QA:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int getHexDigit(char \*hex, int pos) {

  int len = strlen(hex);

   if (pos > len) //If position exceeds length of hexadecimal number

        return 0;

    char digit = hex[len - pos];

    if (digit >= '0' && digit <= '9')

        return digit - '0';

    else if (digit >= 'A' && digit <= 'F')

        return 10 + digit - 'A';

    else

        return -1; // Invalid hexadecimal digit

}

void countSort(char \*\*arr, int n, int pos) {

    const int k = 16; // Possible hexadecimal digits (0-15)

    int count[k];

    char \*\*output = (char \*\*)malloc(n \* sizeof(char \*));

    for(int i=0;i<k;i++){

        count[i]=0;

    }

    // Count the occurrences of each digit

    for (int i = 0; i < n; i++) {

        int digit = getHexDigit(arr[i], pos);

        count[digit]++;

    }

    // Adjust count array to get correct positions

    for (int i = 1; i < k; i++)

        count[i] += count[i - 1];

    // Build the output array

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

        int digit = getHexDigit(arr[i], pos);

        output[--count[digit]] = arr[i];

    }

    // Copy the output array to arr so that arr now contains

    // sorted numbers according to current digit

    for (int i = 0; i < n; i++)

        arr[i] = output[i];

    free(output);

}

void radixSort(char \*\*arr, int n, int k) {

    // Perform counting sort for every digit from least significant to most significant

    for (int pos = 1; pos <= k; pos++)

        countSort(arr, n, pos);

}

int main() {

    int n, k;

    scanf("%d %d", &n, &k);

    char \*\*arr = (char \*\*)malloc(n \* sizeof(char \*));

    for (int i = 0; i < n; i++) {

        arr[i] = (char \*)malloc(101 \* sizeof(char)); // Maximum length of each hexadecimal number is 100

        scanf("%s", arr[i]);

    }

    radixSort(arr, n, k);

    // Output the sorted hexadecimal numbers

    for (int i = 0; i < n; i++)

        printf("%s ", arr[i]);

    printf("\n");

    // Free dynamically allocated memory

    for (int i = 0; i < n; i++)

        free(arr[i]);

    free(arr);

    return 0;

}

QB:

#include<stdio.h>

#include<stdlib.h>

struct node{

    float value;

    struct node\* next;

};

void insert(struct node\*\* head,float val){

    struct node\* newNode=(struct node\*)malloc(sizeof(struct node));

    newNode->value=val;

    newNode->next=NULL;

    if(\*head==NULL){

        \*head=newNode;

    }else{

        struct node\* curr = \*head;

        while(curr->next != NULL){

            curr=curr->next;

        }

        curr->next=newNode;

    }

}

void printList(struct node\* head){

    struct node\* curr=head;

    while(curr != NULL){

        printf("%.1f ",curr->value);

        curr=curr->next;

    }

}

void insertionSort(struct node\*\* head){

    if(\*head==NULL || (\*head)->next==NULL){

        return;

    }

    struct node\* sorted=NULL;

    struct node\* curr=\*head;

    while(curr!=NULL){

        struct node\* next=curr->next;

        if(sorted==NULL || curr->value < sorted->value){

            curr->next=sorted;

            sorted=curr;

        }else{

            struct node\* temp=sorted;

            while(temp->next != NULL && temp->next->value < curr->value){

                temp=temp->next;

            }

            curr->next=temp->next;

            temp->next=curr;

        }

        curr=next;

    }

    \*head=sorted;

}

int main(){

int n;  scanf("%d",&n);

// Array of linked list heads (buckets)

struct node\* buckets[n];

    for (int i = 0; i < n; i++) {

        buckets[i] = NULL;

    }

    for (int i = 0; i < n; i++) {

        float value;

        scanf("%f", &value);

        int index = (int)(value \* n);

        insert(&buckets[index], value);

    }

    for (int i = 0; i < n; i++) {

        insertionSort(&buckets[i]);

        printList(buckets[i]);

    }printf("\n");

    for (int i = 0; i < n; i++) {

        int count = 0;

        struct node\* current = buckets[i];

        while (current != NULL) {

            count++;

            current = current->next;

        }

        printf("%d ", count);

    }

    // Free dynamically allocated memory

    for (int i = 0; i < n; i++) {

        struct node\* current = buckets[i];

        while (current != NULL) {

            struct node\* temp = current;

            current = current->next;

            free(temp);

        }

    }

    return 0;

}

QC:

#include <stdio.h>

#include <stdlib.h>

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

    int count[n + 1]; // Count array to store frequencies

    int output[n]; // Output array to store sorted elements

    int index[n + 1]; // Index array to store indices of elements

    // Initialize count array with zeros

    for (int i = 0; i <= n; i++) {

        count[i] = 0;

    }

    // Count frequencies of elements

    for (int i = 0; i < n; i++) {

        count[arr[i]]++;

    }

    // Calculate cumulative frequencies

    for (int i = 1; i <= n; i++) {

        count[i] += count[i - 1];

    }

    // Build the output array

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

        output[count[arr[i]] - 1] = arr[i];

        index[count[arr[i]]] = i + 1; // Storing index with 1-based indexing

        count[arr[i]]--;

