Ayush Goyal

190905522 CSE D 62

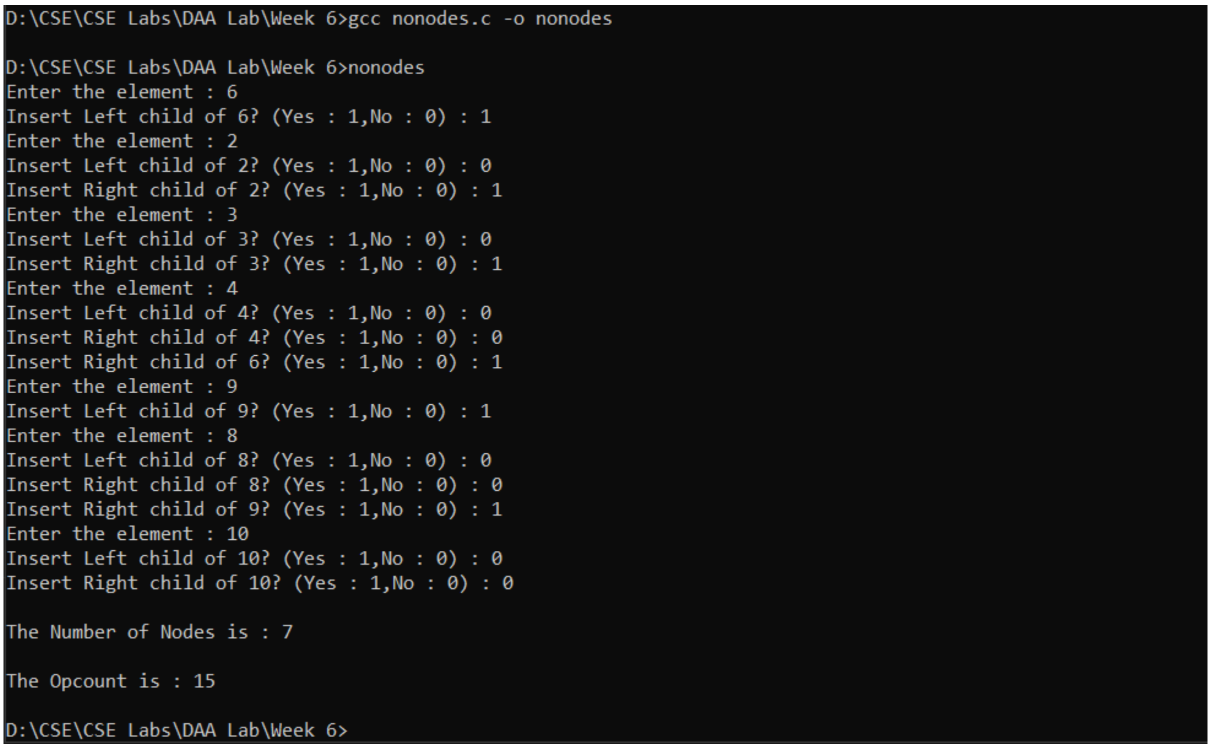
DAA Lab-6 (Week 6) – Divide and Conquer

Q1) Find total number of nodes in a binary tree and analyse its efficiency. Obtain the experimental result of order of growth and plot the result.

CODE:

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

int opcount=0;

typedef struct node{
    int val;
    struct node *left;
    struct node *right;
} *NODE;

NODE insertNode()
{
    int val;
    int check;
    printf("Enter the element : ");
    scanf("%d", &val);
    NODE cur = (NODE)malloc(sizeof(struct node));
    cur->val = val;
    cur->left = NULL;
    cur->right = NULL;
    printf("Insert Left child of %d? (Yes : 1,No : 0) : ", val);
    scanf("%d", &check);
    if (check)
        cur->left = insertNode();
    printf("Insert Right child of %d? (Yes : 1,No : 0) : ", val);
    scanf("%d", &check);
    if (check)
        cur->right = insertNode();
    return cur;
}

int countnodes(NODE header){
    opcount++;
    if (!header)
    {
        return 0;
    }
    return 1 + countnodes(header->right) + countnodes(header->left);
}

int main()
{
    NODE header = insertNode();
    int num = countnodes(header);
    printf("\nThe Number of Nodes is : %d\n", num);
    printf("\nThe Opcount is : %d\n",opcount);
    return 0;
}
```

OUTPUT:

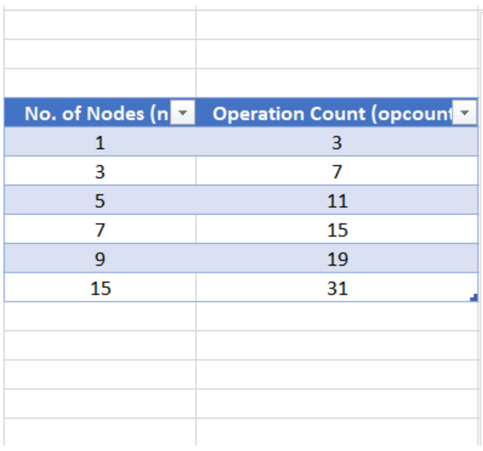


