Experiment No. : 2

Title: Divide and Conquer Strategy

Batch: A2 **Roll No.: 16010421059** **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:

Quick Sort:

Quick Sort Pivot Algorithm

Step 1 − Choose the highest index value has pivot
Step 2 − Take two variables to point left and right of the list excluding pivot
Step 3 − left points to the low index
Step 4 − right points to the high
Step 5 − while value at left is less than pivot move right
Step 6 − while value at right is greater than pivot move left
Step 7 − if both step 5 and step 6 does not match swap left and right
Step 8 − if left ≥ right, the point where they met is new pivot

Quick Sort Algorithm:

Step 1 − Make the right-most index value pivot
Step 2 − partition the array using pivot value
Step 3 − quicksort left partition recursively
Step 4 − quicksort right partition recursively

Merge Sort:

1. Create a function called merge_sort that accepts a list of integers as its argument. All following instructions presented are within this function.
2. Start by dividing the list into halves. Record the initial length of the list.
3. Check that the recorded length is equal to 1. If the condition evaluates to true, return the list as this means that there is just one element within the list. Therefore, there is no requirement to divide the list.
4. Obtain the midpoint for a list with a number of elements greater than 1. When using the Python language, the // performs division with no remainder. It rounds the division result to the nearest whole number. This is also known as floor division.
5. Using the midpoint as a reference point, split the list into two halves. This is the divide aspect of the divide-and-conquer algorithm paradigm.
6. Recursion is leveraged at this step to facilitate the division of lists into halved components. The variables *'left half'* and *'right half'* are assigned to the invocation of the '*merge sort'* function, accepting the two halves of the initial list as parameters.
7. The '*merge_sort*' function returns the invocation of a function that merges two lists to return one combined, sorted list.

Algorithm of Quick sort & Merge sort:

Quick Sort:

```
QUICKSORT (array A, start, end)
{
    if (start < end)
    {
     p = partition(A, start, end)
    QUICKSORT (A, start, p - 1)
    QUICKSORT (A, p + 1, end)
    }
}

PARTITION (array A, start, end)
{
    pivot ? A[end]
    i ? start-1
    for j ? start to end -1 {
    do if (A[j] < pivot) {
    then i ? i + 1
    swap A[i] with A[j]
    }}
    swap A[i+1] with A[end]
    return i+1
}
```

Merge Sort:

