

## **Optimized Bubble Sort**

In simple bubble sort algorithm, the time complexity will remain  $O(n^2)$  every time, even if the given array is already sorted.

Your task is to optimized the bubble sort such that its best case time complexity becomes  $O(n)$ .

### **Input Format**

In the function an integer vector is passed.

### **Output Format**

Return an integer vector in non decreasing order.

### **Sample Input**

{1, 3, 5, 7, 9}

### **Sample Output**

{1, 3, 5, 7, 9}}

### **Explanation**

given vector is already sorted

**Solution:** optimizedBubbleSort.cpp

## **Sorting with Comparator**

Given an integer vector and a bool variable flag, your task is to sort the vector in accordance to the boolean value. If the bool value passed is true then sort it in non-decreasing order or vice versa. You can use any sorting technique of your choice.

### **Input Format**

In the function an integer vector and a boolean is passed.

### **Output Format**

Return an integer vector.

### **Sample Input**

{111, 33, 5, 7, 29}, flag = 1

**Sample Output**

{5, 7, 29, 33, 111}

**Solution:** sortWithComparator.cpp

**Sorting Cartesian Points**

Given co-ordinates of N points on X-Y axis, your task is to sort them in accordance to the x-coordinate. If the x-coordinates are same then compare y-coordinates.

**Input Format**

In the function an integer vector of pairs is passed.

**Output Format**

Return the vector in sorted order.

**Sample Input**

{ (3, 4), (2, 3), (3, 7), (1, 5), (3, 4) }

**Sample Output**

{ (1, 5), (2, 3), (3, 4), (3, 4), (3, 7) }

**Solution:** sortCoords.cpp

**Chopsticks**

Given N sticks and an array **length** where each **length[i]** represents length of each stick. Your task is to make maximum number of pairs of these sticks such that the difference between the length of two sticks is at most **D**. A stick can't be part of more than one pair of sticks.

**Input Format**

In the function an integer vector length and number D is passed.

**Output Format**

Return an integer representing total number of such pairs.

**Sample Input**

{1, 3, 3, 9, 4}, x = 2

## Sample Output

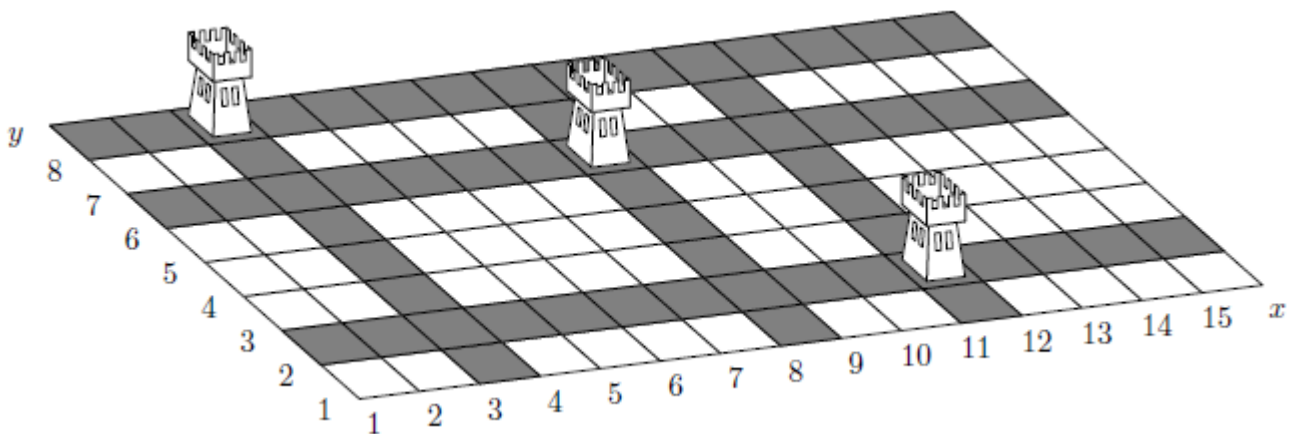
2

**Solution:** chopsticks.cpp

## Defense Kingdom

Theodore implements a new strategy game “Defense of a Kingdom”. On each level a player defends the Kingdom that is represented by a rectangular grid of cells. The player builds crossbow towers in some cells of the grid. The tower defends all the cells in the same row and the same column. No two towers share a row or a column.

The penalty of the position is the number of cells in the largest undefended rectangle. For example, the position shown on the picture has penalty 12.



Your task is to find the penalty of the give position.

### Input Format

In the function an width and height of the kingdom and a vector of pairs representing positions of towers is passed.

### Output Format

Return an integer representing the number of cells in the largest rectangle that is not defended by the towers.

### Sample Input

15 8 {(3, 8), (11, 2), (8, 6)}

### Sample Output

**Solution:** defenceKingdom.cpp