```
D:\CSE\CSE Labs\DAA Lab\Week 6>gcc nonodes.c -o nonodes

D:\CSE\CSE Labs\DAA Lab\Week 6>nonodes
Enter the element : 6
Insert Left child of 6? (Yes : 1,No : 0) : 1
Enter the element : 2
Insert Left child of 2? (Yes : 1,No : 0) : 0
Insert Right child of 2? (Yes : 1,No : 0) : 1
Enter the element : 3
Insert Left child of 3? (Yes : 1,No : 0) : 0
Insert Right child of 3? (Yes : 1,No : 0) : 1
Enter the element : 4
Insert Left child of 4? (Yes : 1,No : 0) : 0
Insert Right child of 4? (Yes : 1,No : 0) : 0
Insert Right child of 6? (Yes : 1,No : 0) : 1
Enter the element : 9
Insert Left child of 9? (Yes : 1,No : 0) : 1
Enter the element : 8
Insert Left child of 8? (Yes : 1,No : 0) : 0
Insert Right child of 8? (Yes : 1,No : 0) : 0
Insert Right child of 9? (Yes : 1,No : 0) : 1
Enter the element : 10
Insert Left child of 10? (Yes : 1,No : 0) : 0
Insert Right child of 10? (Yes : 1,No : 0) : 0

