Name : Shrinivas Hatyalikar

Div: TY-B (B2)

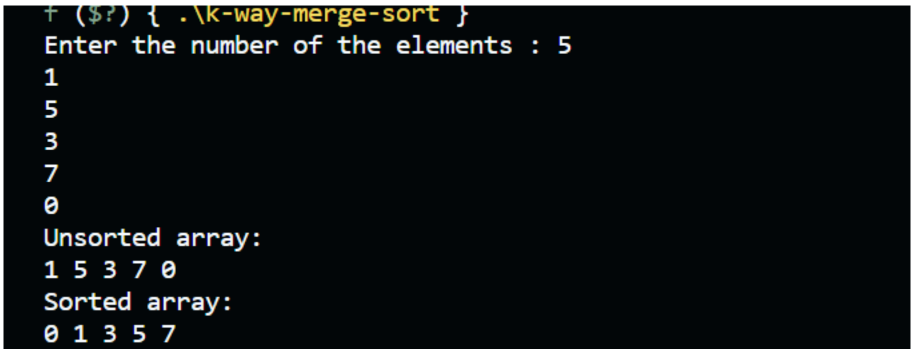
Roll No: 26

K-merge sort

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

typedef struct {
    int value;
    int index;
    int size;
    int end;
    int start;
} HeapNode;

void merge(int arr[], int left, int mid, int right){
    int i, j, k;
    int b[1000];
    for(i = left, j = mid + 1, k = left; i <= mid && j <= right; k++){
        if(arr[i] <= arr[j]){
            b[k] = arr[i++];
        }
        else{
            b[k] = arr[j++];
        }
    }
    while(i <= mid){
        b[k++] = arr[i++];
    }
    while(j <= right){
        b[k++] = arr[j++];
    }
    for(k = left; k <= right; k++){
        arr[k] = b[k];
    }
}
void mergeSort(int arr[],int left,int right,int n){
    if(left<right){
    int mid = (left+right)/2;
    mergeSort(arr,left,mid,n);
    mergeSort(arr,mid+1,right,n);
    merge(arr,left,mid,right);
```

```c
    }
}

void k_array_sort(int arr[], int n, int k, int sublist_size){
    for (int i = 0; i < k; i++) {
        if(i==k-1){
            int l = sublist_size*i;
            int temp[n-l];
            int x=0;
            for(int j=l; j<=n-1; ++j) temp[x++] = arr[j];
            mergeSort(temp, 0, n-l-1, n-l);
            x=0;
            for(int j=l; j<=n-1; ++j) arr[j] = temp[x++];
        }
        else{
            int l = sublist_size*i;
            int r = l+sublist_size-1;
            int temp[sublist_size];
            int x=0;
            for(int j=l; j<=r; ++j) temp[x++] = arr[j];
            mergeSort(temp, 0, sublist_size-1, sublist_size);
            x=0;
            for(int j=l; j<=r; ++j) arr[j] = temp[x++];
        }
    }
}


void min_heapify(HeapNode *heap, int index, int heap_size) {
    int smallest = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    if (left < heap_size && heap[left].value < heap[smallest].value) {
        smallest = left;
    }
    if (right < heap_size && heap[right].value < heap[smallest].value) {
        smallest = right;
    }
    if (smallest != index) {
        HeapNode temp = heap[index];
        heap[index] = heap[smallest];
        heap[smallest] = temp;
        min_heapify(heap, smallest, heap_size);
    }
}

HeapNode* build_heap(int array[], int sublist_size, int n, int k){
    HeapNode *heap = malloc(k * sizeof(HeapNode));
    // Initialize the heap and indexes
    for (int i = 0; i < k; i++) {
        if(i==k-1){
            int l = i * sublist_size;
            heap[i].value = array[l];
            heap[i].start = l;
            heap[i].index = i;
            heap[i].size = n-l;
            heap[i].end = n-1;
        }
        else{
            int l = i * sublist_size;
            heap[i].value = array[l];
            heap[i].start = l;
            heap[i].index = i;
            heap[i].size = sublist_size;
            heap[i].end = l+sublist_size-1;
        }
    }
    // Build the initial min heap
    for (int i = k / 2 - 1; i >= 0; i--) {
        min_heapify(heap, i, k);
    }
    return heap;
}

void heap_merge(int array[], int sublist_size, int n, int k) {
    HeapNode* heap = build_heap(array,sublist_size,n,k);
    // Merge the sorted subsets using the min heap property
    int final_arr[n];
    for (int i = 0; i < n; i++) {
        final_arr[i] = heap[0].value;  // Store the minimum element in the
sorted array
        // Replace the minimum element with the next element from its subset
        // int start_index = heap[0].start;
        int next_index = heap[0].start + 1;
        if (next_index <= heap[0].end) {
            heap[0].value = array[next_index];
            heap[0].start += 1;
        } else {
            heap[0].value = INT_MAX;  // Mark subset as fully merged
        }
        min_heapify(heap, 0, k);  // Fix the min heap property
    }
    for (int i = 0; i < n; i++) array[i] = final_arr[i];
    free(heap);
}

void k_way(int arr[],int left,int right, int n, int k){
    int sublist_size = n / k;
    k_array_sort(arr,n,k,sublist_size);
    heap_merge(arr,sublist_size,n,k);
}

void Print(int arr[],int n){
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");

}
int main(){
    int n;
    printf("Enter the number of the elements : ");
    scanf("%d", &n);
    int arr[n];
    for(int i=0; i<n; ++i){
        scanf("%d", &arr[i]);
    }
    printf("Unsorted array: \n");
    Print(arr,n);
    printf("Sorted array: \n");
    k_way(arr,0,n-1,n,4);
    Print(arr,n);
    return 0;
}
```

Output :



```
+ ($?) { .\k-way-merge-sort }
Enter the number of the elements : 5
1
5
3
7
0
Unsorted array:
1 5 3 7 0
Sorted array:
0 1 3 5 7
```