|  |  |
| --- | --- |
| **Name** | **Vedant Sanjay Deshmukh** |
| **UID** | **2021300025** |
| **Subject** | **Data Analysis Algorithm** |
| **Experiment No** | **2** |

**Aim-**

Experiment based on divide and conquers approach.

**Algorithm-**

**MERGE SORT ALGORITHM**

1. MERGE\_SORT(arr, beg, end)
3. **if** beg < end
4. set mid = (beg + end)/2
5. MERGE\_SORT(arr, beg, mid)
6. MERGE\_SORT(arr, mid + 1, end)
7. MERGE (arr, beg, mid, end)
8. end of **if**
10. END MERGE\_SORT

**QUICK SORT ALGORITHM**

partition (arr[], low, high)

{ // pivot (Element to be placed at right position)

pivot = arr[high];

i = (low – 1) // Index of smaller element and indicates the

// right position of pivot found so far

for (j = low; j <= high- 1; j++){

// If current element is smaller than the pivot

if (arr[j] < pivot){

i++; // increment index of smaller element

swap arr[i] and arr[j]

}

}

swap arr[i + 1] and arr[high])

return (i + 1)

}

quickSort(arr[], low, high) {

if (low < high) {

/\* pi is partitioning index, arr[pi] is now at right place \*/

pi = partition(arr, low, high);

quickSort(arr, low, pi – 1); // Before pi

quickSort(arr, pi + 1, high); // After pi

}

}

**Code-**

**1.**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**void merge(int arr[],int beg,int mid,int end)**

**{**

**int temp[100000];**

**int i=beg,j=mid+1,index=0,k;**

**while(i<=mid && j<=end)**

**{**

**if(arr[i]<arr[j])**

**{**

**temp[index]=arr[i];**

**i++;**

**index++;**

**}**

**else**

**{**

**temp[index]=arr[j];**

**j++;**

**index++;**

**}**

**}**

**while(i<=mid)**

**{**

**temp[index]=arr[i];**

**i++;**

**index++;**

**}**

**while(j<=end)**

**{**

**temp[index]=arr[j];**

**j++;**

**index++;**

**}**

**for(i=beg,k=0;i<=end;i++,k++)**

**{**

**arr[i]=temp[k];**

**}**

**}**

**void merge\_sort(int arr[],int beg,int end)**

**{**

**int mid;**

**if(beg<end)**

**{**

**mid=(beg+end)/2;**

**merge\_sort(arr,beg,mid);**

**merge\_sort(arr,mid+1,end);**

**merge(arr,beg,mid,end);**

**}**

**}**

**int partition(int arr[],int lb,int ub)**

**{**

**int pivot=arr[lb];**

**int start=lb;**

**int end=ub,temp;**

**while(start<end)**

**{**

**while( arr[start]<=pivot)**

**{**

**start++;**

**}**

**while(pivot<arr[end])**

**{**

**end--;**

**}**

**if(start<end)**

**{**

**temp=arr[start];**

**arr[start]=arr[end];**

**arr[end]=temp;**

**}**

**}**

**temp=arr[lb];**

**arr[lb]=arr[end];**

**arr[end]=temp;**

**return end;**

**}**

**void quicksort(int arr[],int lb,int ub)**

**{**

**int loc;**

**if(lb<ub)**

**{**

**loc=partition(arr,lb,ub);**

**quicksort(arr,lb,loc-1);**

**quicksort(arr,loc+1,ub);**

**}**

**}**

**int main()**

**{**

**int n=0;**

**for(int k=0; k<(100000/100); k++)**

**{**

**n=n+100;**

**int num[n];**

**int arr[n];**

**int merge\_arr[n];**

**int j, min;**

**clock\_t start\_t, end\_t;**

**double total\_t;**

**printf("%d\t",n);**

**for(int i=0; i<n; i++)**

**{**

**num[i]=rand() % 10;**

**arr[i]=num[i];**

**merge\_arr[i]=num[i];**

**}**

**start\_t = clock();**

**quicksort(arr,0,n-1);**

**end\_t = clock();**

**total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;**

**printf("%f\t", total\_t );**

**start\_t = clock();**

**merge\_sort(merge\_arr, 0 ,n-1);**

**end\_t = clock();**

**total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;**

**printf("%f\t", total\_t );**

**printf("\n");**

**}**

**return 0;**

**}**

**Conclusion-**

**Thus I have understood the Merge and Quick sort algorithm and their time complexities. I also understood how to calculate them and draw similar inferences.**