1: start

step 2: declare array and left, right, mid variable

step 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)
```

step 4: Stop

Derivation of Analysis Quick sort & Merge sort:

Worst Case Analysis
Quick Sort:

- O(N^2)

- lets T(n) ne total time complexity for worst case
- n = total number of elements
-
- T(n) = T(n-1) + constant*n
- as we are dividing array into two parts one consist of single element and other of n-1
- and we will traverse individual array
-
- T(n) = T(n-2) + constant*(n-1) + constant*n = T(n-2) + 2*constant*n - constant
- T(n) = T(n-3) + 3*constant*n - 2*constant - constant
- T(n) = T(n-k) + k*constant*n - (k-1)*constant ..... - 2*constant - constant
-
- T(n) = T(n-k) + k*constant*n - constant*[(k-1) .... + 3 + 2 + 1]
- T(n) = T(n-k) + k*n*constant - constant*[k*(k-1)/2]
- put n=k
- T(n) = T(0) + constant*n*n - constant*[n*(n-1)/2]
- removing constant terms
- T(n) = n*n - n*(n-1)/2
- T(n) = O(n^2)

Best Case Analysis
Quick Sort:

- O(Nlog(N))

- Lets T(n) be the time complexity for best cases
- n = total number of elements

- then
- T(n) = 2*T(n/2) + constant*n
- 2*T(n/2) is because we are dividing array into two array of equal size
- constant*n is because we will be traversing elements of array in each level of tree
-
- therefore,
- T(n) = 2*T(n/2) + constant*n
- further we will devide arrai in to array of equalsize so
- T(n) = 2*(2*T(n/4) + constant*n/2) + constant*n == 4*T(n/4) + 2*constant*n
-
- for this we can say that
- T(n) = 2^k * T(n/(2^k)) + k*constant*n
- then n = 2^k
- k = log2(n)
-
- therefore,
- T(n) = n * T(1) + n*logn = O(n*log2(n))

Average Case Analysis Quick Sort:

- O(Nlog(N))

- lets T(n) be total time taken
-
- then for average we will consider random element as pivot
- lets index i be pivot
-
- then time complexity will be
- T(n) = T(i) + T(n-i)
-
- T(n) = 1/n *[\sum_{i=1}^{n-1} T(i)] + 1/n*[\sum_{i=1}^{n-1} T(n-i)]
-
-
- As [\sum_{i=1}^{n-1} T(i)] and [\sum_{i=1}^{n-1} T(n-i)] equal likely functions

- therefore
- T(n) = 2/n*[\sum_{i=1}^{n-1} T(i)]
-
- multiply both side by n
- n*T(n) = 2*[\sum_{i=1}^{n-1} T(i)] .....(1)
-
- put n = n-1
- (n-1)*T(n-1) = 2*[\sum_{i=1}^{n-2} T(i)] .....(2)
-
- substract 1 and 2
- then we will get
- n*T(n) - (n-1)*T(n-1) = 2*T(n-1) + c*n^2 + c*(n-1)^2
- n*T(n) = T(n-1)[2+n-1] + 2*c*n - c
- n*T(n) = T(n-1)*(n+1) + 2*c*n [removed c as it was constant]
-
- divide both side by n*(n+1),
- T(n)/(n+1) = T(n-1)/n + 2*c/(n+1) .....(3)
-
- put n = n-1,
- T(n-1)/n = T(n-2)/(n-1) + 2*c/n .....(4)
-
- put n = n-2,
- T(n-2)/n = T(n-3)/(n-2) + 2*c/(n-1) .....(5)
-
- by putting 4 in 3 and then 3 in 2 we will get
- T(n)/(n+1) = T(n-2)/(n-1) + 2*c/(n-1) + 2*c/n + 2*c/(n+1)
-
- also we can find equation for T(n-2) by putting n = n-2 in (3)
-
- at last we will get
-
- T(n)/(n+1) = T(1)/2 + 2*c * [1/(n-1) + 1/n + 1/(n+1) + ...]
-
- T(n)/(n+1) = T(1)/2 + 2*c*log(n) + C
-
- T(n) = 2*c*log(n) * (n+1)
-
- now by removing constants,

-
- T(n) = log(n)*(n+1)
-
- therefore,
-
- T(n) = O(n*log(n))

Program(s) of Quick sort & Merge sort:
Quick Sort:

```
#include <stdio.h>
#include<math.h>
void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}
int partition(int arr[], int low, int high) {
  int pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j < high; 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) {
  if (low < high) {

    int i = partition(arr, low, high);
    quickSort(arr, low, i - 1);
    quickSort(arr, i + 1, high);
  }
}

void user(){
    int i,j,n,a[1000];
    printf("No of elements: ");
scanf("%d",&n);
printf("Elements:\n ");
    for(i=0;i<=n-1;i++){
        scanf("%d", &a[i]);
    }
  quickSort(a, 0, n - 1);
  printf("Sorted array:\n");
  {
```

```
  for (int j = 0; j <= n-1; j++) {
    printf("%d ", a[j]);
  }
}
}
void R(int n)
{

    printf("Time complexity of the best case : %f \n",n*log2(n));
    printf("Time complexity of the worst case : %d \n",n);
    printf("Time complexity of the average case : %f \n",n*log2(n));
}


int main() {
int p;
printf("1.User Input\n2.Random value\nEnter your choice: ");
scanf("%d",&p);
switch (p){
case 1:
    user();
    break;

case 2:
    int n;
        printf("Enter the size: ");
        scanf("%d",&n);
```

```
        R(n);

        break;

default:

   printf("Invalid input!");

}

}
```

Merge Sort:

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

void merge(int arr[], int l, int m, int r){
int i,j;

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

int R[n2],L[n1];

for (i = 0; i < n1; i++){
      L[i] = arr[l + i];
}
   for (j = 0; j < n2; j++){
      R[j] = arr[m + 1 + j];
}

int k = l;
i=0;
j=0;
while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
         arr[k] = L[i];
         i++;
      }
      else {
         arr[k] = R[j];
         j++;
      }
      k++;
   }

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