    }

    for (int i = 0; i < n; i++) {

        printf("%d ", output[i]);

    }printf("\n");

    for (int i = 1; i <= n; i++) {

        printf("%d ", index[i]);

    } printf("\n");

}

int main() {

    int n;

    scanf("%d", &n);

    int arr[n];

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    countingSort(arr, n);

    return 0;

}

QD:

#include <stdio.h>

#include <stdlib.h>

int contains(int arr[],int n,int sum){

    int count[n];

    for (int i = 0; i <n; i++) {

        count[i] = 0;

    }

    for (int i = 0; i < n; i++) {

        count[arr[i]]++;

    }

int missing[n]; int k=0;

    for(int i=0;i<n;i++){

        if(count[i]==0){

            missing[k++]=i;

        }

    }

    int left = 0, right = k - 1;

    while (left < right) {

        int currentSum = missing[left] + missing[right];

        if (currentSum == sum) {

            return 1; // Found the pair with the given sum

        } else if (currentSum < sum) {

            left++;

        } else {

            right--;

    }

    return 0;

}

int main() {

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

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);    }

    if(contains(arr,n,k)){

        printf("YES\n");

    }else{

        printf("NO\n");

    }

    return 0;}

QE:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for the pair

typedef struct Pair {

    int first;

    int second;

} Pair;

// Function to perform counting sort on the array of pairs based on the specified digit (either first or second)

void countingSort(Pair arr[], int n, int exp) {

    const int k = n;

    Pair output[k];

    int count[k + 1];

    // Initialize count array with zeros

    for (int i = 0; i <= k; i++) {

        count[i] = 0;

    }

    // Count the occurrences of the current digit in the array

    for (int i = 0; i < n; i++) {

        int index = (exp == 1) ? arr[i].first : arr[i].second;

        count[index]++;

    }

    // Calculate cumulative counts

    for (int i = 1; i <= k; i++) {

        count[i] += count[i - 1];

    }

    // Build the output array

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

        int index = (exp == 1) ? arr[i].first : arr[i].second;

        output[--count[index]] = arr[i];

    }

    // Copy the output array to the original array

    for (int i = 0; i < n; i++) {

        arr[i] = output[i];

    }

}

// Function to perform Radix Sort on the array of pairs

void radixSort(Pair arr[], int n) {

    countingSort(arr, n, 2); // Sort based on second digit

    countingSort(arr, n, 1); // Sort based on first digit

}

int main() {

    int n;

    scanf("%d", &n);

    // Read the pairs into the array

    Pair arr[n];

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i].first);

    }

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i].second);

    }

    radixSort(arr, n);

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i].first);

    }

    printf("\n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i].second);

    }

    printf("\n");

    return 0;

}

QF:

#include <stdio.h>

struct element{

    int value;

    int freq;

};

void frequencysort(struct element \* arr,int n){

    int freq[n+1];

    struct element output[n];

    for(int i=1;i<=n;i++){

        freq[i]=0;

    }

    for(int i=0;i<n;i++){

        freq[arr[i].value]++;

    }

     for(int i=0;i<n;i++){

        arr[i].freq=freq[arr[i].value];

     }

    int count[n+1];

    for(int i=1;i<=n;i++){

        count[i]=0;

    }

    for(int i=0;i<n;i++){

        count[arr[i].freq]++;

    }

    for(int i=2;i<=n;i++){

        count[i]=count[i]+count[i-1];

    }

    for(int i=n-1;i>=0;i--){

        output[count[arr[i].freq]-1]=arr[i];

        count[arr[i].freq]--;

    }

    for(int i=0;i<n;i++){

        arr[i]=output[i];

    }

}

int main(){

    int n;

    scanf("%d",&n);

    struct element arr[n];

    for(int i=0;i<n;i++){

        scanf("%d",&arr[i].value);

    }

    frequencysort(arr,n);

    int sorted[n];

    int k=0;

    for(int i=0;i<n;i++){

        while(arr[i].freq!=0){

        if(k>n){

            break;

        }

         sorted[k++]=arr[i].value;

         arr[i].freq--;

        }

    }

   for(int i=0;i<n;i++){

    printf("%d ",sorted[i]);

   }

    return 0;

}

QG://

QH:// time complexity is O(n^2)

#include <stdio.h>

#include <limits.h>

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

    int max\_sum = INT\_MIN;

    for (int i = 0; i < n; i++) {

        int sum = 0;

        int count = 0;

        for (int j = i; j < n; j++) {

            sum += arr[j];

            count++;

            // If the subarray size is greater than k,

            // break the inner loop and move to the next subarray

            if (count > k) {

                break;

            }

            // Update the maximum sum found so far

            if (sum > max\_sum) {

                max\_sum = sum;

            }

        }

    }

    return max\_sum;

}

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

    int mx=i;

    int ri=2\*i+2;

    int li=2\*i+1;

    if(li<n && arr[li]>arr[mx]){

        mx=li;

    }

    if(ri<n && arr[ri]>arr[mx]){

        mx=ri;