The Number of Nodes is : 7

The Opcount is : 15

D:\CSE\CSE Labs\DAA Lab\Week 6>
```

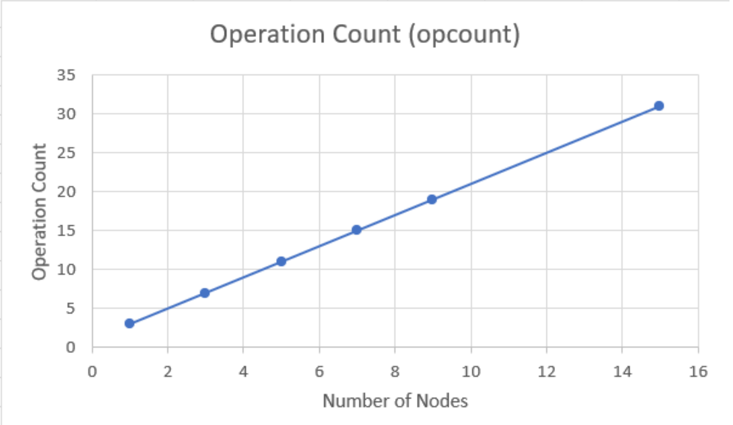
Time Efficiency Analysis:

The time efficiency for this algorithm is of the order of growth of **O(n).**
The basic operation is to check if the node is null or not.
We can see that the operation count is dependent on the number of nodes ‘n’ and it is related by (2*n)+1 and therefore it belongs to O(n) which is a liner time complexity.

Table and Plotted Graph:



No. of Nodes (n	Operation Count (opcount
1	3
3	7
5	11
7	15
9	19
15	31



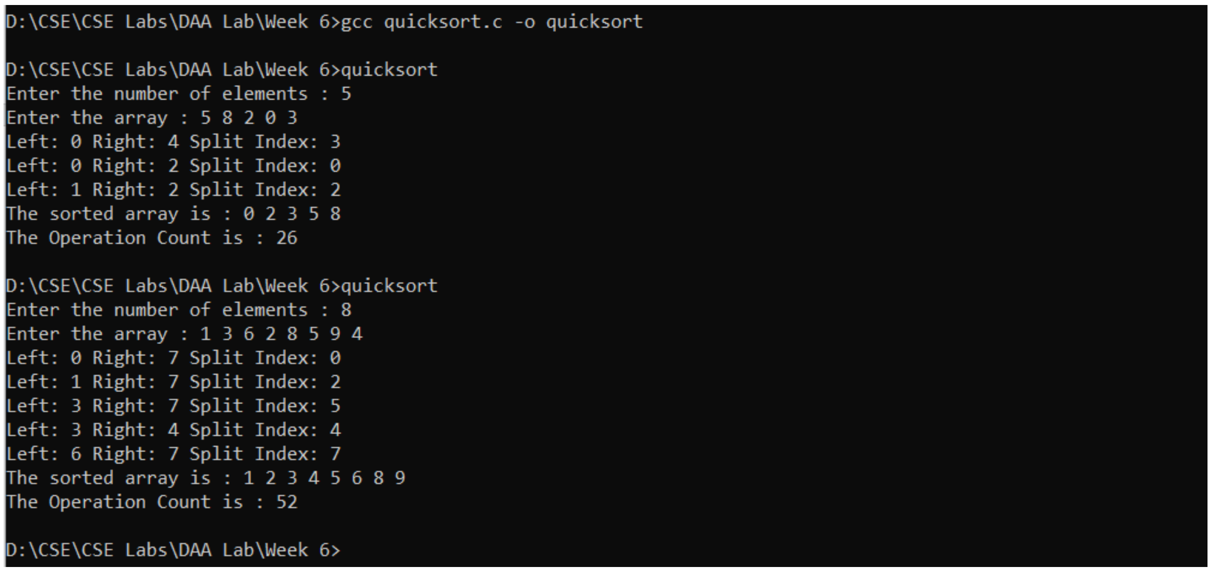
Q2) Sort given set of integers using Quick sort and analyse its efficiency. Obtain the experimental result of order of growth and plot the result.

CODE:

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

int opcount=0;

void swap(int *arr, int i, int j){
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}


int partition(int *arr, int left, int right){
    int start = left + 1;
    int pivot = left;
    int end = right;
    while(1){
        while (arr[start] <= arr[pivot]){
            opcount++;
            start++;
        }
        opcount++; //for while
        while (arr[end] > arr[pivot]){
```

```c
            opcount++;
            end--;
        }
        opcount++; //for while
        opcount++; //for if statement comp
        if (start >= end){
            break;
        }
        swap(arr, start, end);
    }
    swap(arr, pivot, end);
    return end;
}


void quicksort(int *arr, int left, int right){
    opcount++;
    if (left < right){
        int split = partition(arr, left, right);
        printf("Left: %d Right: %d Split Index: %d\n", left, right, split);
        quicksort(arr, left, split - 1);
        quicksort(arr, split + 1, right);
    }
}


int main()
{
    printf("Enter the number of elements : ");
    int n;
    scanf("%d", &n);
    printf("Enter the array : ");
    int *arr = (int *)calloc(n, sizeof(int));
    for (int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    quicksort(arr, 0, n - 1);
    printf("The sorted array is : ");
    for (int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\nThe Operation Count is : %d",opcount);
    printf("\n");
    return 0;
}
```

OUTPUT:

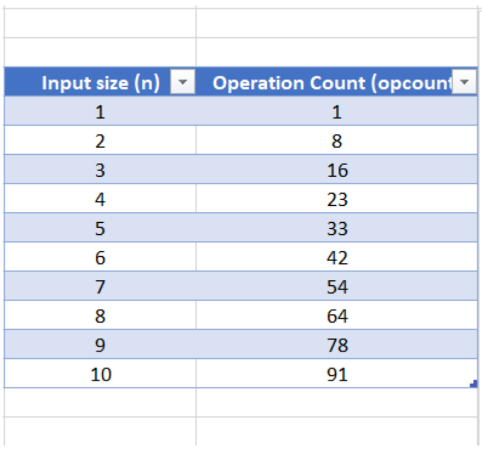


```
D:\CSE\CSE Labs\DAA Lab\Week 6>gcc quicksort.c -o quicksort

D:\CSE\CSE Labs\DAA Lab\Week 6>quicksort
Enter the number of elements : 5
Enter the array : 5 8 2 0 3
Left: 0 Right: 4 Split Index: 3
Left: 0 Right: 2 Split Index: 0
Left: 1 Right: 2 Split Index: 2
The sorted array is : 0 2 3 5 8
The Operation Count is : 26

D:\CSE\CSE Labs\DAA Lab\Week 6>quicksort
Enter the number of elements : 8
Enter the array : 1 3 6 2 8 5 9 4
Left: 0 Right: 7 Split Index: 0
Left: 1 Right: 7 Split Index: 2
Left: 3 Right: 7 Split Index: 5
Left: 3 Right: 4 Split Index: 4
Left: 6 Right: 7 Split Index: 7
The sorted array is : 1 2 3 4 5 6 8 9
The Operation Count is : 52

D:\CSE\CSE Labs\DAA Lab\Week 6>
```