```
while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

}

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 user(){
    int a[1000],i,n,l,r;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    printf("Elements: ");
    for(i=0;i<=n-1;i++){
        scanf("%d",&a[i]);
    }
    l=0;
    r=n-1;
    mergeSort(a,l,r);
    printf("Sorted: ");
    for(i=0;i<=n-1;i++){
        printf(" %d",a[i]);
    }
}
void R(int n)
{

printf("Time complexity of the best case : %f \n",n*log2(n));
printf("Time complexity of the worst case : %d \n",n);
printf("Time complexity of the average case : %f \n",n*log2(n));
}

int main() {
int c;
printf("1.User input\n2.Random number\nEnter your choice: ");
scanf("%d",&c);
```

```
switch(c){
   case 1:
      user();
      break;
   case 2:
    int n;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    R(n);
    break;

   default:
   printf("Invalid input!");
}
}
```

Output(o) of Quick sort & Merge sort:

Quick Sort:

```
1.User Input
2.Random value
Enter your choice: 1
No of elements: 7
Elements:
 12 32 3 1 65 4 0
 Sorted array:
0 1 3 4 12 32 65
```


```
1.User Input
2.Random value
Enter your choice: 2
Enter the size: 1000
Time complexity of the best case : 9965.784285
Time complexity of the worst case : 1000
Time complexity of the average case : 9965.784285
```

```
1.User Input
2.Random value
Enter your choice: 2
Enter the size: 10000
Time complexity of the best case : 132877.123795
Time complexity of the worst case : 10000
Time complexity of the average case : 132877.123795
```

```
1.User Input
2.Random value
Enter your choice: 2
Enter the size: 20000
Time complexity of the best case : 285754.247591
Time complexity of the worst case : 20000
Time complexity of the average case : 285754.247591
```

Merge Sort:

```
Output

/tmp/2kExscFCVF.o
1.User input
2.Random number
Enter your choice: 1
Enter the number of elements: 5
Elements: 1 34 32 21 0
Sorted:  0 1 21 32 34
```

```
Output

/tmp/2kExscFCVF.o
1.User input
2.Random number
Enter your choice: 2
Enter the number of elements: 1000
Time complexity of the best case : 9965.784285
Time complexity of the worst case : 1000
Time complexity of the average case : 9965.784285
```

```
Output

/tmp/2kExscFCVF.o
1.User input
2.Random number
Enter your choice: 2
Enter the number of elements: 10000
Time complexity of the best case : 132877.123795
Time complexity of the worst case : 10000
Time complexity of the average case : 132877.123795
```