    }

    if(mx!=i){

        int t=arr[i];

        arr[i]=arr[mx];

        arr[mx]=t;

        heapify(arr,i,0);

    }

}

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

    for(int i=n/2-1;i>=0;i--){

        heapify(arr,n,i);

    }

    for(int i=n-1;i>0;i--){

        int t=arr[0];

        arr[0]=arr[i];

        arr[i]=t;

        heapify(arr,i,0);

    }

}

int main(){

   int n,k;  scanf("%d %d",&n,&k);    int a[n],b[n];

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

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

    int c[n];

for(int i=0;i<n;i++){

    c[i]=a[i]+b[i];

}

heapsort(c,n);

       printf("%d",maxSumSubarray(c,n,k));

    return 0;

}

QI:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_CHAR 26

// Function to calculate min sum of squares of characters after k removals

int minStringValue(char\* str, int k) {

    int l = 0;

    while (str[l] != '\0') {

        l++; // find length of string

    }

    // If k is greater than or equal to the length of the string,

    // the reduced string will become empty

    if (k >= l) {

        return 0;

    }

    // Initialize an array to store character frequencies

    int frequency[MAX\_CHAR] = {0};

    for (int i = 0; i < l; i++) {

        frequency[str[i] - 'a']++;

    }

    // Create a max heap (priority queue) to store character frequencies

    int\* heap = (int\*)malloc(MAX\_CHAR \* sizeof(int));

    int heapSize = 0;

    // Insert character frequencies into the heap

    for (int i = 0; i < MAX\_CHAR; i++) {

        if (frequency[i] > 0) {

            heap[heapSize++] = frequency[i];

        }

    }

    // Build a max heap

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

        int parent = i;

        while (parent < heapSize) {

            int leftChild = 2 \* parent + 1;

            int rightChild = 2 \* parent + 2;

            int largest = parent;

            if (leftChild < heapSize && heap[leftChild] > heap[largest]) {

                largest = leftChild;

            }

            if (rightChild < heapSize && heap[rightChild] > heap[largest]) {

                largest = rightChild;

            }

            if (largest != parent) {

                // Swap parent and largest

                int temp = heap[parent];

                heap[parent] = heap[largest];

                heap[largest] = temp;

                parent = largest;

            } else {

                break;

            }

        }

    }

    // Removal of k characters

    while (k--) {

        if (heapSize > 0) {

            heap[0]--;

            if (heap[0] == 0) {

                heapSize--;

            }

            // Heapify after decrementing

            int parent = 0;

            while (parent < heapSize) {

                int leftChild = 2 \* parent + 1;

                int rightChild = 2 \* parent + 2;

                int largest = parent;

                if (leftChild < heapSize && heap[leftChild] > heap[largest]) {

                    largest = leftChild;

                }

                if (rightChild < heapSize && heap[rightChild] > heap[largest]) {

                    largest = rightChild;

                }

                if (largest != parent) {

                    // Swap parent and largest

                    int temp = heap[parent];

                    heap[parent] = heap[largest];

                    heap[largest] = temp;

                    parent = largest;

                } else {

                    break;

                }

            }

        }

    }

    // After removal of k characters, find the sum of squares of string values

    int result = 0;

    for (int i = 0; i < heapSize; i++) {

        result += heap[i] \* heap[i];

    }

    free(heap); // Free allocated memory

    return result;

}

int main() {

    int n,k;scanf("%d %d",&n,&k);

    char str[n+1];

    for(int i=0;i<n;i++){

        scanf("%c ",&str[i]);

    }    scanf("%c",&str[n]);

    str[n+1]='\0';

    int k1 = n-k;

    printf("%d\n", minStringValue(str, k1));

    return 0;}

QJ:

#include<stdio.h>

#include<limits.h>

struct element{

    int value;

    int index;

};

void minheapify(struct element\* a,int i,int n){

 int l=2\*i+1;

    int r=2\*i+2;

    int smallest=i;

    if(l<n&&(a[l].value<a[smallest].value)){

        smallest=l;

    }

     if(r<n&&(a[r].value<a[smallest].value)){

        smallest=r;

    }

    if(smallest!=i){

        struct element temp=a[smallest];

        a[smallest]=a[i];

        a[i]=temp;

        minheapify(a,smallest,n);

    }

}

void buildminheap(struct element \*a,int n){

     for(int i=(n-1)/2;i>=0;i--){

        minheapify(a,i,n);

    }

}

int sheild(struct element \*a,int\* index,int x,int n){

    int  k=0;

    for(int j=0;j<n;j++){

        buildminheap(a,n);

        if(a[0].value<x){

            index[k++]=a[0].index;

            a[0].value=INT\_MAX;

        }

    }

    return k;

}

int main(){

    int n;

    int x;

    scanf("%d",&n);

    scanf("%d",&x);

    struct element a[n];

    for(int i=0;i<n;i++){

        scanf("%d",&a[i].value);

    }

    for(int i=0;i<n;i++){

        a[i].index=i+1;

    }

    int index[n];

    int k=sheild(a,index,x,n);

    printf("%d\n",k);

    for(int i=0;i<k;i++){

        printf("%d ",index[i]);

    }

    return 0;

}