QA:

#include <stdio.h>

void merge(int arr[],int l,int mid,int r){

    int n1=mid-l+1;

    int n2=r-mid;

    int a[n1],b[n2];

    for(int i=0;i<n1;i++){

        a[i]=arr[l+i];

    }

    for(int i=0;i<n2;i++){

        b[i]=arr[mid+i+1];

    }

    int i=0,j=0,k=l;

    while(i<n1 && j<n2){

        if(a[i]>b[j]){

            arr[k]=b[j];

            k++; j++;

        }else{

            arr[k]=a[i];

            k++; i++;

        }

    }

    while(i<n1){

        arr[k]=a[i];

        k++; i++;

    }

    while(j<n2){

        arr[k]=b[j];

        k++; j++;

    }

}

void mergesort(int arr[],int l,int r){

    if(l<r){

        int m=(l+r)/2;

        mergesort(arr,l,m);

        mergesort(arr,m+1,r);

        merge(arr,l,m,r);

    }

}

int main(){

    int n;  scanf("%d",&n);

    int arr[n];

    for(int i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    mergesort(arr,0,n);

    for(int i=0;i<n;i++){

        printf("%d ",arr[i]);

    }

return 0;

}

QB:

#include <stdio.h>

void swap(int arr[],int a,int b){

    int t=arr[a];

    arr[a]=arr[b];

    arr[b]=t;

}

int partition(int arr[],int l,int r){

    int pivot=arr[r];

    int i=l-1;

    for(int j=l;j<r;j++){

        if(arr[j]<pivot){

            i++;

            swap(arr,i,j);

        }

    }

    swap(arr,i+1,r);

    return i+1;

}

void quicksort(int arr[],int l,int r){

    if(l<r){

        int pi=partition(arr,l,r);

        quicksort(arr,l,pi-1);

        quicksort(arr,pi+1,r);

    }

}

int main(){

    int n;  scanf("%d",&n);

    int arr[n];

    for(int i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    quicksort(arr,0,n);

    for(int i=0;i<n;i++){

        printf("%d ",arr[i]);

    }

return 0;}

QC:

#include <stdio.h>

void heapify(int arr[], int n, int i) {

    int temp, maximum, left\_index, right\_index;

    maximum = i;

    right\_index = 2 \* i + 2;

    left\_index = 2 \* i + 1;

    if (left\_index < n && arr[left\_index] > arr[maximum])

        maximum = left\_index;

    if (right\_index < n && arr[right\_index] > arr[maximum])

        maximum = right\_index;

    if (maximum != i) {

        temp = arr[i];

        arr[i] = arr[maximum];

        arr[maximum] = temp;

        heapify(arr, n, maximum);

    }

}

void heapsort(int arr[], int n) {

    int i, temp;

    for (i = n / 2 - 1; i >= 0; i--) {

        heapify(arr, n, i);

    }

    for (i = n - 1; i > 0; i--) {

        temp = arr[0];

        arr[0] = arr[i];

        arr[i] = temp;

        heapify(arr, i, 0);

    }

}

int main(){

    int n;  scanf("%d",&n);    int arr[n];

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

    heapsort(arr,n);

    for(int i=0;i<n;i++){

        printf("%d ",arr[i]);    }

return 0;}

QD:

#include <stdio.h>

// Function to perform insertion sort on a sublist

void insertionSort(int arr[], int n) {

    for (int i = 1; i < n; ++i) {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

// Function to merge two sorted subarrays

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++;

    }

}

// Function to perform modified merge sort

void modifiedMergeSort(int arr[], int l, int r, int k) {

    if (r - l + 1 <= k) {

        insertionSort(arr + l, r - l + 1);

        return;

    }

    int m = l + (r - l) / 2;

    modifiedMergeSort(arr, l, m, k);

    modifiedMergeSort(arr, m + 1, r, k);

    merge(arr, l, m, r);

}

int main() {

    int n, k;

    scanf("%d %d", &n, &k);

    int arr[n];

    for (int i = 0; i < n; ++i)

        scanf("%d", &arr[i]);

    modifiedMergeSort(arr, 0, n - 1, k);

    for (int i = 0; i < n; ++i)

        printf("%d ", arr[i]);

    printf("\n");

    return 0;

}

QE:

#include <stdio.h>

#include <math.h>

void swap(int\* a, int\* b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int partition(int arr[], int low, int high) {

    int pivot = arr[high];

    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {

        if (arr[j] < pivot) {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1);

}

void merge(int arr[],int l,int mid,int r){

    int n1=mid-l+1;

    int n2=r-mid;

    int a[n1],b[n2];

    for(int i=0;i<n1;i++){

        a[i]=arr[l+i];

    }

    for(int i=0;i<n2;i++){

        b[i]=arr[mid+i+1];

    }

    int i=0,j=0,k=l;

    while(i<n1 && j<n2){

        if(a[i]>b[j]){

            arr[k]=b[j];

            k++; j++;

        }else{

            arr[k]=a[i];

            k++; i++;

        }

    }

    while(i<n1){

        arr[k]=a[i];

        k++; i++;

    }

    while(j<n2){

        arr[k]=b[j];

        k++; j++;

    }

}

// QuickMerge Sort function

void quickMergeSort(int arr[], int low, int high, int maxDepth) {

    if (low < high) {

        if (maxDepth > 0) {

            int pi = partition(arr, low, high);

            quickMergeSort(arr, low, pi - 1, maxDepth - 1);

            quickMergeSort(arr, pi + 1, high, maxDepth - 1);

        } else {

            if(low<high){

        int m=(low+high)/2;

        quickMergeSort(arr,low,m,maxDepth-1);

        quickMergeSort(arr,m+1,high,maxDepth-1);

        merge(arr,low,m,high);

          }

        }

    }

}

int main() {

    int n;    scanf("%d", &n);

    int arr[n];

    for (int i = 0; i < n; ++i){

        scanf("%d", &arr[i]);}

    int maxDepth = (int)(2 \* log2(n)); //Calculate max depth (2logn)

    quickMergeSort(arr, 0, n - 1, maxDepth);

    for (int i = 0; i < n; ++i)}

        printf("%d ", arr[i]);}

return 0;}

QF:

#include <stdio.h>

int merge(int arr[], int p, int q, int r) {

    int inv = 0;

    int n1 = q - p + 1;

    int n2 = r - q;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) {

        L[i] = arr[p + i];

    }

    for (int j = 0; j < n2; j++) {

        R[j] = arr[q + j + 1];

    }

    int i = 0, j = 0, k = p;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k++] = L[i++];

        } else {

            arr[k++] = R[j++];

            inv += (n1 - i); // Count inversions

        }

    }

    while (i < n1) {

        arr[k++] = L[i++];

    }

    while (j < n2) {

        arr[k++] = R[j++];

    }

    return inv;

}

int mergeSort(int arr[], int p, int r) {

    int inv = 0;

    if (p < r) {

        int q = (p + r) / 2;

        inv += mergeSort(arr, p, q);

        inv += mergeSort(arr, q + 1, r);

        inv += merge(arr, p, q, r);

    }

    return inv;

}

int main() {

    int n;

    scanf("%d", &n);

    int arr[n];

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    printf("%d\n", mergeSort(arr, 0, n - 1));

    return 0;

}

QG:

#include <stdio.h>

#include <stdlib.h>

// Swap two elements

void swap(int\* a, int\* b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

// Heapify (maintain max-heap property)

void heapify(int\* heap, int size, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < size && heap[left] > heap[largest])

        largest = left;

    if (right < size && heap[right] > heap[largest])

        largest = right;

    if (largest != i) {

        swap(&heap[i], &heap[largest]);

        heapify(heap, size, largest);

    }

}

// Insert an element into the max-heap

void insert(int\* heap, int\* size, int capacity, int val) {

    if (\*size < capacity) {

           int i = \*size;

    while (i > 0 && val > heap[(i - 1) / 2]) {

        heap[i] = heap[(i - 1) / 2];

        i = (i - 1) / 2;

    }

    heap[i] = val;

    (\*size)++;

    } else if (val < heap[0]) {

        heap[0] = val;

        heapify(heap, \*size, 0);

    }

}

// Calculate the sum of k smallest elements

int sumOfKSmallest(int arr[], int n, int k) {

    int\* maxHeap = (int\*)malloc(k \* sizeof(int));

    int size = 0;

    for (int i = 0; i < k; ++i)

        insert(maxHeap, &size, k, arr[i]);

    for (int i = k; i < n; ++i) {

        if (arr[i] < maxHeap[0])

            insert(maxHeap, &size, k, arr[i]);

