Lab 6 : Dipesh Singh – 190905520

Question 1 : Find total number of nodes in a binary tree and analyze its efficiency. Obtain the experimental result of order of growth and plot the result.

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

    int val;

    struct node \*left;

    struct node \*right;

} \* Node;

void inorder(Node n)

{

    if (n)

    {

        inorder(n->left);

        printf("%d ", n->val);

        inorder(n->right);

    }

}

Node insert()

{

    int val;

    int check;

    printf("Enter the element : ");

    scanf("%d", &val);

    Node n = (Node)malloc(sizeof(struct node));

    n->val = val;

    n->left = NULL;

    n->right = NULL;

    printf("Do you want to insert left child of %d? (1 for Yes, 0 for No) : ", val);

    scanf("%d", &check);

    if (check)

        n->left = insert();

    printf("Do you want to insert right child of %d? (1 for Yes, 0 for No) : ", val);

    scanf("%d", &check);

    if (check)

        n->right = insert();

    return n;

}

int cnt(Node head, int \*opcount)

{

    (\*opcount)++;

    if (head)

    {

        return 1 + cnt(head->right, opcount) + cnt(head->left, opcount);

    }

    return 0;

}

int main()

{

    Node head = insert();

    printf("The inorder is : ");

    inorder(head);

    int opcount = 0;

    int res = cnt(head, &opcount);

    printf("\nThe number of nodes is : %d and the number of operations is : %d\n", res, opcount);

    return 0;

}

Text

Description automatically generated

Table

Description automatically generated

Chart, line chart

Description automatically generated

As we can see from the plot, the runtime of this algorithm is of the order O(n) where n is the number of nodes in the tree.

Question 2 : Sort given set of integers using Quick sort and analyze its efficiency. Obtain the experimental result of order of growth and plot the result.

#include <stdio.h>

#include <stdlib.h>

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 \*opcount)

{

    int start = left + 1;

    int pivot = left;

    int end = right;

    for (;;)

    {

        while (arr[start] <= arr[pivot])

        {

            (\*opcount)++;

            start++;

        }

        (\*opcount)++;

        while (arr[end] > arr[pivot])

        {

            (\*opcount)++;

            end--;

        }

        (\*opcount)++;

        if (start >= end)

        {

            break;

        }

        swap(arr, start, end);

    }

    swap(arr, pivot, end);

    return end;

}

void quick(int \*arr, int left, int right, int \*opcount)

{

    if (left < right)

    {

        int part = partition(arr, left, right, opcount);

        quick(arr, left, part - 1, opcount);

        quick(arr, part + 1, right, opcount);

    }

}

int main()

{

    printf("Enter the number of elements in the array : ");

    int num;

    scanf("%d", &num);

    printf("Enter the array : ");

    int \*arr = (int \*)calloc(num, sizeof(int));

    for (int i = 0; i < num; i++)

    {

        scanf("%d", &arr[i]);

    }

    printf("Your array is : ");

    for (int i = 0; i < num; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

    int opcount = 0;

    quick(arr, 0, num - 1, &opcount);

    printf("Your sorted array is : ");

    for (int i = 0; i < num; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\nThe number of operations is  : %d\n", opcount);

    return 0;

}

A screenshot of a computer

Description automatically generated with medium confidence

Table

Description automatically generated

Chart, line chart

Description automatically generated

As we can see from the plot, the worst case runtime for quicksort is O(n2) and the average case runtime is O(nlogn).

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

#include <stdio.h>

#include <stdlib.h>

void merge(int \*arr, int start, int end, int mid, int \*opcount)

{

    int anum = mid - start + 1;

    int bnum = end - mid;

    int a[anum], b[bnum];

    for (int i = 0; i < anum; i++)

    {

        a[i] = arr[i + start];

    }

    for (int i = 0; i < bnum; i++)

    {

        b[i] = arr[i + mid + 1];

    }

    int i = 0, j = 0, k = start;

    while (i < anum && j < bnum)

    {

        (\*opcount)++;

        if (a[i] < b[j])

        {

            arr[k] = a[i];

            i++;

        }

        else

        {

            arr[k] = b[j];

            j++;

        }

        k++;

    }

    while (i < anum)

    {

        arr[k] = a[i];

        i++;

        k++;

    }

    while (j < bnum)

    {

        arr[k] = b[j];

        j++;

        k++;

    }

}

void mergeSort(int \*arr, int start, int end, int \*opcount)

{

    if (start < end)

    {

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

        mergeSort(arr, start, mid, opcount);

        mergeSort(arr, mid + 1, end, opcount);

        merge(arr, start, end, mid, opcount);

    }

}

int main()

{

    printf("Enter the number of elements in the array : ");

    int num;

    scanf("%d", &num);

    printf("Enter the array : ");

    int \*arr = (int \*)calloc(num, sizeof(int));

    for (int i = 0; i < num; i++)

    {

        scanf("%d", &arr[i]);

    }

    printf("Your array is : ");

    for (int i = 0; i < num; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

    int opcount = 0;

    mergeSort(arr, 0, num - 1, &opcount);

    printf("Your sorted array is : ");

    for (int i = 0; i < num; i++)

    {

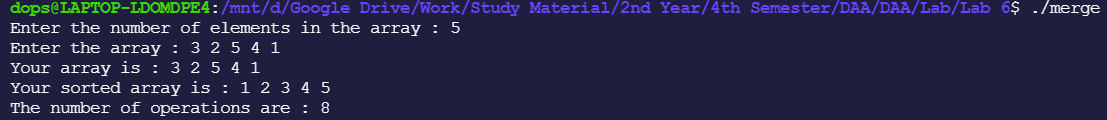
        printf("%d ", arr[i]);

    }

    printf("\nThe number of operations are : %d\n", opcount);

    return 0;

}



Table

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

As we can see from the plot, the runtime for the mergesort algorithm in the worst case is O(nlogn).