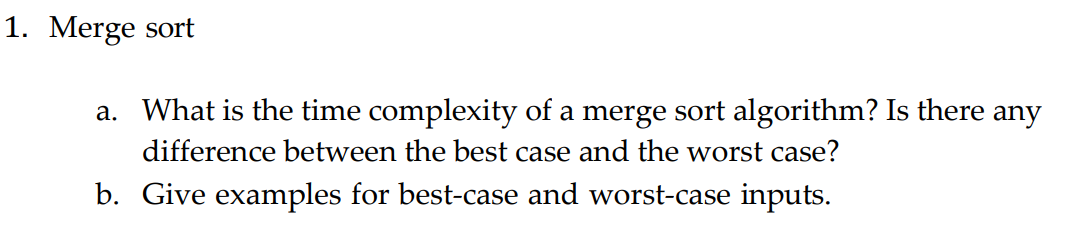


Name : Anuvind M P

Roll no: AM.EN.U4AIE22010



**Code :**

#1

def ms(arr):

    if len(arr) > 1:

        left = arr[:len(arr)//2]

        right = arr[len(arr)//2:]

        ms(left)

        ms(right)

        merge(arr,left,right)

def merge(arr,left,right):

    i = j = k = 0

    while i < len(left) and j < len(right):

        if left[i] < right[j]:

            arr[k] = left[i]

            i+=1

        else:

            arr[k] = right[j]

            j+=1

        k+=1

        while i<len(left):

            arr[k] = left[i]

            i+=1

            k+=1

        while j<len(right):

            arr[k] = right[j]

            j+=1

            k+=1

arr = [3,7,2,9,1,2,5,7]

ms(arr)

print(arr)

**Output :**

****

**a. Time Complexity of Merge Sort:**

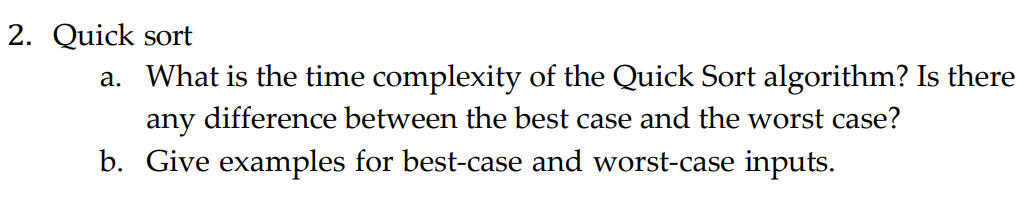
* **Best Case:** O(nlogn)
* **Worst Case:** O(nlogn)

There is no difference between the best case and the worst case time complexity for merge sort. This is because the algorithm always divides the list into two halves and takes linear time to merge them, regardless of the initial order of the elements.

**b. Examples of Best-Case and Worst-Case Inputs:**

* **Best Case Input:** Any input is the best case since merge sort consistently performs O(nlogn) operations regardless of the input.
* **Worst Case Input:** Similar to the best case, the worst-case performance is also O(nlogn) for any input.

Example: [5,4,3,2,1]



**Code :**

#2

def partition(l,h,arr):

    i,j = l,h

    pivot = arr[l]

    while True:

        while i<=j and arr[i]<=pivot:

            i+=1

        while i<=j and arr[j]>=pivot:

            j -=1

        if i<=j:

            arr[i],arr[j] = arr[j],arr[i]

        else:

            break

    arr[l], arr[j] = arr[j],arr[l]

    return j

def quicksort(l,h,arr):

    if l<h:

        pivot = partition(l,h,arr)

        quicksort(l,pivot-1,arr)

        quicksort(pivot+1,h,arr)

arr = [4,8,1,1,2,0]

quicksort(0,len(arr)-1,arr)

print(arr)

**Output :**

****

**a. Time Complexity of Quick Sort:**

* **Best Case:** O(nlogn)
* **Average Case:** O(nlogn)
* **Worst Case:** O(n2)

Best,worst and avg case varies due to the selection of pivot and unbalanced partitions.

**b. Examples of Best-Case and Worst-Case Inputs:**

* **Best Case Input:** [3,1,2,5,4] (divides list into nearly equal halves in each partition)
* **Worst Case Input:** The worst case occurs when the array is sorted or reverse sorted making the pivot the smallest or largest element in each iteration. It also leads to unbalanced partitioning.
  + Example: [1,2,3,4,5]

****

**Code :**

#3

def counting\_sort(arr):

    count = [0] \* 256

    output = [''] \* len(arr)

    for char in arr:

        count[ord(char)] += 1

    for i in range(1, 256):

        count[i] += count[i - 1]

    for char in reversed(arr):

        output[count[ord(char)] - 1] = char

        count[ord(char)] -= 1

    return output

s = ["a", "p", "p", "l", "e"]

s = counting\_sort(s)

print(s)

**Output :**

****

****

**Code :**

#4

def countingSort(arr, exp1):

    n = len(arr)

    output = [0] \* (n)

    count = [0] \* (10)

    for i in range(0, n):

        index = arr[i] // exp1

        count[index % 10] += 1

    for i in range(1, 10):

        count[i] += count[i - 1]

    i = n - 1

    while i >= 0:

        index = arr[i] // exp1

        output[count[index % 10] - 1] = arr[i]

        count[index % 10] -= 1

        i -= 1

    for i in range(0, len(arr)):

        arr[i] = output[i]

def radixSort(arr):

    max1 = max(arr)

    exp = 1

    while max1 / exp >= 1:

        countingSort(arr, exp)

        exp \*= 10

arr = [180, 25, 72, 507, 34, 2, 99]

radixSort(arr)

print("sorted array : ",arr)

**Output :**

****

****

**Code :**

def heapify(arr, n, i):

    largest = i

    left = 2 \* i + 1

    right = 2 \* i + 2

    if left < n and arr[i] < arr[left]:

        largest = left

    if right < n and arr[largest] < arr[right]:

        largest = right

    if largest != i:

        arr[i], arr[largest] = arr[largest], arr[i]

        heapify(arr, n, largest)

def heap\_sort(arr):

    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):

        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):

        arr[i], arr[0] = arr[0], arr[i]

        heapify(arr, i, 0)

arr = [180, 25, 72, 507, 34, 2, 99]

heap\_sort(arr)

print("sorted : ",arr)

**Output :**

****