    }

    int sum = 0;

    for (int i = 0; i < k; ++i)

        sum += maxHeap[i];

    free(maxHeap);

    return sum;

}

int main() {

    int n, k;

    scanf("%d %d", &n, &k);

    int arr[n];

    for (int i = 0; i < n; ++i)

        scanf("%d", &arr[i]);

    int result = sumOfKSmallest(arr, n, k);

    printf("%d\n", result);

    return 0;

}

QH:

#include <stdio.h>

#include <stdlib.h>

// Function to merge two sorted arrays

int\* merge(int \*arr1, int size1, int \*arr2, int size2) {

    int\* result = (int\*)malloc((size1 + size2) \* sizeof(int));

    int i = 0, j = 0, k = 0;

    while (i < size1 && j < size2) {

        if (arr1[i] <= arr2[j]) {

            result[k++] = arr1[i++];

        } else {

            result[k++] = arr2[j++];

        }

    }

    while (i < size1) {

        result[k++] = arr1[i++];

    }

    while (j < size2) {

        result[k++] = arr2[j++];

    }

    return result;

}

// Function to merge k sorted arrays

int\* mergeKArrays(int\*\* arrays, int\* sizes, int k) {

    int\* merged = arrays[0]; // Initialize with the first array

    int mergedSize = sizes[0];

    for (int i = 1; i < k; i++) {

        merged = merge(merged, mergedSize, arrays[i], sizes[i]);

        mergedSize += sizes[i];

    }

    return merged;

}

int main() {

    int k;

    scanf("%d", &k);

    int\*\* arrays = (int\*\*)malloc(k \* sizeof(int\*));

    int\* sizes = (int\*)malloc(k \* sizeof(int));

    // Input arrays

    for (int i = 0; i < k; i++) {

        int size;

        scanf("%d", &size);

        sizes[i] = size;

        arrays[i] = (int\*)malloc(size \* sizeof(int));

        for (int j = 0; j < size; j++) {

            scanf("%d", &arrays[i][j]);

        }

    }

    int\* result = mergeKArrays(arrays, sizes, k);

    int x=0;

    for(int i=0;i<k;i++){

        x+=sizes[i];

    }

    // Output sorted merged array

    for (int i = 0; i < x; i++) {

        printf("%d ", result[i]);

    }

    // Free allocated memory

    for (int i = 0; i < k; i++) {

        free(arrays[i]);

    }

    free(arrays);

    free(sizes);

    free(result);

    return 0;

}

QI: //could be wrong !!

#include <stdio.h>

#include <stdlib.h>

// Function declarations

void insertMinHeap(int \*minHeap, int \*minSize, int element);

void insertMaxHeap(int \*maxHeap, int \*maxSize, int element);

int extractMin(int \*minHeap, int \*minSize);

int extractMax(int \*maxHeap, int \*maxSize);

void balanceHeaps(int \*minHeap, int \*minSize, int \*maxHeap, int \*maxSize);

int getMedian(int \*minHeap, int minSize, int \*maxHeap, int maxSize);

int main() {

    int q;

    scanf("%d", &q);

    int minHeap[100001] = {0}; // Min-heap

    int minSize = 0; // Size of min-heap

    int maxHeap[100001] = {0}; // Max-heap

    int maxSize = 0; // Size of max-heap

int arr[q];

int k=0;

    while (q--) {

        int query, x;

        scanf("%d", &query);

        if (query == 1) {

            scanf("%d", &x);

            if (minSize == 0 || x > minHeap[0]) {

                insertMinHeap(minHeap, &minSize, x);

            } else {

                insertMaxHeap(maxHeap, &maxSize, x);

            }

            balanceHeaps(minHeap, &minSize, maxHeap, &maxSize);

        } else if (query == 2) {

            arr[k++]=getMedian(minHeap, minSize, maxHeap, maxSize);

        }

    }

    for(int i=0;i<k;i++){

        printf("%d\n",arr[i]);

    }

    return 0;

