Experiment No. : 2

Title: Divide and Conquer Strategy

Batch:B1 **Roll No.: 16010420133** **Experiment No.:2**

Aim: To implement and analyse time complexity of Quick-sort and Merge sort and compare both.

Explanation and Working of Quick sort & Merge sort:

Algorithm of Quick sort & Merge sort:

Merge sort :

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Quick sort :

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Derivation of Analysis Quick sort :

Worst Case Analysis

T(n) = O(n2).

Best Case Analysis

T(n) = O(n(log(n))).

Derivation of Analysis Merge sort:

Worst Case Analysis

T(n) = O(n2).

Best Case Analysis

T(n) = O(n(log(n))).

Program(s) of Quick sort:

```
#include <iostream>
using namespace std;

int c=0;

void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}
void printArray(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++)
  cout << arr[i] << " ";
  cout << endl;
}
int partition(int arr[], int p, int r) {

  int pivot = arr[r];

  int i = (p - 1);

  for (int j = p; j < r; j++) {
    if (arr[j] <= pivot) {

      i++;
```

```
      swap(&arr[i], &arr[j]);
      c++;
    }
  }

  swap(&arr[i + 1], &arr[r]);
  c++;

  return (i + 1);
}

void quickSort(int arr[], int p, int r) {
  if (p < r) {

    int pi = partition(arr, p, r);

    quickSort(arr, p, pi - 1);

    quickSort(arr, pi + 1, r);
  }
}

int main() {
  int size = 100;
  int data[size];
  for(int i=0;i<size;i++){
    data[i] = i+1;
  }
  int n = sizeof(data) / sizeof(data[0]);

  cout << "Unsorted Array: \n";
  printArray(data, n);

  quickSort(data, 0, n - 1);

  cout << "Sorted arr in ascending order: \n";
  printArray(data, n);

  cout << "No of passes \n"<<c;
}
```

Merge sort:

```
#include <iostream>
using namespace std;

int c=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++;
      c++;
    } else {
      arr[k] = M[j];
      j++;
      c++;
    }
    k++;
  }

  while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
    c++;
  }

  while (j < n2) {
    arr[k]  =  M[j];
    j++;
    k++;
    c++;
  }
}

void mergeSort(int arr[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);
```

```
    merge(arr, l, m, r);
  }
}

void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++)
    cout << arr[i] << " ";
  cout << endl;
}

int main() {
  int n = 100;
  int arr[n];
  for(int i=0;i<n;i++){
    arr[i] = i+1;
  }
  int size = sizeof(arr) / sizeof(arr[0]);

  mergeSort(arr, 0, size - 1);

  /* cout << "Sorted array: \n";
  printArray(arr, size); */

  cout << "No of passes \n"<<c;
  return 0;
}
```

Output(o) of Quick sort:

Merge sort:

Results:

Time Complexity of Quick sort:

Worst Case Analysis:

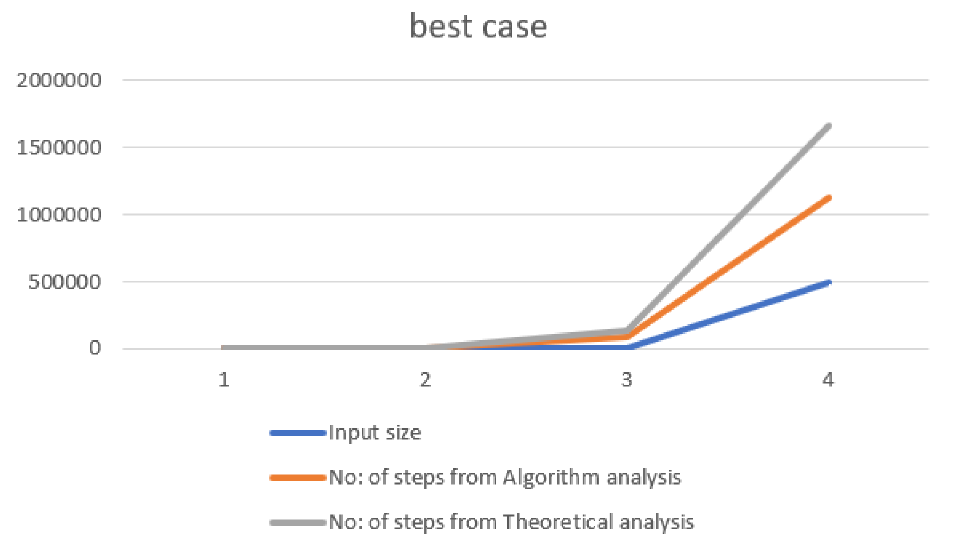
Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1	100	5030	10000
2	500	125247	250000
3	10000	50003998	100000000
4	500000	$\sim 125 * 10^9$	250000000000

