**Slip 4**

Q.1) Write a program to implement a Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements

// C program for Merge Sort

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

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]

    // Initial index of first subarray

    i = 0;

    // Initial index of second subarray

    j = 0;

    // Initial index of merged subarray

    k = l;

    while (i < n1 && j < n2) {

        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) {

        arr[k] = L[i];

        i++;

        k++;

    }

    // Copy the remaining elements of

    // R[], if there are any

    while (j < n2) {

        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 r

        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 code

int main()

{

    int arr[] = { 12, 11, 13, 5, 6, 7 };

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");

    printArray(arr, arr\_size);

    clock\_t s,e,z;

    s=clock();

    mergeSort(arr, 0, arr\_size - 1);

    printf("\nSorted array is \n");

    printArray(arr, arr\_size);

    e=clock();

    z=e-s;

    printf("\nTime taken: %f" , (float) z /CLOCKS\_PER\_SEC);

    printf(" seconds");

    return 0;

}

Q.2) Write a program to implement Knapsack problems using Greedy method

//Knapsack Problem using Greedy Solution

#include<stdio.h>

int main()

{

    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;

    int i,j,num;

    printf("Enter number of items :");

    scanf("%d",&num);

    for (i = 0; i < num; i++)

    {

        printf("\n\nEnter Weight and Profit for item[%d] :\n",i);

        scanf("%f %f", &weight[i], &profit[i]);

    }

    printf("\n\nEnter capacity of knapsack :\n");

    scanf("%f",&capacity);

    for(i=0;i<num;i++)

        ratio[i]=profit[i]/weight[i];

    for (i = 0; i < num; i++)

    {

        for (j = i + 1; j < num; j++)

         {

            if (ratio[i] < ratio[j])

            {

                temp = ratio[j];

                ratio[j] = ratio[i];

                ratio[i] = temp;

                temp = weight[j];

                weight[j] = weight[i];

                weight[i] = temp;

                temp = profit[j];

                profit[j] = profit[i];

                profit[i] = temp;

            }

        }

    }

    printf("\nKnapsack Problem using Greedy Method :\n");

    for (i = 0; i < num; i++)

    {

        if (weight[i] > capacity)

            break;

        else

        {

            Totalvalue = Totalvalue + profit[i];

            capacity = capacity - weight[i];

        }

    }

    if (i < num)

        Totalvalue = Totalvalue + (ratio[i]\*capacity);

    printf("\nThe maximum value is :%f\n",Totalvalue);

    return 0;

}