p=j=h
    global com
    while i<j:
        while arr[p]>=arr[i] and i<h:
            i+=1
            com+=1
        while arr[p]<arr[j]:
            j-=1
            com+=1
        if i<j:
            arr[i],arr[j]=arr[j],arr[i]
            com+=1

    arr[j],arr[p]=arr[p],arr[j]
```

```
    return j



def quicksort(arr,l,h):
    if l<h:
        m=partition(arr,l,h)
        quicksort(arr,l,m-1)
        quicksort(arr,m+1,h)

comlist=[]
timeList=[]
n=int(input("number of lists:"))
for j in range(n):
    arr=[]
    for i in range(0,j):
        ran=random.randint(1,1000)
        arr.append(ran)
    com=1

    start = time.time()

    quicksort(arr,0,len(arr)-1)
    total=time.time()-start
    #print(arr)
    timeList.append(total*1000)
    comlist.append(com)
print("Average time taken to sort n lists using quick sort where pivot is 1st
element",sum(timeList)/len(timeList),"seconds")
    #print(arr)

n1=[*range(1,n+1,1)]
nn=[]
nn1=[]
for x in n1:
    nn.append(x*x)
    nn1.append(x*math.log(x,2))
#print(nn,n1)
plt.plot(timeList,n1,color='blue', linewidth=3,label='timelist')

plt.plot(nn,n1,color='green', linewidth=3,label='n^2')
plt.plot(comlist,n1,color='black', linewidth=3,label='quicksort time')
plt.plot(nn1,n1,color='red', linewidth=3,label='nlogn')

plt.title('quick Sort Running time where pivot is last element')
plt.xlabel('value of time')
plt.ylabel('value of n')
plt.legend()
plt.show()
```

Input and output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week4> python 4c.py
number of lists:200
time list is  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.99825
85906982422, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.99945068359375, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0039806365966797, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.99945068359375, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.9999275207519531, 0.0, 0.0, 0.0, 0.0, 1.0008811950683594, 0.0, 0.0, 0.0, 0.9987354278564453, 0.0, 0.0, 0.0, 1.0004043579101562, 0.0, 0.0, 0.0, 0.9999275207519531,
0.0, 0.0, 0.0, 0.0, 0.0, 1.001596450805664, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9996891021728516, 0.0, 0.0, 1.0006427764892578, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.9987354278564453, 0.0, 0.0, 0.9906291961669922, 0.0, 0.9996891021728516, 0.0, 0.0, 0.9999275207519531, 0.0, 1.0001659393310547, 0.0, 0.0, 1.0023117065429688, 0.0, 0.0,
0.0, 0.0, 1.0128021240234375, 0.0, 0.0, 0.0, 0.0, 0.9992122650146484, 0.0, 0.9987354278564453, 0.0, 0.0, 0.0, 0.0, 1.0120868682861328, 0.0, 0.0, 0.0, 0.0, 0.9999275207519531, 0.
0, 0.99945068359375, 0.0, 0.0, 0.0, 0.9915828704833984, 0.0, 0.0, 0.9987354278564453, 0.0, 0.9984970092773438, 0.0, 1.0001659393310547, 0.0, 0.0, 1.0001659393310547, 0.0, 0.9999
275207519531, 1.0004043579101562, 0.0, 0.9996891021728516, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0001659393310547, 0.0, 0.9999275207519531, 0.0, 1.0008811950683594, 0.0, 0.9
989738464355469, 1.0097026824951172, 0.0, 1.0013580322265625, 0.0, 0.9977817535400391, 0.0, 0.0, 0.988006591796875, 0.0, 1.0113716125488281, 0.0, 1.0004043579101562, 0.0, 0.0, 1
.0001659393310547, 0.0, 1.0006427764892578, 0.0, 0.0, 1.0023117065429688, 0.0, 0.9982585906982422, 0.0]
Average time taken to sort n lists using quick sort where pivot is last element 0.23007869720458984 seconds
```

plot:



3) Case-wise complexities for Quicksort taking pivot as random element:

Best case : O(nlog n)
Average case: O(nlog n)
Worst case: O(n^2)
Taking pivot as random element:

Using random pivoting we improve the time complexity to O (N log N). The Worst-Case complexity is still O (N^2).

Taking pivot as first or last element:

When we take either the first or last element as pivot , the worst case may occur when the list is already sorted.

- Choosing middle element of list as pivot can avoid worst case

- Choosing a random element as pivot can also avoid the worst case, but there are chances to get 1st or last element as pivot .

Observation & Learning:

I have observed and learned that

i) time complexity of quicksort is O(nlog n) at best case but choosing pivot can make complexity worse.

ii) Complexity may become worse when the input array is sorted or reverse sorted and either the first or last element is picked as pivot.

iii) Picking pivot as middle element may result in best case .

iv) Quicksort exhibits poor performance when the input set contains lots of duplicates. Here as elements are read randomly (so the list may contain duplicates) performance of quick sort is reduced.

v)Random picking of pivot can also lead to worst case as the randomly picked element might often be 1st element or last element of list

Conclusion:

I have designed, analyzed and implemented the algorithms quicksort taking pivot as 1st, last , random elements respectively and plotted running time for the same.

Questions:

1. Is quick stable sorting?
2. Is quick internal sorting?
3. Is quick in-place sorting?

Answers:

1. No,quick stable sorting is not stable as relative order of equal sort items is not preserved
2. Yes, Quick sort is internal sorting as a data sorting process that takes place entirely within the main memory of a computer. There is no need for external memory
3. Yes Quick sort is inplace sorting as there is no use of extra / additional data structure or memory to sort the elements.