GRAPH:

Best Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1	100	320	664
2	500	2784	4482
3	10000	85031	132877
4	500000	1130033	1660964

GRAPH

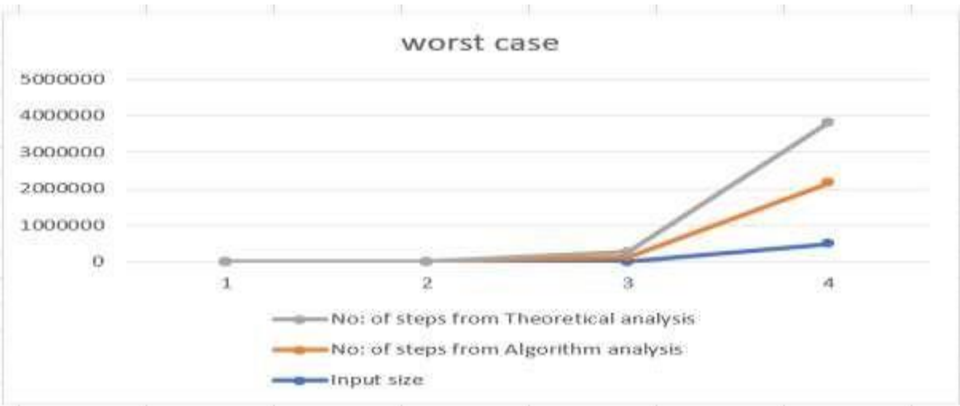


Time Complexity of Merge sort:

Worst Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1	100	664	654
2	500	4468	4462
3	10000	133516	132787
4	500000	1668822	1660854

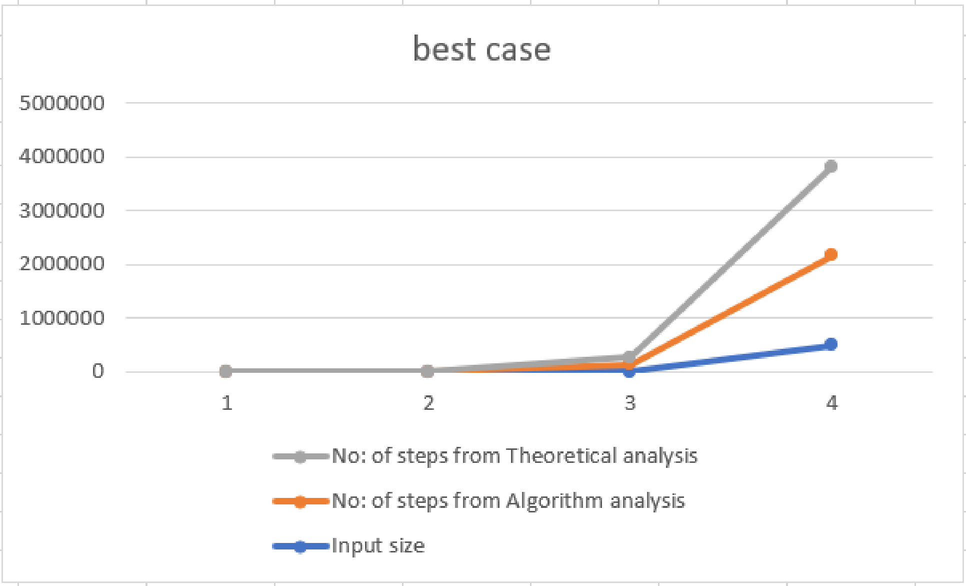
GRAPH:



Best Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1	100	664	654
2	500	4468	4462
3	10000	133516	132787
4	500000	1668822	1660854

GRAPH:



Conclusion: (Based on the observations): learned to implement and analyse time complexity of Quick-sort and Merge sort and compare both.

Outcome: CO1: Analyze time and space complexity of basic algorithms.

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.