**College : Vishwakarma Institute of Technology**

**Name : Vedika Vikas Sontakke**

**Roll no : 37**

**PRN No : 1222006**

**Course : Data Structure**

**Assignment No 1 Problem Statement 7**

* **Quick Sort**
* **Aim :**

WAP to implement Quick Sort on 1D array of Employee

structure (contains employee\_name, emp\_no, emp\_salary), with key as

emp\_no. And count the number of swap performed.

* **Program :**

/\*

WAP to implement Quick Sort on 1D array of Employee

structure (contains employee\_name, emp\_no, emp\_salary), with key as

emp\_no. And count the number of swap performed.

\*/

#include<stdio.h>

#include <string.h>

int count\_swap = 0;

struct Employee

{

    char employee\_name[40];

    int emp\_no;

    int emp\_salary;

};

void swap(struct Employee \*emp\_array , int x  , int y)

{

    struct Employee temp;

    temp = emp\_array[x];

    emp\_array[x] = emp\_array[y];

    emp\_array[y] = temp;

    count\_swap++;

}

void quick\_sort(struct Employee emp\_array[] , int first , int last)

{

    int i , j , pivot;

    if(first < last){

    i = first;

    j = last;

    pivot = first;

    while(i < j)

    {

       while(emp\_array[i].emp\_no <= emp\_array[pivot].emp\_no && i<last)

         i++;

       while(emp\_array[j].emp\_no > emp\_array[pivot].emp\_no)

         j--;

       if(i < j) swap(emp\_array , i , j);

    }

    swap(emp\_array , j , pivot);

    // for left digits

    quick\_sort(emp\_array , 0 , pivot-1);

      // for right digits;

    quick\_sort(emp\_array , pivot+1 , last);

    }

}

int main()

{

    int size ;

    printf("enter the size of the array \n");

    scanf("%d",&size);

    struct Employee emp\_array[size];

    printf("enter the employee no , employee name , employee salary :");

    for(int i=0 ; i<size ; i++)

       scanf("%d %s %d" , &emp\_array[i].emp\_no , &emp\_array[i].employee\_name , &emp\_array[i].emp\_salary);

    quick\_sort(emp\_array , 0 , size-1);

    printf("number of swaps are : %d" , count\_swap);

    printf("\n print the employee no , employee name , employee salary  after sorting :");

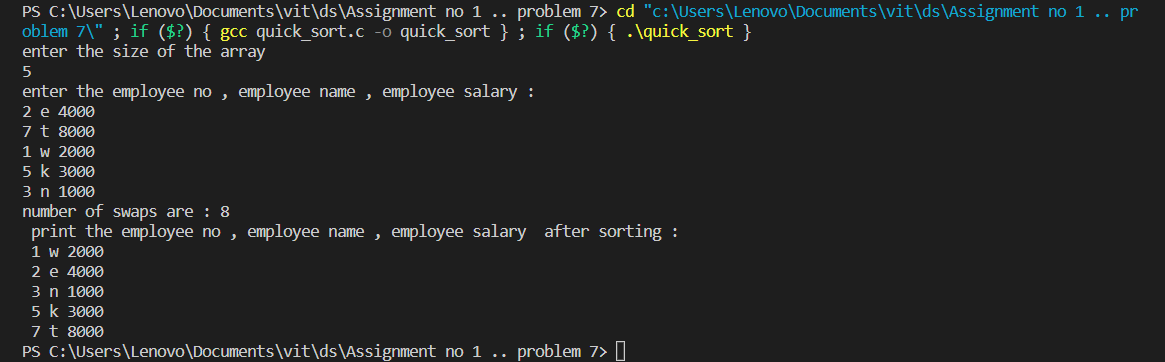
    for(int i=0 ; i<size ; i++)

       printf("\n %d %s %d ",emp\_array[i].emp\_no,emp\_array[i].employee\_name , emp\_array[i].emp\_salary);

    return 0;

