LAB 6

Aniket sambher

Reg no-190905466

Section -A

Roll no-58

1.Nodes in binary tree

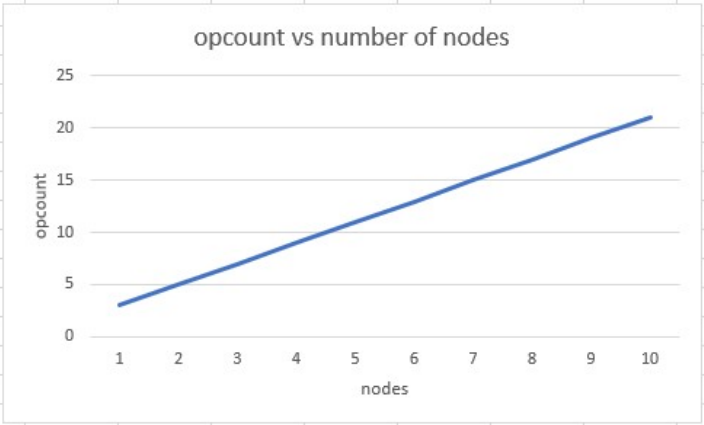
```
#include<stdio.h>
#include <stdlib.h>
int opcount=0;
typedef struct node *Nodeptr;
struct node{
int data;
Nodeptr rchild;
Nodeptr lchild;
}NODE;

int number_of_nodes(Nodeptr root){
   opcount++;
   if(root)
      return 1+ number_of_nodes(root->lchild)+number_of_nodes(root->rchild);
   return 0;
}
Nodeptr getnode(){


   return ((Nodeptr)malloc (sizeof(NODE)));

}
```

```
Nodeptr CreateBinaryTree(int item){
int x;
if (item!=-1) { //until input is not equal to -1

Nodeptr temp=getnode() ;
temp->data = item;
printf("Enter the lchild of %d :",item);
scanf("%d",&x);
temp->lchild = CreateBinaryTree(x);
printf("Enter the rchild of %d :",item);
scanf("%d",&x);
temp->rchild = CreateBinaryTree(x);
return temp;
}
return NULL;
}

int main()
{
Nodeptr root = NULL;
int item;
printf("Creating the tree : \n");
printf("Enter the root :");
scanf("%d",&item);
root=CreateBinaryTree(item);

int nodes=number_of_nodes(root);

printf("number of nodes=%d",nodes);
printf("number of operations=%d",opcount);
```

}

```
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6\" ; if ($?) { gcc binarytree
.c -o binarytree } ; if ($?) { .\binarytree }
Creating the tree :
Enter the root :2
Enter the lchild of 2 :1
Enter the lchild of 1 :-1
Enter the rchild of 1 :-1
Enter the rchild of 2 :-1
number of nodes=2number of operations=5
```

number of nodes	opcount
1	3
2	5
3	7
4	9
5	11
6	13
7	15
8	17
9	19
10	21



TIME COMPLEXITY ANALYSIS

The algorithm visits every nodes including nodes pointing to NULL values thereby for n nodes the algorithm works for 2n+1 iterations(opcount) because n nodes with have n+1 null nodes

Thereby time complexity = O(2n+1)=O(n)

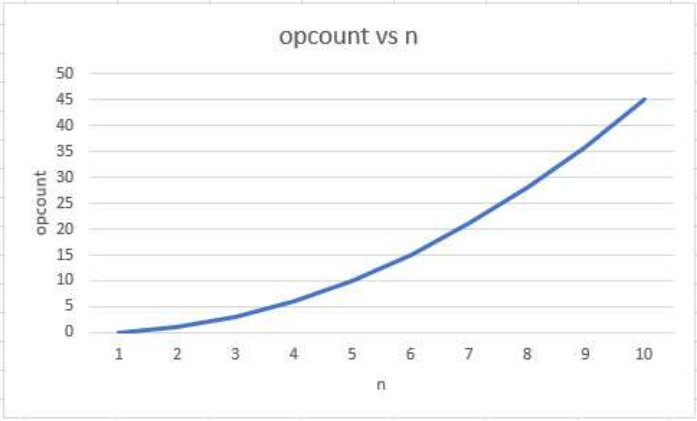
2.quick sort

```
#include<stdio.h>
#include <stdlib.h>
int opcount=0;
void swap(int arr[],int i,int j){
   int temp;
   temp=arr[i];
   arr[i]=arr[j];
   arr[j]=temp;
}
int partition(int arr[],int start,int end){
   int pivot=arr[end];
   int i=start;
   for(int j=start;j<end;++j){
      opcount++;
      if(arr[j]<=pivot){

         swap(arr,i,j);
         i++;
      }
   }
      swap(arr,i,end);
      return i;
}



void quicksort(int arr[],int start ,int end){
   if(start<end){
   int p=partition(arr,start,end);
   quicksort(arr,start,p-1);
```

```
    quicksort(arr,p+1,end);
    }
}

void main(){
    //int arr[5]={1,2,3,4,-1};
    int n;
    printf("enter the number of elements");
    scanf("%d",&n);
    int arr[n];
    for (int i = 0; i < n; ++i)
                        scanf(" %d", &arr[i]);


    quicksort(arr,0,n-1);
    for(int i=0;i<n;++i)
       printf("%d   ",arr[i]);
    printf("opcount=%d",opcount);

}
```

```
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6\" ; if ($?) { gcc quicksort.
c -o quicksort } ; if ($?) { .\quicksort }
enter the number of elements5
7 6 5 4 3
3   4   5   6   7   opcount=10
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6>
```

