|  |  |
| --- | --- |
| **Name** | **Harshal Sunil Chawan** |
| **UID** | **2021300019** |
| **Subject** | **Data Analysis Algorithm** |
| **Experiment No** | **2** |

# Aim-

1. To implement and find the runtime of divide and conquer sorting algorithms(merge sort and quick sort).

# Algorithm-

1. **Merge Sort**
   1. start
   2. declare array and left, right, mid variable
   3. perform merge function. if left > right

return

mid= (left+right)/2 mergesort(array, left, mid) mergesort(array, mid+1, right) merge(array, left, mid, right)

* 1. Stop

1. **Quick Sort**
   1. quickSort(arr[], low, high) {
   2. if (low < high) {
      1. /\* pi is partitioning index, arr[pi] is now at right place \*/
      2. pi = partition(arr, low, high);
      3. quickSort(arr, low, pi – 1); // Before pi
      4. quickSort(arr, pi + 1, high); // After pi

}

}

* 1. partition (arr[], low, high)

{

// pivot (Element to be placed at right position) pivot = arr[high];

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

// right position of pivot found so far

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

}

# Code-

#include <stdio.h> #include<stdlib.h> #include<time.h>

void merge(int mrgsort[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1; int n2 = r - m;

int Left[n1], Right[n2]; for (i = 0; i < n1; i++)

Left[i] = mrgsort[l + i]; for (j = 0; j < n2; j++)

Right[j] = mrgsort[m + 1 + j]; i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) { if (Left[i] <= Right[j]) {

mrgsort[k] = Left[i]; i++;

} else {

mrgsort[k] = Right[j]; j++;

} k++;

}

while (i < n1) { mrgsort[k] = Left[i]; i++;

k++;

}

while (j < n2) { mrgsort[k] = Right[j]; j++;

k++;

}

}

void mergesort(int mrgsort[], int count, int n)

{

if (count < n) {

int temp = count + (n - count) / 2; mergesort(mrgsort, count, temp); mergesort(mrgsort, temp + 1, n); merge(mrgsort, count, temp, n);

}

}

void display(int mrgsort[], int quicksort[], int n)

{

for(int i=0; i<n; i++) { printf("%d\t%d\n",mrgsort[i],quicksort[i]);

}

}

void swap(int \*a, int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int array[], int low, int high)

{

int pivot = array[high]; int i = (low - 1);

for (int j = low; j < high; j++) { if (array[j] <= pivot) {

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]); return (i + 1);

}

void quickSort(int array[], int low, int high)

{

if (low < high) {

int pi = partition(array, low, high); quickSort(array, low, pi - 1); quickSort(array, pi + 1, high);

}

}

void printArray(int array[], int size)

{

for (int i = 0; i < size; ++i) { printf("%d ", array[i]);

}

printf("\n");

}

void main()

{

int n=0;

for(int j=0; j<(10000/100); j++)

{

n=n+100;

int num[n];

int mrgsort[n]; int quicksort[n];

clock\_t start\_t, end\_t; double total\_t;

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

num[i]=rand() % 10; mrgsort[i]=num[i]; quicksort[i]=num[i];

}

printf("%d\t",n); start\_t = clock();

mergesort(mrgsort, 0, n - 1); end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC; printf("%f\t", total\_t );

start\_t = clock(); quickSort(quicksort, 0, n - 1); end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC; printf("%f\n", total\_t );

//display(mrgsort, quicksort, n);

}

}

# 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.