}

**Output :**



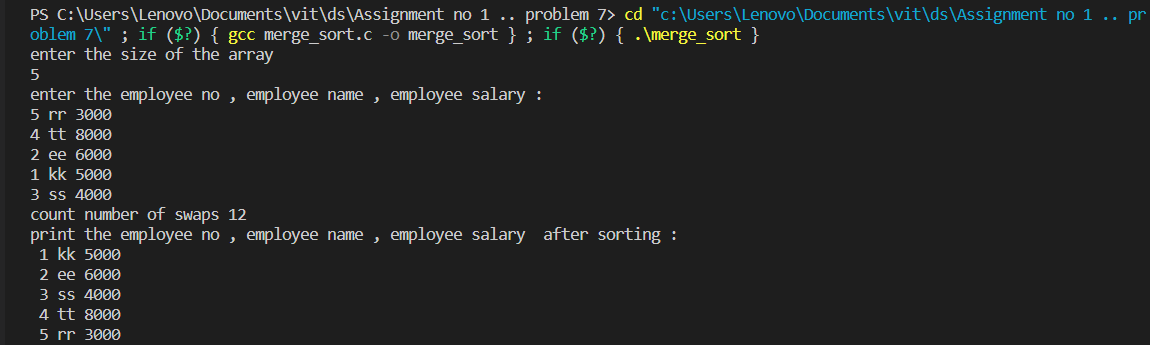
* **Merge Sort**
* **Aim :**

WAP to implement Merge Sort on 1D array of Employee

structure (contains employee\_name, emp\_no, emp\_salary), with key as

emp\_no. And count the number of swap performed.

* **Program :**
* /\*
* WAP to implement Merge Sort on 1D array of Employee
* structure (contains employee\_name, emp\_no, emp\_salary), with key as
* emp\_no. And count the number of swap performed.
* \*/
* #include<stdio.h>
* void merge\_sort();
* void merge();
* int count\_swap = 0;
* struct Employee
* {
* char employee\_name[40];
* int emp\_no;
* int emp\_salary;
* };
* int main()
* {
* int size;
* printf("enter the size of the array \n");
* scanf("%d",&size);
* struct Employee emp\_array[size];
* printf("enter the employee no , employee name , employee salary :");
* for(int i=0 ; i<size ; i++)
* scanf("%d %s %d" , &emp\_array[i].emp\_no , &emp\_array[i].employee\_name , &emp\_array[i].emp\_salary);
* merge\_sort(emp\_array , 0 , size-1);
* printf("count number of swaps %d", count\_swap);
* printf("\nprint the employee no , employee name , employee salary  after sorting :");
* for(int i=0 ; i<size ; i++)
* printf("\n %d %s %d ",emp\_array[i].emp\_no,emp\_array[i].employee\_name , emp\_array[i].emp\_salary);
* return 0;
* }
* void merge\_sort(struct Employee emp\_array[], int first , int last)
* {
* int mid ;
* if(first < last)
* {
* mid = (first+last)/2;
* merge\_sort(emp\_array , first , mid);
* merge\_sort(emp\_array , mid+1 , last);
* merge(emp\_array , first , mid , last);
* }
* }
* void merge(struct Employee emp\_array[], int first , int mid , int last)
* {
* struct Employee b[50];
* int i , j , k;
* i = first ;
* j = mid+1;
* k = first;
* while(i<=mid && j<=last)
* {
* if(emp\_array[i].emp\_no <= emp\_array[j].emp\_no)
* {
* b[k++] = emp\_array[i++];
* count\_swap++;
* }
* else {
* b[k++] = emp\_array[j++];
* count\_swap++;
* }
* }
* if(i>mid)
* {
* while(j<=last){
* b[k++]= emp\_array[j++];
* count\_swap++;
* }
* }
* else{
* while(i<=mid){
* b[k++] = emp\_array[i++];
* count\_swap++;
* }
* }
* for( i=first ; i<=last ; i++)
* emp\_array[i] = b[i];
* }
* **Output :**

****

* **Heap Sort**
* **Aim :**

WAP to implement Heap Sort on 1D array of Employee

structure (contains employee\_name, emp\_no, emp\_salary), with key as

emp\_no. And count the number of swap performed

* **Program :**

/\*

WAP to implement Heap Sort on 1D array of Employee

structure (contains employee\_name, emp\_no, emp\_salary), with key as

emp\_no. And count the number of swap performed.