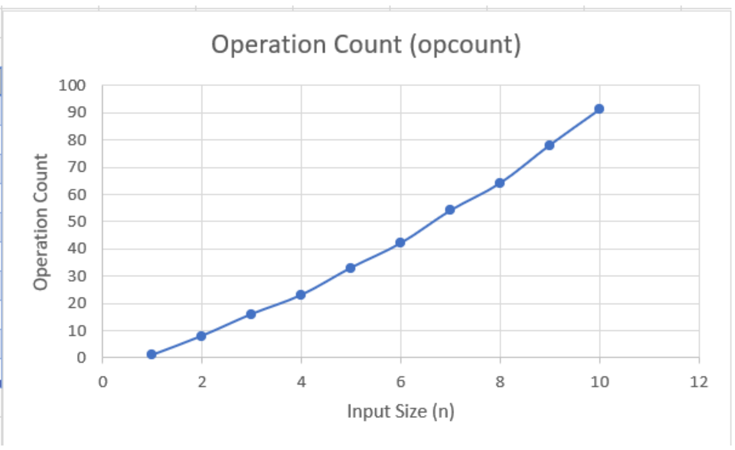
Time Efficiency Analysis:

Let array size be input which is equal to n,
The basic operation is to compare and check for values of start and end so as to get the split point, then analysis is as follows:
For best case,
On every iteration for all sub arrays the split point occurs at the middle position.
The recurrence relation is given by :
$C_{best}(n) = 2\,C_{best}(n/2) + n$ for n > 1
$C_{best}(1) = 0$
Thus according to Master's Theorem ,
$C_{best}(n)$ belongs to $\Theta(n \log_2 n)$
For worst case,
If the input array is already in sorted order it is the worst case.
Always the left subarray is left empty and size is one less than the size of the array.
$C_{worst}(n) = n+1 + n + n-1 + \ldots.. + 3$
Thus, **$C_{worst}(n)$ belongs to $\Theta(n_2)$.**
For average case,
We assume that split point can occur in each position (0<=s<=n-1) with same problem 1/n.
We write the recurrence relation:
$C_{avg}(n) = 1/n\ \Sigma[\ (n+1)\ {}_{n-1s=0} + Cavg\ (s) + Cavg\ (n-1-s)]$ for n > 1.
$C_{avg}(0) = 0$, $C_{avg}(1) = 0$
Thus, **$C_{avg}(n) \approx 2n \ln n \approx 1.38\, n \log_2 n$.**

Table and Plotted Graph:



Input size (n)	Operation Count (opcount
1	1
2	8
3	16
4	23
5	33
6	42
7	54
8	64
9	78
10	91

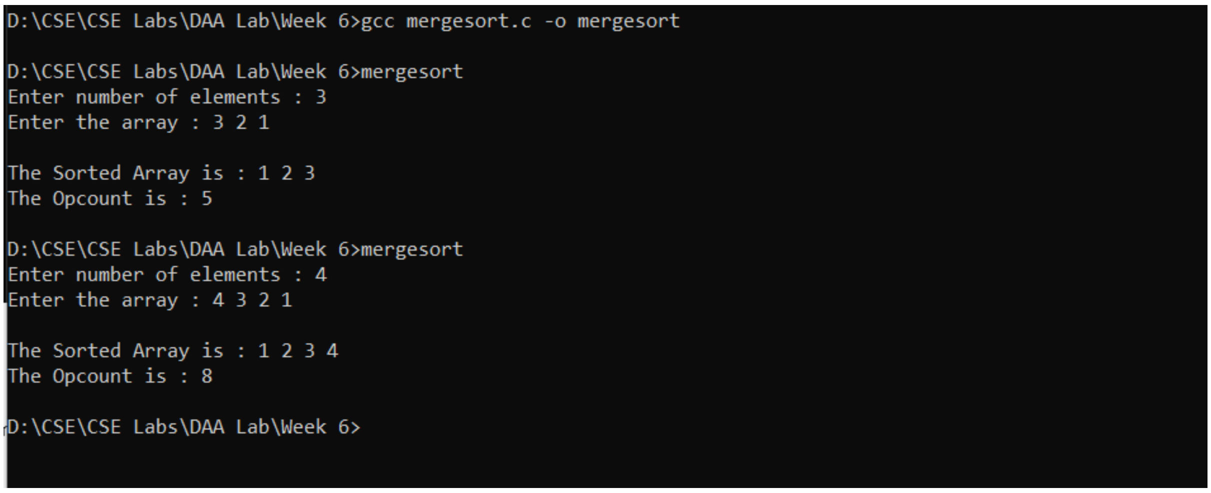


3) Sort given set of integers using Merge sort and analyse its efficiency. Obtain the experimental result of order of growth and plot the result.

