C S Deeraj

CH.SC.U4CSE24106

Week 3

Date – 18/12/2025

Design and Analysis of Algorithms (23CSE211)

Merge and Quick Sort

Merge Sort Code

Mergesortweek3 - Notepad

File Edit Format View Help

```
//CH.SC.U4CSE24106
#include <stdio.h>

void merge(int a[],int l,int m,int r){
        int i=l,j=m+1,k=0;
        int temp[100];

        while(i<=m && j<=r){
                if(a[i]<=a[j])
                        temp[k++]=a[i++];
                else
                        temp[k++]=a[j++];
        }

        while(i<=m)
                temp[k++]=a[i++];

        while(j<=r)
                temp[k++]=a[j++];

        for(i=l,k=0;i<=r;i++,k++)
                a[i]=temp[k];
}

void mergesort(int a[],int l,int r){
        if(l<r){
                int m=(l+r)/2;
                mergesort(a,l,m);
                mergesort(a,m+1,r);
                merge(a,l,m,r);
        }
}

int main(){
        int a[100],n;

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

        printf("Enter The Roll Numbers:\n");
        for(int i=0;i<n;i++)
                scanf("%d",&a[i]);
```

```
        scanf("%d",&n);

        printf("Enter The Roll Numbers:\n");
        for(int i=0;i<n;i++)
                scanf("%d",&a[i]);

        mergesort(a,0,n-1);

        printf("Sorted using merge sort:\n");
        for(int i=0;i<n;i++)
                printf("%d ",a[i]);

        printf("\nCH.SC.U4CSE24106");

        return 0;
}
```

Output:

```
C:\Users\savit\OneDrive\Desktop\Sem IV CSE\Ws1 haskell>gcc Mergesortweek3.c -o Mergesortweek3

C:\Users\savit\OneDrive\Desktop\Sem IV CSE\Ws1 haskell>Mergesortweek3
Enter number of elements: 12
Enter The Roll Numbers:
157
110
147
122
111
149
151
141
123
112
117
133
Sorted using merge sort:
110 111 112 117 122 123 133 141 147 149 151 157
CH.SC.U4CSE24106
C:\Users\savit\OneDrive\Desktop\Sem IV CSE\Ws1 haskell>
```

Time Complexity of Merge Sort

Merge sort uses the divide and conquer approach. The array is repeatedly divided into two halves until subarrays of size one are obtained, which takes log n levels of recursion. At each level, all n elements are merged back in sorted order, and the merging process takes linear time. Since there are log n levels and each level requires O(n) time, the total time complexity of merge sort is O(n log n) in the best, average, and worst cases.

Space Complexity of Merge Sort

Merge sort requires additional memory to store temporary arrays during the merging process. The size of this auxiliary array is proportional to the number of elements, which requires O(n) extra space. In addition, recursive calls use stack space of O(log n), but this is dominated by the auxiliary array. Therefore, the overall space complexity of merge sort is O(n).

Quick Sort Code

quicksortweek3 - Notepad

File Edit Format View Help

```
//CH.SC.U4CSE24106
#include <stdio.h>

void quicksort(int a[],int low,int high){
        int i=low,j=high;
        int pivot=a[(low+high)/2];

        while(i<=j){
                while(a[i]<pivot)
                        i++;
                while(a[j]>pivot)
                        j--;
                if(i<=j){
                        int t=a[i];
                        a[i]=a[j];
                        a[j]=t;
                        i++;
                        j--;
                }
        }

        if(low<j)
                quicksort(a,low,j);
        if(i<high)
                quicksort(a,i,high);
}

int main(){
        int a[100],n;

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

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

        quicksort(a,0,n-1);

        printf("Sorted list:\n");
        for(int i=0;i<n;i++)
                printf("%d ",a[i]);
```

```
        printf("Enter elements:\n");
        for(int i=0;i<n;i++)
                scanf("%d",&a[i]);

        quicksort(a,0,n-1);

        printf("Sorted list:\n");
        for(int i=0;i<n;i++)
                printf("%d ",a[i]);

        printf("\nRoll number: CH.SC.U4CSE24106");

        return 0;
}
```

Output:

```
C:\Users\savit\OneDrive\Desktop\Sem IV CSE\Ws1 haskell>quicksortweek3
Enter number of elements: 12
Enter elements:
157
110
147
122
111
149
151
141
123
112
117
133
Sorted list:
110 111 112 117 122 123 133 141 147 149 151 157
Roll number: CH.SC.U4CSE24106
```

Time Complexity:

The time complexity is typically O(n log n) when the pivot splits the array well. This happens because we sort each half separately, reducing the problem size quickly. In unlucky cases where the pivot is poor, the time can rise to $O(n^2)$.

Space Complexity

The space complexity comes from the recursion stack. With good splits, we use O(log n) stack space. With bad splits, we might need O(n) stack space. Besides the stack, the algorithm uses only O(1) extra memory for temporary variables.