n	Quicksort
1	0
2	1
3	3
4	6
5	10
6	15
7	21
8	28
9	36
10	45



Date

Analysis of Quicksort

$T(n) = T(k) + T(n-k-1) + f(n)$

where k number of elements smaller than pivot

Worst Case

Occurs when we pick the greatest or smallest as pivot

$T(n) = T(0) + T(n-1) + f(n)$

$= T(n-1) + f(n)$

$\approx \frac{n(n+1)}{2} \approx O(n^2)$

Best Case

We we partition take middle elem

$T(n) = 2T(n/2) + f(n)$

$= n \log n$ (Master's Theorem)

Averge Case

We take the averge case to be $O(n \log n)$

3.mergesort

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

int opcount=0;
void merge(int arr[], int l, int m, int r)
{
   int i, j, k;
   int n1 = m - l + 1;
   int n2 =  r - m;

   /* create temp arrays */
   int L[n1], R[n2];

   /* Copy data to temp arrays L[] and R[] */
   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1+ j];

   /* Merge the temp arrays back into arr[l..r]*/
   i = 0; // Initial index of first subarray
   j = 0; // Initial index of second subarray
   k = l; // Initial index of merged subarray
   while (i < n1 && j < n2)
   {   opcount++;
      if (L[i] <= R[j])
      {
         arr[k] = L[i];
         i++;
      }
```

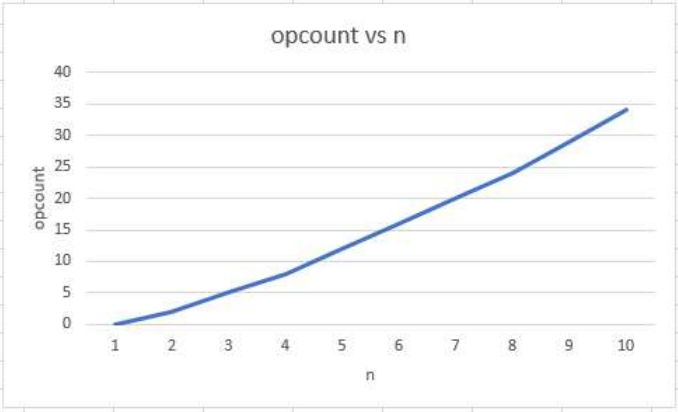
```
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)

    { opcount++;
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2)
    { opcount++;
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
```

```
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

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

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
void main(){
    //int arr[5]={1,2,3,4,-1};
    int n;
    printf("enter the number of elements");
    scanf("%d",&n);
```

```
    int arr[n];
    for (int i = 0; i < n; ++i)
                        scanf(" %d", &arr[i]);



    mergeSort(arr,0,n-1);
    for(int i=0;i<n;++i)
        printf("%d",arr[i]);
    printf("opcount=%d",opcount);


}
```

```
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6\" ; if ($?) { gcc merge.c -o
 merge } ; if ($?) { .\merge }
enter the number of elements5
6 5 4 3 2
23456opcount=12
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week6\" ; if ($?) { gcc tempCodeRu
nnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
enter the number of elements4
6 5 4 3
3456opcount=8
```

n	Mergesort
1	0
2	2
3	5
4	8
5	12
6	16
7	20
8	24
9	29
10	34



Page No.
Date

Analysis of Merge Sort

We always divide the array in two halves so we get the same best, average, worst case time complexity

$$T(n) = 2T(n/2) + f(n)$$

$$= n \log n \quad \text{(Master's theorem)}$$