III. Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach)

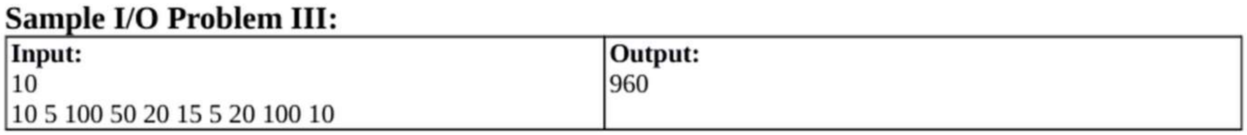
Input Format:

First line will take the size n of the array.

Second line will take array s an input.

Output Format:

Output will be the minimum computation cost required to merge all the elements of the array.



Sample I/O Problem III:

Input:	**Output:**
10 10 5 100 50 20 15 5 20 100 10	960

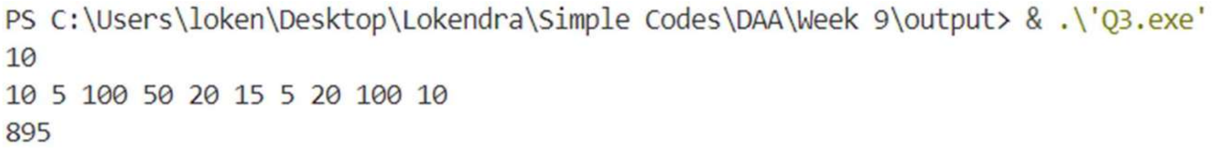
Source Code:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 1000


typedef struct {

   int arr[MAX];

   int size;

} MinHeap;


void insert(MinHeap *heap, int value) {

   heap->arr[heap->size] = value;

   heapifyUp(heap, heap->size);

   heap->size++;

}
```

```
int extractMin(MinHeap *heap) {
    int min = heap->arr[0];
    heap->arr[0] = heap->arr[heap->size - 1];
    heap->size--;
    heapifyDown(heap, 0);
    return min;
}
void heapifyUp(MinHeap *heap, int index) {
    int parent = (index - 1) / 2;
    while(index > 0 && heap->arr[index] < heap->arr[parent]) {
        int temp = heap->arr[index];
        heap->arr[index] = heap->arr[parent];
        heap->arr[parent] = temp;

        index = parent;
        parent = (index - 1) / 2;
    }
}
void heapifyDown(MinHeap *heap, int index) {
    int smallest = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;

    if(left < heap->size && heap->arr[left] < heap->arr[smallest])
        smallest = left;
    if(right < heap->size && heap->arr[right] < heap->arr[smallest])
```

```
        smallest = right;

    if(smallest != index) {
        int temp = heap->arr[index];
        heap->arr[index] = heap->arr[smallest];
        heap->arr[smallest] = temp;

        heapifyDown(heap, smallest);
    }
}

int main() {
    int n, i, a, b, cost, totalCost = 0;
    MinHeap heap;
    heap.size = 0;

    scanf("%d", &n);
    for(i = 0; i < n; i++) {
        int val;
        scanf("%d", &val);
        insert(&heap, val);
    }

    while(heap.size > 1) {
        a = extractMin(&heap);
        b = extractMin(&heap);
        cost = a + b;
        totalCost += cost;

        insert(&heap, cost);

    }


    printf("%d\n", totalCost);


    return 0;

}
```

Output:



```
PS C:\Users\loken\Desktop\Lokendra\Simple Codes\DAA\Week 9\output> & .\'Q3.exe'
10
10 5 100 50 20 15 5 20 100 10
895
```

WEEK 10

I. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

Input Format:

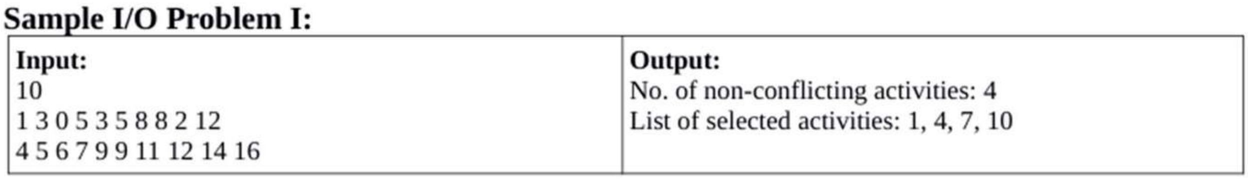
First line of input will take number of activities N.

Second line will take N space-separated values defining starting time for all the N activities.

Third line of input will take N space-separated values defining finishing time for all the N activities.

Output Format:

Output will be the number of non-conflicting activities and the list of selected activities.



Sample I/O Problem I:

Input:	**Output:**

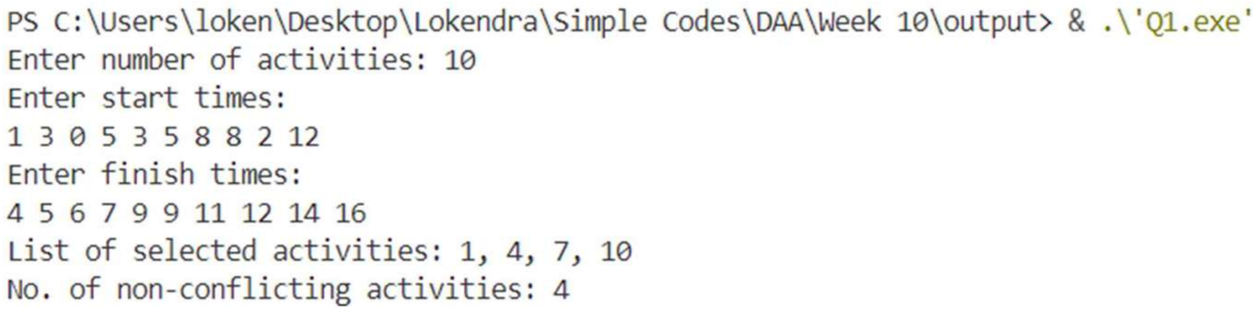
10
1 3 0 5 3 5 8 8 2 12
4 5 6 7 9 9 11 12 14 16
	No. of non-conflicting activities: 4
	List of selected activities: 1, 4, 7, 10

Source Code:

```
#include <stdio.h>


void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}


void sortActivities(int start[], int finish[], int index[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (finish[j] > finish[j + 1]) {
        swap(&finish[j], &finish[j + 1]);
        swap(&start[j], &start[j + 1]);
        swap(&index[j], &index[j + 1]);
      }
    }
  }
}

int main() {
  int n;
  printf("Enter number of activities: ");
  scanf("%d", &n);

  int start[n], finish[n], index[n];

  printf("Enter start times:\n");
  for (int i = 0; i < n; i++) {
    scanf("%d", &start[i]);
    index[i] = i + 1;
  }

  printf("Enter finish times:\n");
  for (int i = 0; i < n; i++) {
    scanf("%d", &finish[i]);
  }
```

```
    sortActivities(start, finish, index, n);

    printf("List of selected activities: %d", index[0]);
    int count = 1;
    int last_finish_time = finish[0];

    for (int i = 1; i < n; i++) {
        if (start[i] >= last_finish_time) {
            printf(", %d", index[i]);
            last_finish_time = finish[i];
            count++;
        }
    }

    printf("\nNo. of non-conflicting activities: %d\n", count);
    return 0;
}
```

Output:



```
PS C:\Users\loken\Desktop\Lokendra\Simple Codes\DAA\Week 10\output> & .\'Q1.exe'
Enter number of activities: 10
Enter start times:
1 3 0 5 3 5 8 8 2 12
Enter finish times:
4 5 6 7 9 9 11 12 14 16
List of selected activities: 1, 4, 7, 10
No. of non-conflicting activities: 4
```