```
Output

/tmp/2kExscFCVF.o
1.User input
2.Random number
Enter your choice: 2
Enter the number of elements: 20000
Time complexity of the best case : 285754.247591
Time complexity of the worst case : 20000
Time complexity of the average case : 285754.247591
```

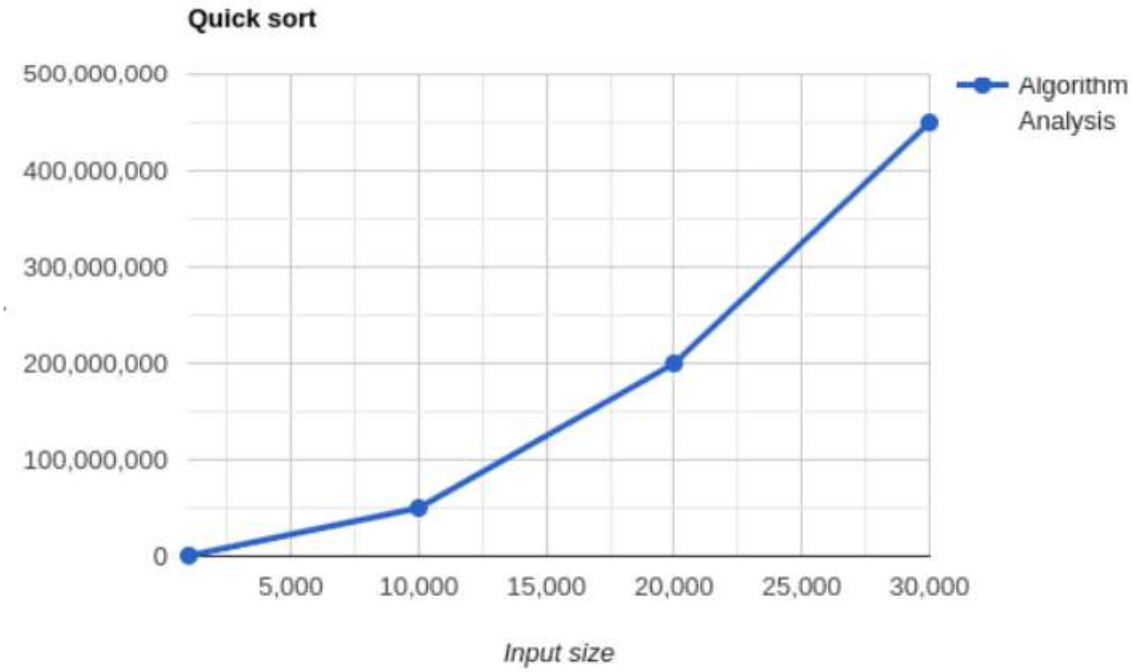
Results:

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	1000	499500	6907
2	10000	49995000	92103
3	20000	1999900000	198069

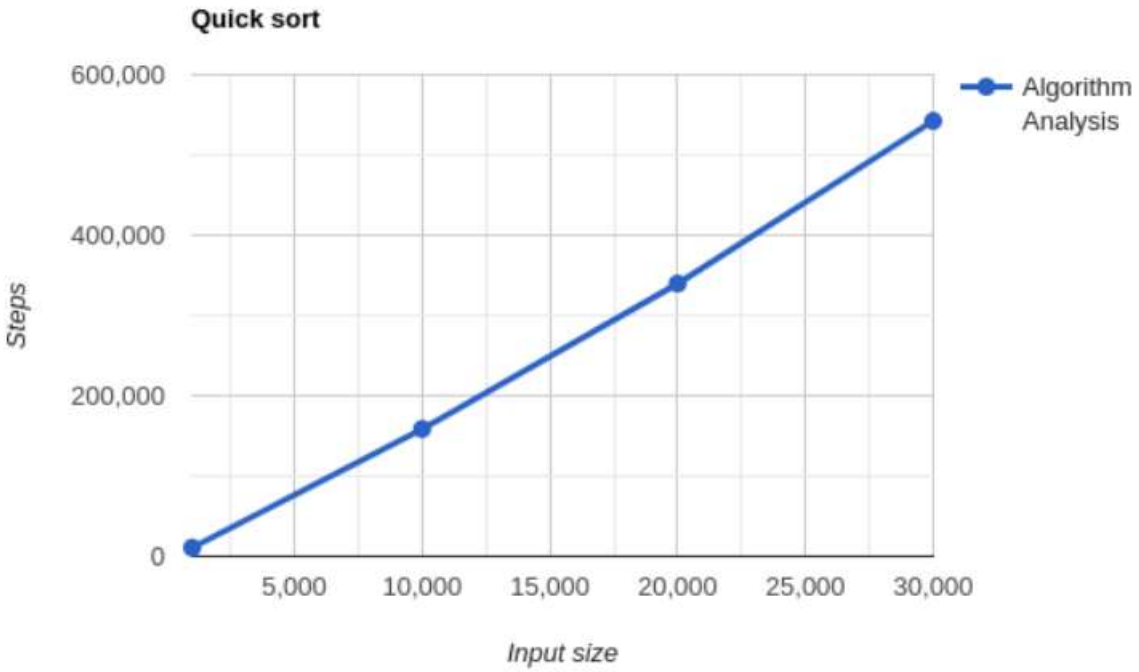
GRAPH:



Best Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1	1000	10676	6907
2	10000	158562	92103
3	20000	339756	198069

GRAPH

