EX - 6 SORTING ALGORITHM

T SADAKOPA RAMAKRISHNAN | 3122225002109 | IT - B

Q1) 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.

AIM:

To create a function that sorts the given elements in ascending order using merge sort and perform a binary search using the sorted list from merge sort.

CODE:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that sorts the
given elements in ascending order using merge sort which
has the time complexity of nlogn. This module also contains
binary search that takes in the sorted list from
merge sort and then search for the search element. It has
time complexity of logn. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed May 10 2023

Revised on Wed May 24 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""
```

```
import random
from timeit import default_timer as timer

def createLIST(n):
    '''
    This function creates a sequence of integer using random
module
    '''
    seq = []
    for i in range(n):
        seq.append(random.randint(-100,100))
    print("Original List:", seq)
    return seq

def Merge(a,b):
    '''
    Merge function takes in two lists a and b and then
    merges the two lists into one with the elements
    arranged in ascending order.

    Input : 2 lists
    Returns : a merged list
    '''
    c = []
    i = j = 0
    m = len(a)
    n = len(b)
    while (i < m and j < n):
        if b[j] < a[i]:
            c.append(b[j])
            j += 1
        else:
            c.append(a[i])
            i+=1
    if i < len(a):
        c.extend(a[i:])
    elif j < len(b):
        c.extend(b[j:])
    return c
```

```
def MergeSort(list):
    '''
    This function splits the input list into two and
    calls the merge() function to merge the two lists
    with elements arranged in ascending order by using
    recursion.
    '''
    n = len(list)
    if n < 2:
        return list[:]
    else:
        mid = n // 2
        return Merge(MergeSort(list[:mid]),
MergeSort(list[mid:]))

def BinarySearch(list, low, high, srele):
    '''
    Takes in a sorted list and searches for a search
    element by dividing the list into two for every iteration
    '''
    if low > high:
        return -1
    else:
        mid = (low + high) // 2
        if srele == list[mid]:
            return mid
        elif srele < list[mid]:
            return BinarySearch(list,low, mid-1, srele)
        elif srele > list[mid]:
            return BinarySearch(list, mid+1, high, srele)

start = timer()
print("Sorted List: ", MergeSort(createLIST(10)))
end = timer()
print("Time taken is", end - start)

print("The search element is present
at",BinarySearch(MergeSort(createLIST(10)),0,9,15))
```

OUTPUT:

```
Original List: [-40, -10, -41, 3, 95, -65, 11, -47, -79, -38]
Sorted List:  [-79, -65, -47, -41, -40, -38, -10, 3, 11, 95]
Time taken is 0.00014970499978517182
Original List: [34, 14, 86, -2, -6, -3, -75, 91, -83, 71]
The search element is present at -1
```

Q2) 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

Aim:

To create a python program that sorts the given elements in ascending order using a quick sort algorithm.

Code:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that sorts the
given elements in ascending order using quick sort which
has the time complexity of nlogn. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed May 10 2023

Revised on Wed May 24 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""
```

```
import random
from timeit import default_timer as timer

def createLIST(n):
    '''
    This function creates a sequence of integer using random
module
    '''
    seq = []
    for i in range(n):
        seq.append(random.randint(-100,100))
    print("Original List:", seq)
    return seq

def median(L):
    '''
    This function returns the median index to use that
    as the pivot element. It swaps the median element to the
    last position.
    '''
    n = len(L)
    low, mid, high = L[0], L[n//2], L[n-1]
    if low > high:
        L[0], L[n-1], L[n-1], L[0]
    if high > mid:
        L[n-1], L[n//2] = L[n//2], L[n-1]
    if low > mid:
        L[0], L[n//2] = L[n//2], L[0]
    return L


def partition(L, begin, end):
    '''
    This function partition the given list using in such a way
    the elements to the left of the pivot element are lesser
than
    pivot element and elements to the right of the pivot
elements
    are greater than the pivot elements.
    '''
    comparisons = 0
```

```
    swappings = 0
    pivot = L[end]
    i, j = begin, end - 1
    while(i<=j):
        while(L[i]<=pivot and i<end):
            comparisons += 1
            i += 1
        while(L[j]>pivot and j>=begin):
            comparisons += 1
            j -= 1
        if i<j:
            swappings += 1
            L[i], L[j] = L[j], L[i]
    swappings += 1
    L[i], L[end] = L[end], L[i]
    return i, L[i: i+1], comparisons, swappings


def quick_sort(L):
    '''
    This function takes the partition list and then
    sort it in ascending order.
    '''
    if len(L) < 2:
        return L, 0, 0
    else:
        L = median(L)
        postion, mid, comparisons1,swappings1 = partition(L, 0,
len(L) - 1)
        lhs, comparisons2, swappings2 = quick_sort(L[:postion])
        rhs, comparisons3,swappings3 = quick_sort(L[postion +
1:])
        return lhs + mid + rhs, comparisons1 + comparisons2 +
comparisons3,swappings1 + swappings2+ swappings3


start = timer()
list,comparisons, swappings = quick_sort(createLIST(10))
print("Sorted List: ")
print(list)
print("Total Comparisons:", comparisons)
```

```
print("Total Swapings:", swappings)
end = timer()
print("The time taken is", end - start)
```

Output:

```
Original List: [32, -88, -99, -81, 3, 35, 15, -61, -73, 43]
Sorted List:
[-99, -88, -81, -73, -61, 3, 15, 32, 35, 43]
Total Comparisons: 23
Total Swapings: 7
The time taken is 0.00024080400180537254
```