2026

DESIGN AND ANALYSIS OF ALGORITHM

LAB EXPERIMENT - 5

KOLATHUR SURYA-[CH.SC.U4CSE24224]

1 . QUICK SORT

CODE:

```
  GNU nano 7.2
#include <stdio.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 pi=partition(arr,low,high);
        quickSort(arr,low,pi-1);
        quickSort(arr,pi+1,high);
    }
}

int main(){
    printf("Name: KOLATHUR SURYA\n");
    printf("Roll No: CH.SC.U4CSE24224\n");

    int n;
    printf("Enter number of elements: ");
    scanf("%d",&n);
```

```
    int arr[n];

    printf("Enter elements: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }

    quickSort(arr,0,n-1);

    printf("Sorted array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }

    return 0;
}
```

Output :

```
surya@DESKTOP-FJ2EKA4:~$ nano quick.c
surya@DESKTOP-FJ2EKA4:~$ gcc quick.c -o quick
.surya@DESKTOP-FJ2EKA4:~$ ./quick
Name: KOLATHUR SURYA
Roll No: CH.SC.U4CSE24224
Enter number of elements: 5
Enter elements: 4
2
5
1
3
Sorted array: 1 2 3 4 5 surya@DESKTOP-FJ2EKA4:~$
```

1. Quick sort is a divide-and-conquer sorting algorithm used to arrange elements in ascending or descending order.
2. It works by selecting a pivot element from the array.
3. The array is partitioned into two parts: elements smaller than the pivot and elements greater than the pivot.
4. The same process is recursively applied to the left and right subarrays.
5. It is an in-place sorting algorithm, so it requires less extra memory.
6. Average time complexity is O(n log n), but worst case is $O(n^2)$.
7. Quick sort is widely used because it is fast and efficient for large datasets.

2.merge sort

Code :

```
  GNU nano 7.2
#include <stdio.h>

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

    int L[n1],R[n2];

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

    int i=0,j=0,k=l;

    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);
    }
}

int main(){
    printf("Name: KOLATHUR SURYA\n");
    printf("Roll No: CH.SC.U4CSE24224\n");

    int n;
    printf("Enter number of elements: ");
    scanf("%d",&n);

    int arr[n];

    printf("Enter elements: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }

    mergeSort(arr,0,n-1);

    printf("Sorted array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }

    return 0;
}
```

Output :

```
surya@DESKTOP-FJ2EKA4:~$ nano merge.c
surya@DESKTOP-FJ2EKA4:~$
surya@DESKTOP-FJ2EKA4:~$ gcc merge.c -o merge
surya@DESKTOP-FJ2EKA4:~$ ./merge
Name: KOLATHUR SURYA
Roll No: CH.SC.U4CSE24224
Enter number of elements: 5
Enter elements: 6
7
4
9
1
Sorted array: 1 4 6 7 9 surya@DESKTOP-FJ2EKA4:~$
```

1. Merge sort is a divide-and-conquer sorting algorithm used to sort elements efficiently.
2. It divides the array into two halves repeatedly until each subarray has one element.
3. Then it merges the subarrays back together in sorted order.
4. The merging process compares elements and arranges them correctly.
5. It is not an in-place algorithm because it needs extra memory for merging.
6. Time complexity is O(n log n) in best, average, and worst cases.
7. Merge sort is stable and works well for large datasets and linked lists.

3. Binary search tree

Code :

```
  GNU nano 7.2
#include <stdio.h>
#include <stdlib.h>

struct node{
    int data;
    struct node *left,*right;
};

struct node* create(int val){
    struct node* temp=(struct node*)malloc(sizeof(struct node));
    temp->data=val;
    temp->left=temp->right=NULL;
    return temp;
}

struct node* insert(struct node* root,int val){
    if(root==NULL)
        return create(val);
    if(val<root->data)
        root->left=insert(root->left,val);
    else if(val>root->data)
        root->right=insert(root->right,val);
    return root;
}

void inorder(struct node* root){
    if(root!=NULL){
        inorder(root->left);
        printf("%d ",root->data);
        inorder(root->right);
    }
}

int main(){
    printf("Name: KOLATHUR SURYA\n");
    printf("Roll No: CH.SC.U4CSE24224\n");
```

```
    struct node* root=NULL;
    int n,val;

    printf("Enter number of elements: ");
    scanf("%d",&n);

    printf("Enter elements: ");
    for(int i=0;i<n;i++){
        scanf("%d",&val);
        root=insert(root,val);
    }

    printf("Inorder traversal: ");
    inorder(root);

    return 0;
}
```

Output:

```
surya@DESKTOP-FJ2EKA4:~$ nano bst.c
surya@DESKTOP-FJ2EKA4:~$ gcc bst.c -o bst
surya@DESKTOP-FJ2EKA4:~$ ./bst
Name: KOLATHUR SURYA
Roll No: CH.SC.U4CSE24224
Enter number of elements: 6
Enter elements: 3
2
1
7
4
2
Inorder traversal: 1 2 3 4 7 surya@DESKTOP-FJ2EKA4:~$
```

1. A Binary Search Tree (BST) is a data structure where each node has at most two children.
2. In BST, the left child contains values smaller than the parent node.
3. The right child contains values greater than the parent node.
4. It allows fast searching, insertion, and deletion of elements.
5. The average time complexity for operations is O(log n).
6. In the worst case (skewed tree), time complexity becomes O(n).
7. BST is widely used in searching, sorting, and maintaining ordered data.