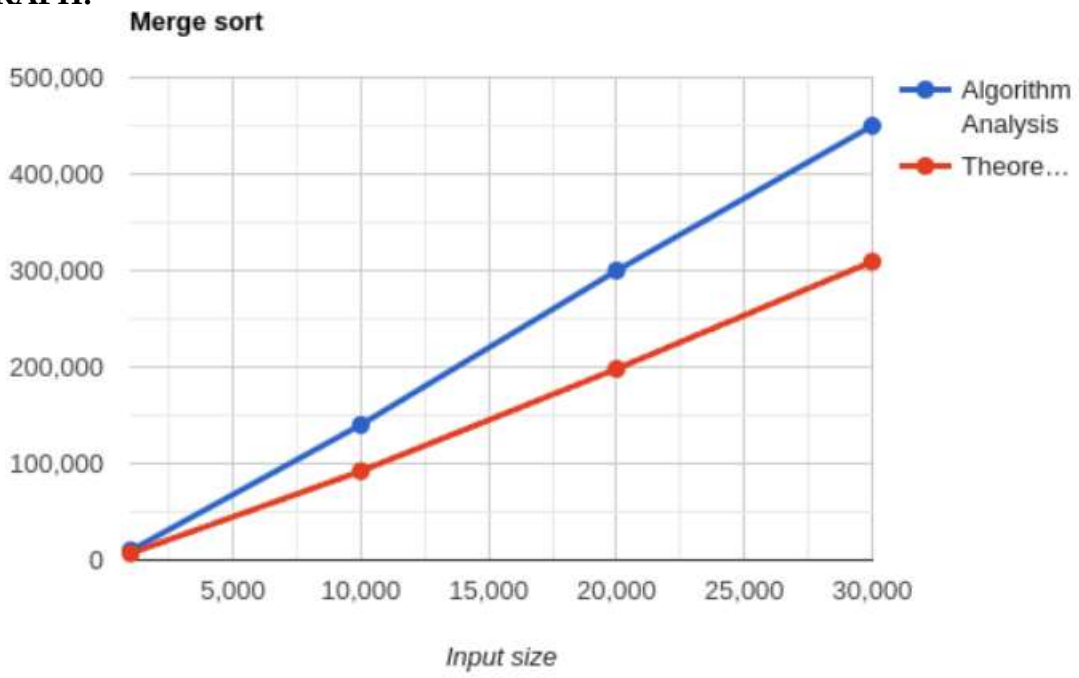

Quick sort
600,000
400,000
200,000
0
Steps
5,000
10,000
15,000
20,000
25,000
30,000
Input size
Algorithm Analysis

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	1000	10000	6907
2	10000	140000	92103
3	20000	300000	198069

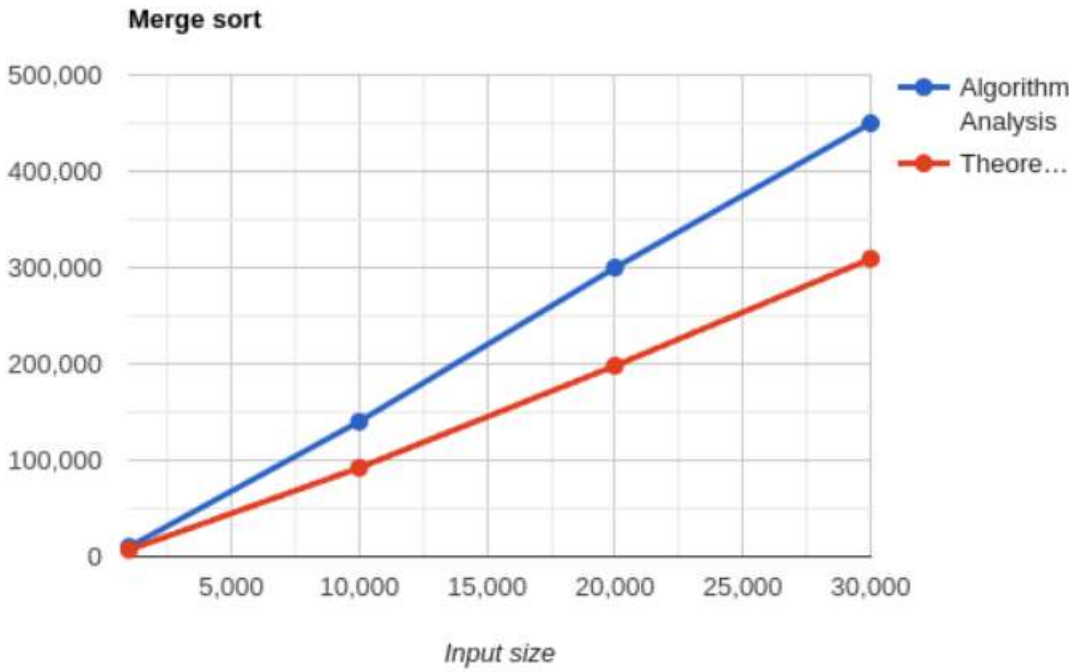
GRAPH:



Best Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1	1000	10000	6907
2	10000	140000	92103
3	20000	300000	198069

GRAPH



Conclusion: (Based on the observations):

The detailed analysis of the time complexity of quick sort and merge sort was done

Outcome:

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.