\*/

#include<stdio.h>

void heap\_Sort();

void heapify();

int count\_swap = 0;

struct Employee

{

    char employee\_name[40];

    int emp\_no;

    int emp\_salary;

};

int main()

{

    int size;

    printf("enter the size of the array \n");

    scanf("%d",&size);

    struct Employee emp\_array[size];

    printf("enter the employee no , employee name , employee salary :");

    for(int i=0 ; i<size ; i++)

       scanf("%d %s %d" , &emp\_array[i].emp\_no , &emp\_array[i].employee\_name , &emp\_array[i].emp\_salary);

    heap\_Sort(emp\_array , size);

     printf("count number of swaps are %d", count\_swap);

    printf("\nprint the employee no , employee name , employee salary  after sorting :");

    for(int i=0 ; i<size ; i++)

       printf("\n %d %s %d ",emp\_array[i].emp\_no,emp\_array[i].employee\_name , emp\_array[i].emp\_salary);

    return 0;

}

void swap(struct Employee \*emp\_array , int x  , int y)

{

    struct Employee temp;

    temp = emp\_array[x];

    emp\_array[x] = emp\_array[y];

    emp\_array[y] = temp;

    count\_swap++;

}

void heapify(struct Employee emp\_array[], int size, int i)

{

    //  largest among root, left child and right child

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    // If left child is larger than root

    if (left < size && emp\_array[left].emp\_no > emp\_array[largest].emp\_no)

        largest = left;

    // If right child is larger than largest

    if (right < size && emp\_array[right].emp\_no > emp\_array[largest].emp\_no)

        largest = right;

    // Swap and continue heapifying if root is not largest

    if (largest != i) {

        swap(emp\_array , i , largest);

        // Recursively heapify the affected sub-tree

        heapify(emp\_array, size, largest);

    }

}

 void heap\_Sort(struct Employee emp\_array[], int size)

{

    // Build max heap

    for (int i = size / 2 - 1; i >= 0; i--)

        heapify(emp\_array, size, i);

    // Heap sort

    for (int i = size - 1; i >= 0; i--) {

        swap(emp\_array , 0 , i);

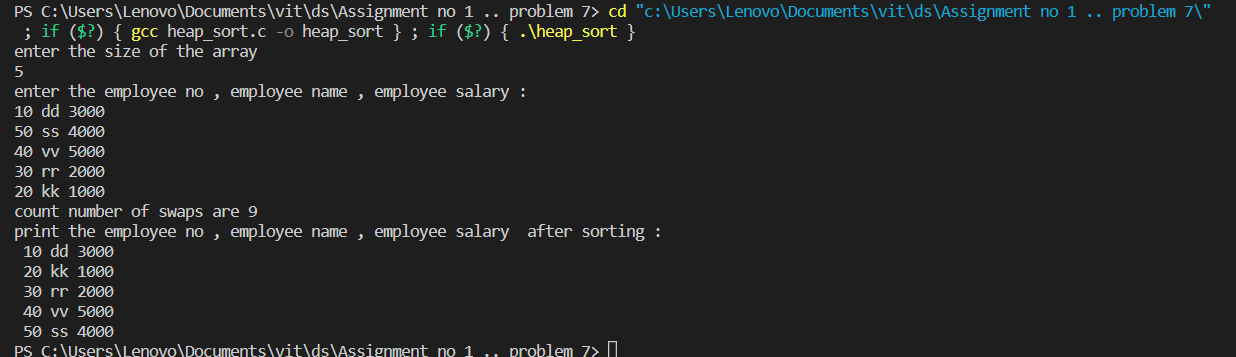
        // Heapify root element to get highest element at root again

        heapify(emp\_array, i, 0);

    }

}

* **Output :**

****

**-THANK YOU**