**PRACTICAL - 3**

**3.1 AIM: Implement and perform analysis of worst case of Merge Sort and Quick sort. Compare both algorithms.**

**PROGRAM CODE**: **Merge Sort**

#include <iostream>

using namespace std;

int counter = 0;

void merge(int arr[], int p, int q, int r)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

M[j] = arr[q + 1 + j];

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2)

{

if (L[i] <= M[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = M[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = M[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

counter++;

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

int size;

cout << "Enter the size of the array: ";

cin >> size;

int arr[size];

cout << "Enter the values in the array: " << endl;

for (int i = 0; i < size; i++)

{

cin >> arr[i];

}

mergeSort(arr, 0, size - 1);

cout << "The sorted array will be: " << endl;

for (int i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

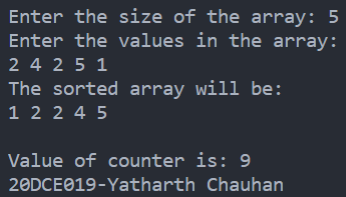
cout << "\nValue of counter is: " << counter;

cout << "\n20DCE019-Yatharth Chauhan ";

return 0;

}

**OUTPUT**:



**PROGRAM CODE**: **Quick Sort**

#include <iostream>

using namespace std;

int counter = 0;

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

{

int pivot = arr[high];

int i = low - 1;

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

{

if (arr[j] <= pivot)

{

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[high]);

return i + 1;

}

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

{

counter++;

if (low < high)

{

int pivotIndex = partition(arr, low, high);

quickSort(arr, low, pivotIndex - 1);

quickSort(arr, pivotIndex + 1, high);

}

}

int main()

{

int size;

cout << "Enter the size of the array: ";

cin >> size;

int arr[size];

cout << "Enter the values in the array: " << endl;

for (int i = 0; i < size; i++)

{

cin >> arr[i];

}

quickSort(arr, 0, size - 1);

cout << "The sorted array will be: " << endl;

for (int i = 0; i < size; i++)

{

cout << arr[i] << " ";

}

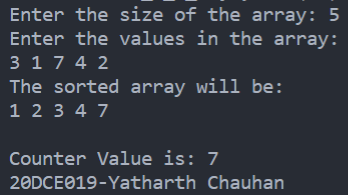
cout << "\n\nCounter Value is: " << counter;

cout << "\n20DCE019-Yatharth Chauhan ";

return 0;

}

**OUTPUT:**

****

**ANALYSIS TABLE**:

|  |  |  |  |
| --- | --- | --- | --- |
| No. of elements in Merge Sort | Merge Sort | No. of elements in Quick Sort | Quick Sort |
| 4 | 7 | 4 | 7 |
| 5 | 9 | 5 | 9 |
| 8 | 15 | 8 | 15 |
| 10 | 19 | 10 | 19 |

**GRAPH**:

**3.2 AIM:** **Implement the program to perform Linear Search and Binary Search. Also compare Time complexity of both.**

**PROGRAM CODE: Binary Search**

#include<iostream>

using namespace std;

int count=0;

int binarySearch(int arr[],int start,int end, int key) {

count++;

int size=end-start+1;

// int start = 0;

// int end = size-1;

int mid = start + (end-start)/2;

while(start <= end) {

if(arr[mid] == key) {

return mid;

}

if(key > arr[mid]) {

return binarySearch(arr, mid+1, end, key);

}

else{ //key < arr[mid]

return binarySearch(arr, start, mid-1, key);

}

mid = start + (end-start)/2;

}

return -1;

}

int main() {

int T;

cin>>T;

while(T>0)

{

int n;

cin>>n;

int arr[n];

int key;

cout<<"Enter key element"<<endl;

cin>>key;

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

{

cin>>arr[i];

}

int evenIndex = binarySearch(arr,0,n-1, key);

cout << " Index of key is " << evenIndex << endl;

T--;

}

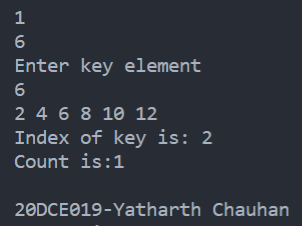
cout<<"Count is:"<<count<<endl;

cout << "\n20DCE019-Yatharth Chauhan ";

return 0;

}

**OUTPUT:**



**ANALYSIS TABLE:**

|  |  |
| --- | --- |
| Input | Counter |
| 6 | 1 |
| 8 | 3 |
| 9 | 4 |

**GRAPH:**

**Program code: Linear search**

#include<iostream>

using namespace std;

int count=0;

void print(int arr[], int n) {

cout << "Size of array is " << n << endl;

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

cout << arr[i] << " ";

} cout << endl;

}

bool linearSearch(int arr[], int size, int k ) {

// print(arr,size);

count++;

//base case

if(size == 0)

return false;

if(arr[0] == k) {

return true;

}

else {

bool remainingPart = linearSearch(arr+1, size-1, k );

return remainingPart;

}

}

int main() {

int T;

cin>>T;

while(T>0){

int n;

cin>>n;

int arr[n];

int key;

cin>>key;

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

{

cin>>arr[i];

}

int size = n;

bool ans = linearSearch(arr, size, key);

if(ans ){

cout << "Present " << endl;

}

else{

cout << "absent " << endl;

}

T--;

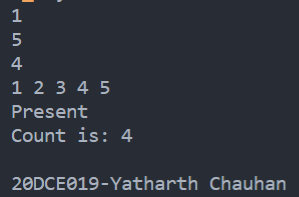
}

cout<<"Count is: "<<count<<endl;

return 0;

}

**OUTPUT:**

****

**Analysis table:**

|  |  |
| --- | --- |
| No of elements | Counter |
| 5 | 5 |
| 3 | 1 |
| 4 | 3 |

**Graph:**

**CONCLUSION:**

In this practical we have learnt the basics of divide and conquer strategy. We have also tried to optimize the power of element problem using divide and conquer strategy