**SSN College of Engineering**

**Department of Information Technology**

**UIT2201 — Programming and Data Structures**

**2022 – 2023**

**Exercise — 06**

R Mohanakrishnaa

3122 22 5002 075

**Part A**

1. Arrange n elements either in ascending or descending order using Merge sort. Write a Python function to sort n numbers and analyze the time complexity of your code and express the same in asymptotic notation. Also, write a recursive binary search function to search for an element in the above sorted list. Analyze the time complexity of your code and express the same in asymptotic notation.

import random

import timeit

start = timeit.default\_timer()

compare\_count = 0

def merge\_sort(lst):

    global compare\_count

    length = len(lst)

    compare\_count += 1

    if length < 2:

        return lst

    else:

        mid = length // 2

        return merge(merge\_sort(lst[:mid]) , merge\_sort(lst[mid:]))

def merge(lst1, lst2):

    global compare\_count

    i, j = 0, 0

    result = []

    len1, len2 = len(lst1), len(lst2)

    compare\_count += 2

    while len1 > i and len2 > j:

        compare\_count += 3

        if lst1[i] > lst2[j]:

            result.append(lst2[j])

            j += 1

        else:

            result.append(lst1[i])

            i += 1

    compare\_count += 1

    if len1 > i:

        return result + lst1[i:]

    elif len2 > j:

        compare\_count += 1

        return result + lst2[j:]

    else:

        return result

time = timeit.default\_timer() - start

if \_\_name\_\_ == "\_\_main\_\_":

    for i in [10,100,1000,10000]:

        lst = [random.uniform(-100.00, 100.0) for \_ in range(i)]

        # lst = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

        # lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

        # print("RANDOM LIST\n",lst, "\n\n")

        print("SORTED LIST\n",merge\_sort(lst))

        print(compare\_count,"\t", time)

# worst case 10 129  2.2380002540862188e-06

#best case  10 100  1.461000010749558e-06

def partition(lst, begin, end):

    medain(lst, begin, end)

    pivot = lst[end]

    i = begin

    j = end - 1

    while i<=j:

        print(i,j)

        while lst[i] < pivot and i<=end:

            i += 1

        while lst[j] > pivot and j>=begin:

            j -= 1

        if i<j:

            lst[i],lst[j] = lst[j],lst[i]

            i+=1

            j-=1

    if i < end :

        lst[end],lst[i] = lst[i],lst[end]

    return i+1

def medain(lst, begin, end):

    mid = (begin+end) // 2

    if lst[begin] > lst[mid]:

        lst[begin], lst[mid] = lst[mid], lst[begin]

    if lst[mid] > lst[end]:

        lst[mid], lst[end] = lst[end], lst[mid]

    if lst[begin] > lst[end]:

        lst[begin], lst[end] = lst[end], lst[begin]

    lst[mid], lst[end] = lst[end], lst[mid]

def Quick\_sort(lst, start, end):

    if end > start:

        p = partition(lst, start, end-1)

        Quick\_sort(lst, start, p-1)

        Quick\_sort(lst, p, end)

alist = [9,8,7,6,5,4,3,2,1]

Quick\_sort(alist, 0, len(alist))

print('Sorted list: ', alist)

import timeit

import random

start = timeit.default\_timer()

compare\_count = 0

def binear\_search(lst, find, low, high):

    global compare\_count

    mid = (high + low) // 2

    compare\_count += 2

    if low > high:

        return -1

    if lst[mid] == find:

        return mid

    elif lst[mid] > find:

        compare\_count += 1

        return binear\_search(lst, find , low, mid-1)

    else:

        return binear\_search(lst, find , mid+1, high)

time = timeit.default\_timer() - start

if \_\_name\_\_ == "\_\_main\_\_":

    for i in [10,100,1000,1000]:

        lst = [random.uniform(-100.00,100.00) for \_ in range(i)]

        lst.sort()

        find\_idx = random.randint(0,len(lst)-1)

        find = lst[find\_idx]