}

// Function to insert element into min-heap

void insertMinHeap(int \*minHeap, int \*minSize, int element) {

    int i = \*minSize;

    while (i > 0 && element < minHeap[(i - 1) / 2]) {

        minHeap[i] = minHeap[(i - 1) / 2];

        i = (i - 1) / 2;

    }

    minHeap[i] = element;

    (\*minSize)++;

}

// Function to insert element into max-heap

void insertMaxHeap(int \*maxHeap, int \*maxSize, int element) {

    int i = \*maxSize;

    while (i > 0 && element > maxHeap[(i - 1) / 2]) {

        maxHeap[i] = maxHeap[(i - 1) / 2];

        i = (i - 1) / 2;

    }

    maxHeap[i] = element;

    (\*maxSize)++;

}

// Function to extract minimum element from min-heap

int extractMin(int \*minHeap, int \*minSize) {

    int minElement = minHeap[0];

    minHeap[0] = minHeap[--(\*minSize)];

    int i = 0;

    while ((2 \* i + 1) < \*minSize) {

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        int smallest = left;

        if (right < \*minSize && minHeap[right] < minHeap[left]) {

            smallest = right;

        }

        if (minHeap[i] <= minHeap[smallest]) {

            break;

        }

        int temp = minHeap[i];

        minHeap[i] = minHeap[smallest];

        minHeap[smallest] = temp;

        i = smallest;

    }

    return minElement;

}

// Function to extract maximum element from max-heap

int extractMax(int \*maxHeap, int \*maxSize) {

    int maxElement = maxHeap[0];

    maxHeap[0] = maxHeap[--(\*maxSize)];

    int i = 0;

    while ((2 \* i + 1) < \*maxSize) {

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        int largest = left;

        if (right < \*maxSize && maxHeap[right] > maxHeap[left]) {

            largest = right;

        }

        if (maxHeap[i] >= maxHeap[largest]) {

            break;

        }

        int temp = maxHeap[i];

        maxHeap[i] = maxHeap[largest];

        maxHeap[largest] = temp;

        i = largest;

    }

    return maxElement;

}

// Function to balance the sizes of the min-heap and max-heap

void balanceHeaps(int \*minHeap, int \*minSize, int \*maxHeap, int \*maxSize) {

    while (\*minSize > \*maxSize + 1) {

        insertMaxHeap(maxHeap, maxSize, extractMin(minHeap, minSize));

    }

    while (\*maxSize > \*minSize) {

        insertMinHeap(minHeap, minSize, extractMax(maxHeap, maxSize));

    }

}

// Function to get the median of the array

int getMedian(int \*minHeap, int minSize, int \*maxHeap, int maxSize) {

    if (minSize > maxSize) {

        return minHeap[0];

    } else if (minSize < maxSize) {

        return maxHeap[0];

    } else {

        return maxHeap[0];

    }

}

QJ:

#include <stdio.h>

struct slots{

    int si;

    int ei;

};

int partition(struct slots \* timings,int low,int high){

    int pivot=timings[high].si;

    int i=low;

    int j=low;

    while(i<=high){

        if(timings[i].si<=pivot){

            struct slots temp=timings[i];

            timings[i]=timings[j];

            timings[j]=temp;

            j++;

        }

        i++;

    }

    return j-1;

}

void sort(struct slots \* timings,int low,int high){

    if(low<high){

        int p=partition(timings,low,high);

        sort(timings,low,p-1);

        sort(timings,p+1,high);

    }

}

int mergeoverlap(struct slots\* timings, int n) {

    int i = 0;

    while (i < n - 1) {

        if (timings[i].ei >= timings[i + 1].si && timings[i].si <= timings[i + 1].ei) {

            timings[i].ei = timings[i + 1].ei;

            for (int j = i + 1; j < n - 1; j++) {

                timings[j] = timings[j + 1];

            }

            n--;

        } else {

            i++;

        }

    }

    return n;

}

int main(){

    int n;

    scanf("%d",&n);

    struct slots timings[n];

    for(int i=0;i<n;i++){

        scanf("%d %d",&timings[i].si,&timings[i].ei);

    }

    sort(timings,0,n-1);

    int newn=mergeoverlap(timings, n);

    printf("%d\n",newn);

    for(int i=0;i<newn;i++){

        printf("%d %d",timings[i].si,timings[i].ei);

        printf("\n");

    }

    return 0;

}