CODE:

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

int opcount = 0;

void merge(int *arr, int start, int end, int mid)
{
    int an = mid - start + 1;
    int bn = end - mid;
    int a[an], b[bn];
    for (int i = 0; i < an; i++){
        a[i] = arr[i + start];
    }
    for (int i = 0; i < bn; i++){
        b[i] = arr[i + mid + 1];
    }
    int i = 0, j = 0, k = start;
    while (i < an && j < bn){
        opcount++;
        if (a[i] < b[j]){
            arr[k] = a[i];
            i++;
        }
        else{
            arr[k] = b[j];
            j++;
        }
        k++;
    }
    while (i < an){
        opcount++;
        arr[k] = a[i];
        i++;
        k++;
    }
    while (j < bn){
        opcount++;
        arr[k] = b[j];
        j++;
        k++;
    }
}
void mergesort(int *arr, int start, int end){
    if (start < end){
        int mid = start + (end - start) / 2;
        mergesort(arr, start, mid);
        mergesort(arr, mid + 1, end);
        merge(arr, start, end, mid);
    }
}

int main(){
    printf("Enter number of elements : ");
    int n;
    scanf("%d", &n);
    printf("Enter the array : ");
    int *arr = (int *)calloc(n, sizeof(int));
    for (int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    printf("\n");
    mergesort(arr, 0, n - 1);
    printf("The Sorted Array is : ");
    for (int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\nThe Opcount is : %d\n",opcount);
    return 0;
}
```

OUTPUT:

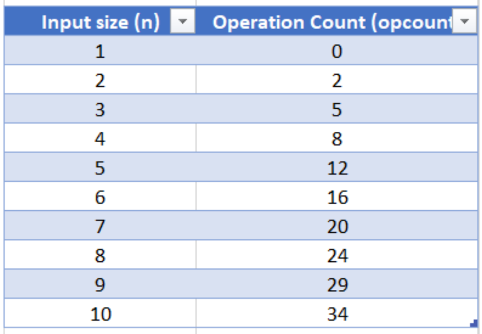


```
D:\CSE\CSE Labs\DAA Lab\Week 6>gcc mergesort.c -o mergesort

D:\CSE\CSE Labs\DAA Lab\Week 6>mergesort
Enter number of elements : 3
Enter the array : 3 2 1

The Sorted Array is : 1 2 3
The Opcount is : 5

D:\CSE\CSE Labs\DAA Lab\Week 6>mergesort
Enter number of elements : 4
Enter the array : 4 3 2 1

The Sorted Array is : 1 2 3 4
The Opcount is : 8

D:\CSE\CSE Labs\DAA Lab\Week 6>
```

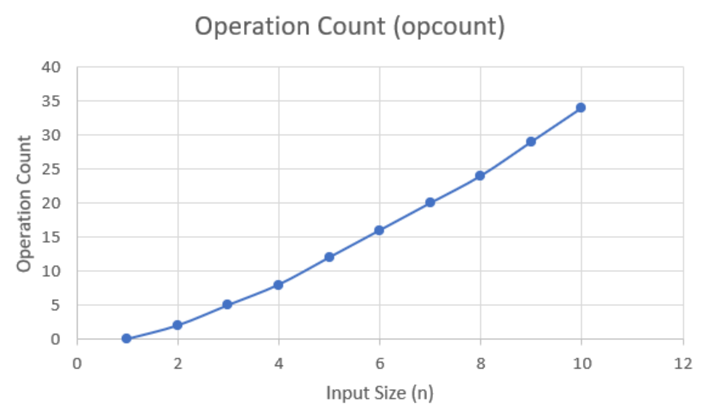
Time Efficiency Analysis:

We have as input an array of size n.
The basic operation is to compare the elements in both the subarrays and to merge them.
$C(n) = 2C(n/2) + C_{merge}(n)$ for $n > 1$
$C(1) = 0$
Where $C_{merge}(n)$ is the number of comparisons performed during the merging stage.
In worst case,
$C_{merge}(n) = n-1$
$C_{worst}(n) = 2C_{worst}(n/2) + n-1$ for $n > 1$
$C_{worst}(1) = 0$
By Master's Theorem we get that,
$C_{worst}(n)$ belongs to $\Theta(n \log n)$.

Table and Plotted Graph:



Input size (n)	Operation Count (opcount
1	0
2	2
3	5
4	8
5	12
6	16
7	20
8	24
9	29
10	34



THE END