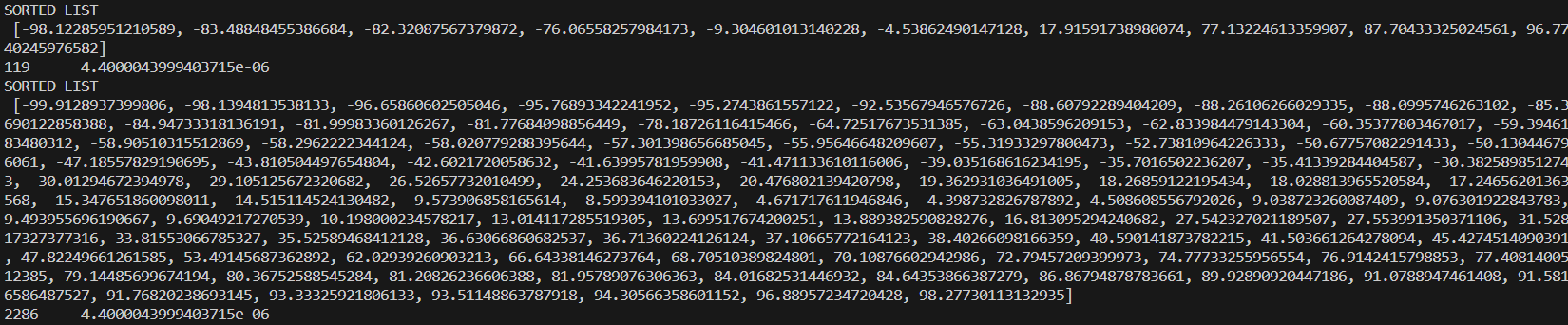
        print(binear\_search(lst, find, 0, len(lst)))

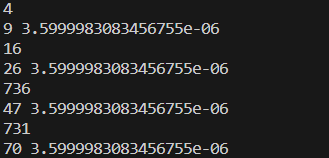
        print(compare\_count, time)

# woste case 10 5  9.300001693191007e-07

# best case  10 1  1.11999906948767e-06

Output:





**Part B**

1. Write a Python function to sort n numbers using Quick sort and analyze the time complexity of your code using the number of comparisons required and express the same in asymptotic notation.

import timeit

import random

start = timeit.default\_timer()

compare\_count = 0

def partition(lst, begin, end):

    global compare\_count

    median(lst,begin,end)

    pivot = lst[end]

    i = begin

    j = end - 1

    while i<=j:

        compare\_count += 1

        while lst[i] < pivot and i<=end:

            compare\_count += 2

            i += 1

        while lst[j] > pivot and j>=begin:

            compare\_count += 2

            j -= 1

        compare\_count += 1

        if i<j:

            lst[i],lst[j] = lst[j],lst[i]

            i += 1

            j -= 1

    compare\_count += 1

    if i < pivot:

        lst[i],lst[end] = lst[end],lst[i]

    return i

def quick\_sort(lst, begin, end):

    global compare\_count

    compare\_count += 1

    if begin < end:

        p = partition(lst, begin, end-1)

        quick\_sort(lst, begin, p-1)

        quick\_sort(lst, p+1, end)

def median(lst, begin, end):

    global compare\_count

    mid = (begin+end)//2

    compare\_count += 3

    if lst[mid] > lst[begin]:

        lst[mid],lst[begin] = lst[begin],lst[mid]

    if lst[mid] < lst[end]:

        lst[mid],lst[end] = lst[end],lst[mid]

    if lst[begin] > lst[end]:

        lst[begin],lst[end] = lst[end],lst[begin]

    lst[mid],lst[end]=lst[end],lst[mid]

end = timeit.default\_timer()

time = end-start

if \_\_name\_\_ == "\_\_main\_\_":

    for i in [10,100,500,1000,5000,1000]:

        lst = [random.uniform(-100.00,100.00) for \_ in range(i)]

        begin=0

        end=len(lst)

        quick\_sort(lst,begin,end)

        print(lst)

        print(compare\_count, time)

Output:

