# Programming & Data Structure

## CS 11002

**Partha Bhowmick**

http://cse.iitkgp.ac.in/~pb

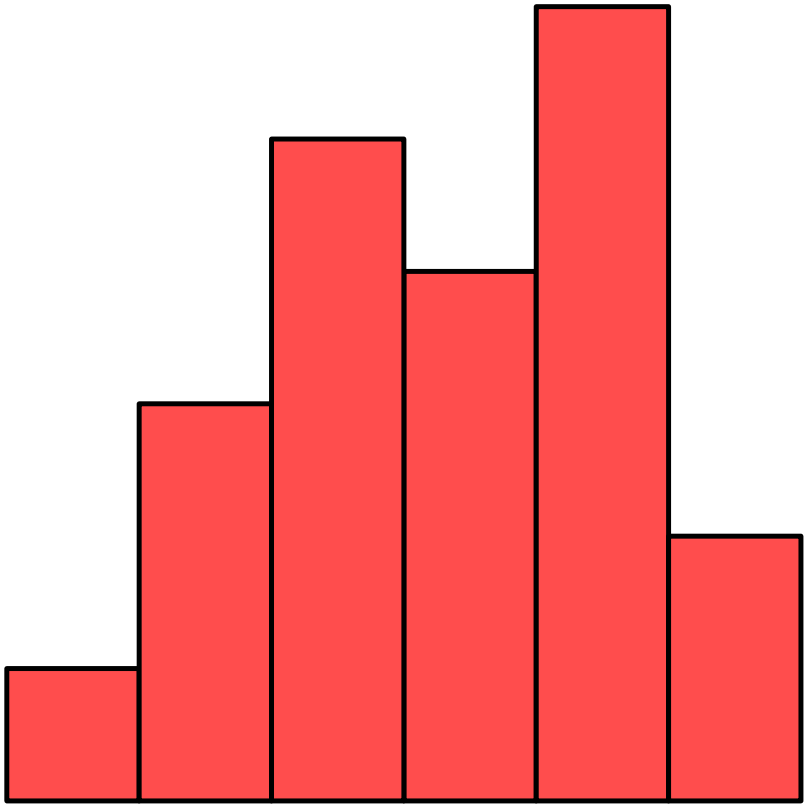
CSE Department
IIT Kharagpur

**Spring 2012-2013**

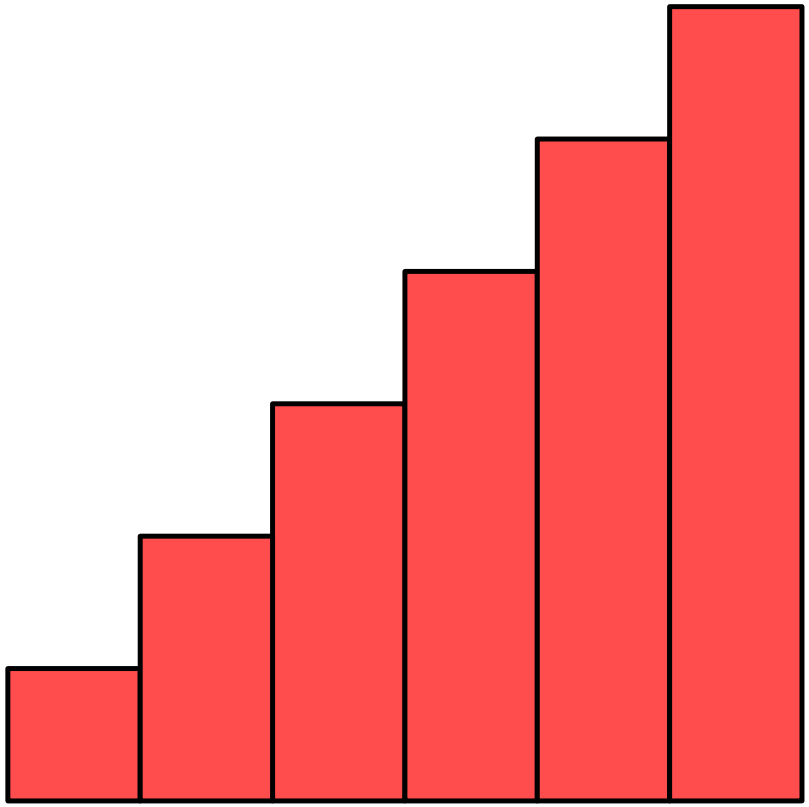
# Sorting

# Sorting





input



output
(in ascending order)

# Sorting

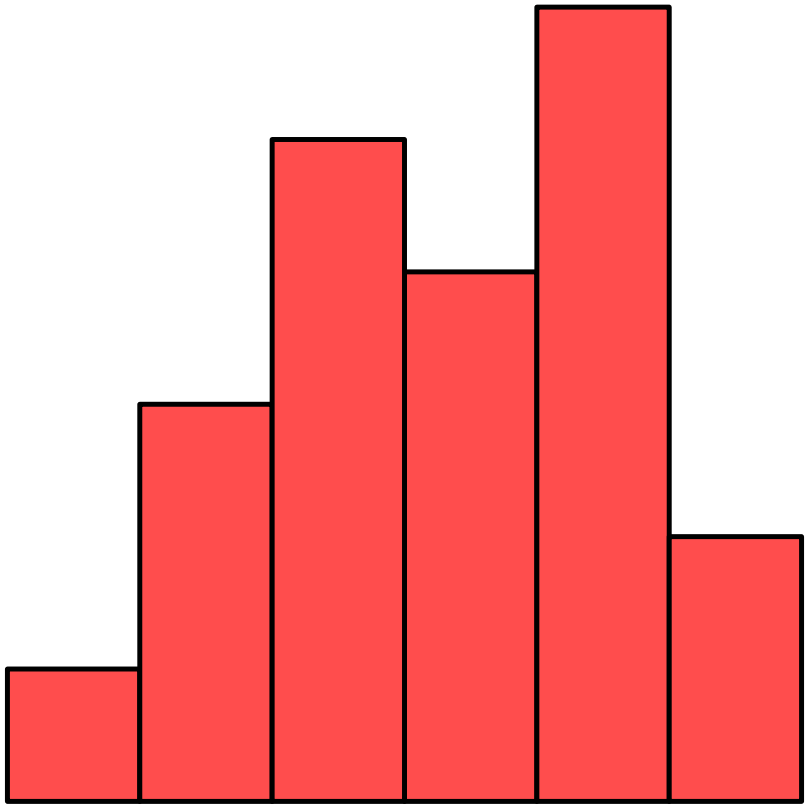


## Definition

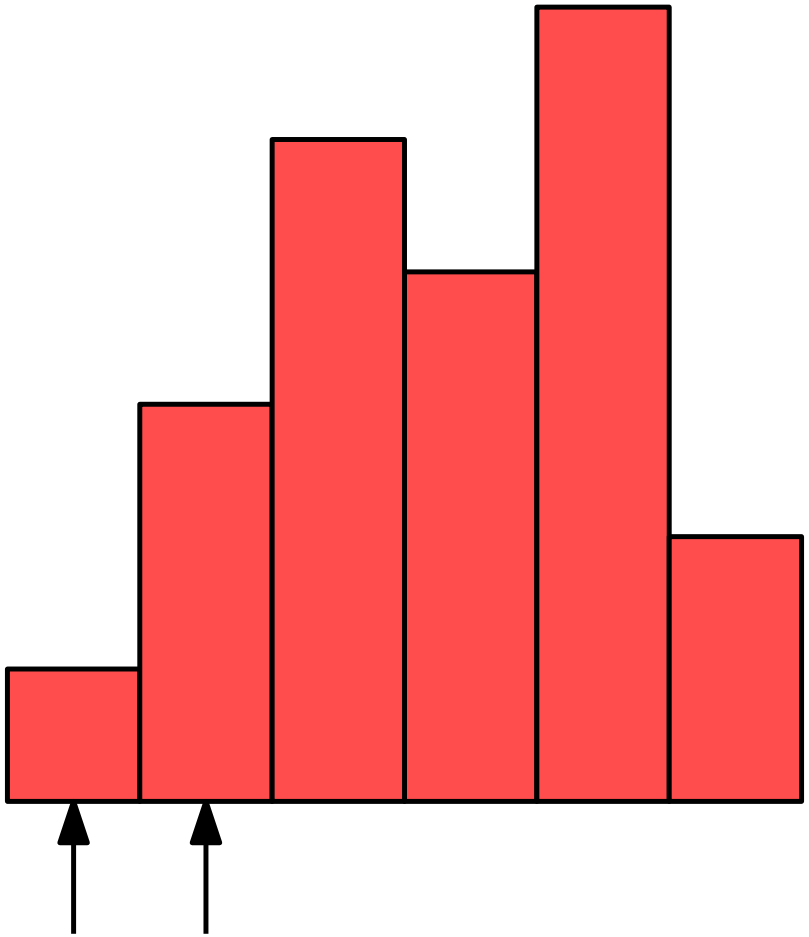
Given a list (1D array) of $n$ elements, we have to arrange the elements in non-decreasing (or non-increasing) order of their values.

**Constraint:** At any time instant, the computer can compare only two elements.

# Demo

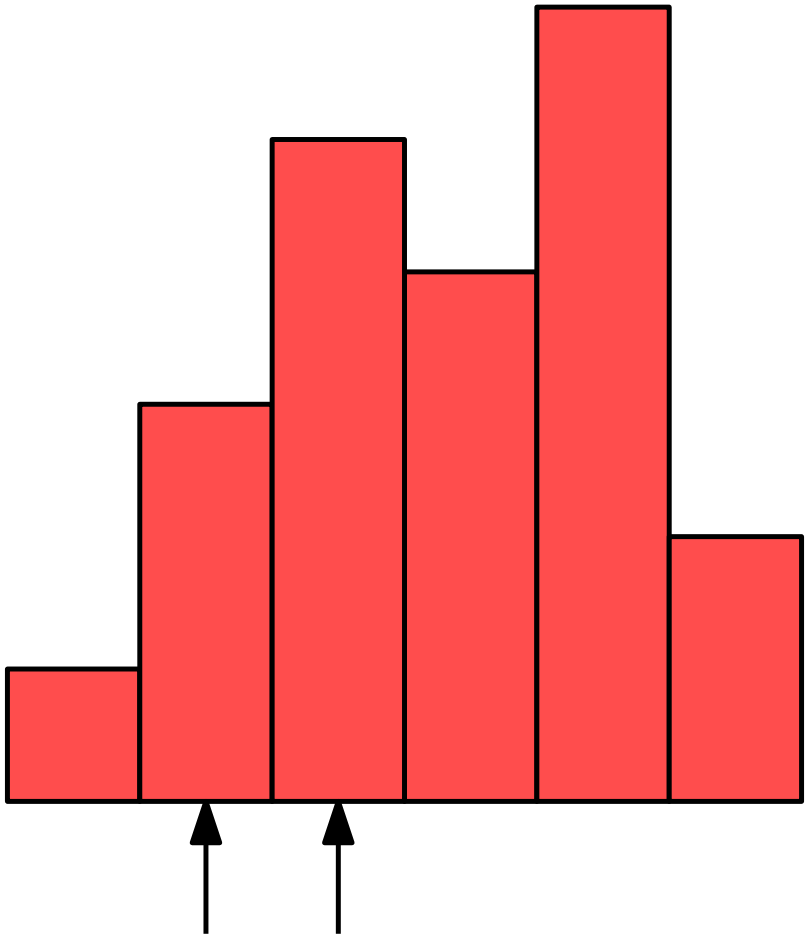


# Demo

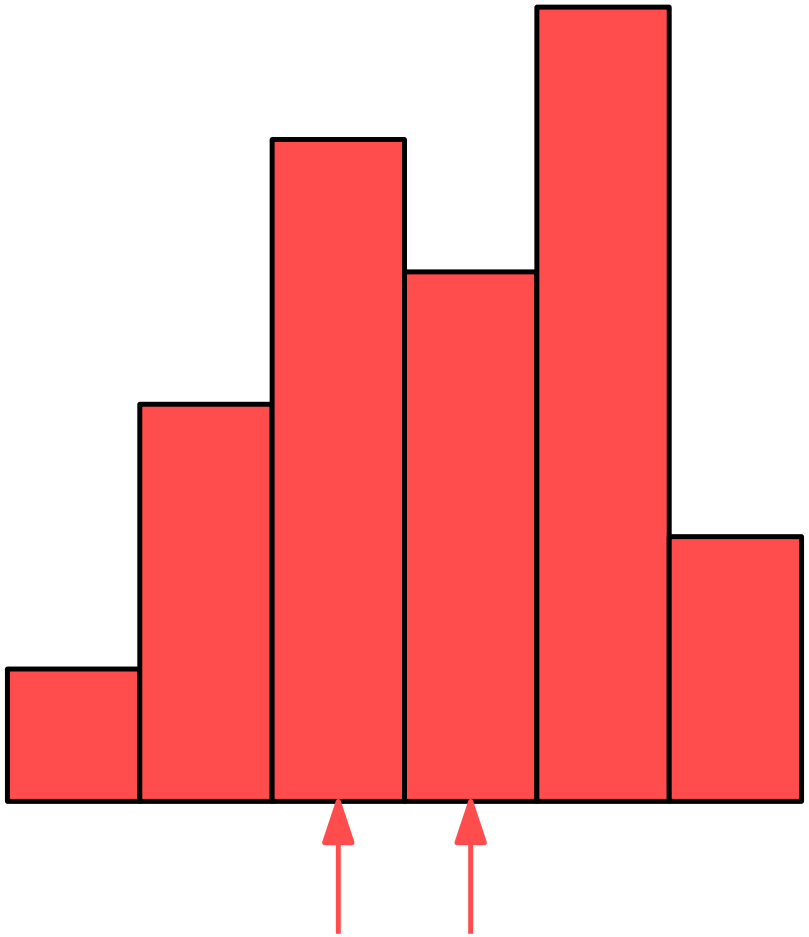


Pass 1 starts

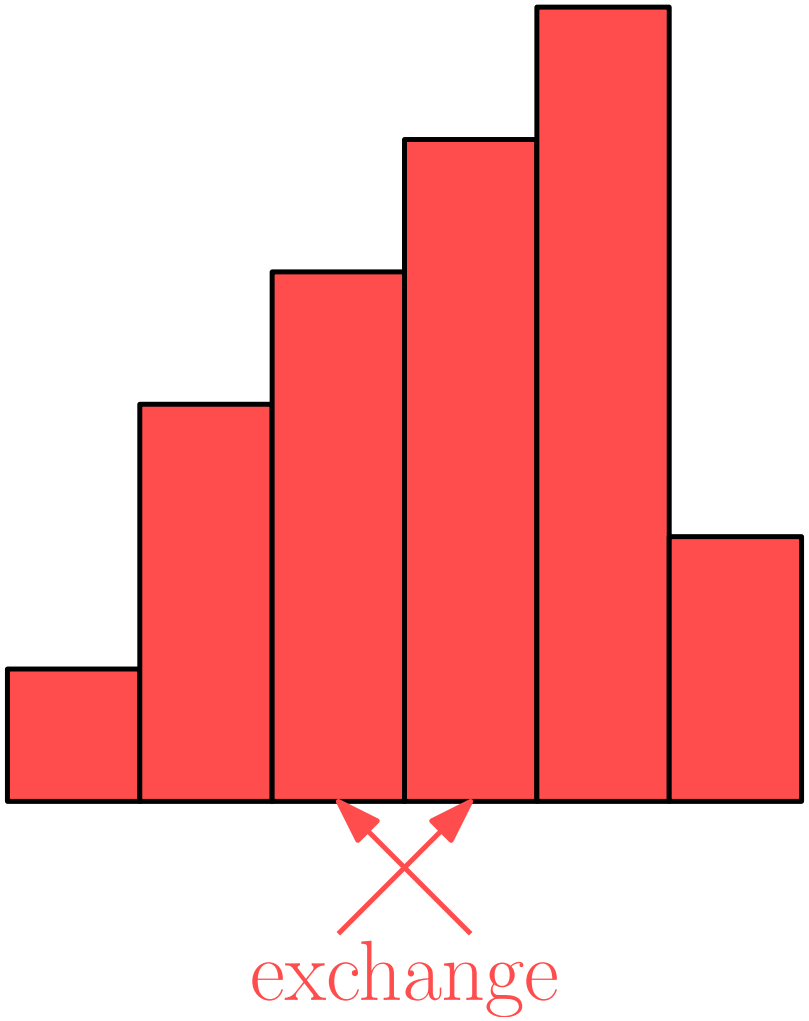
# Demo



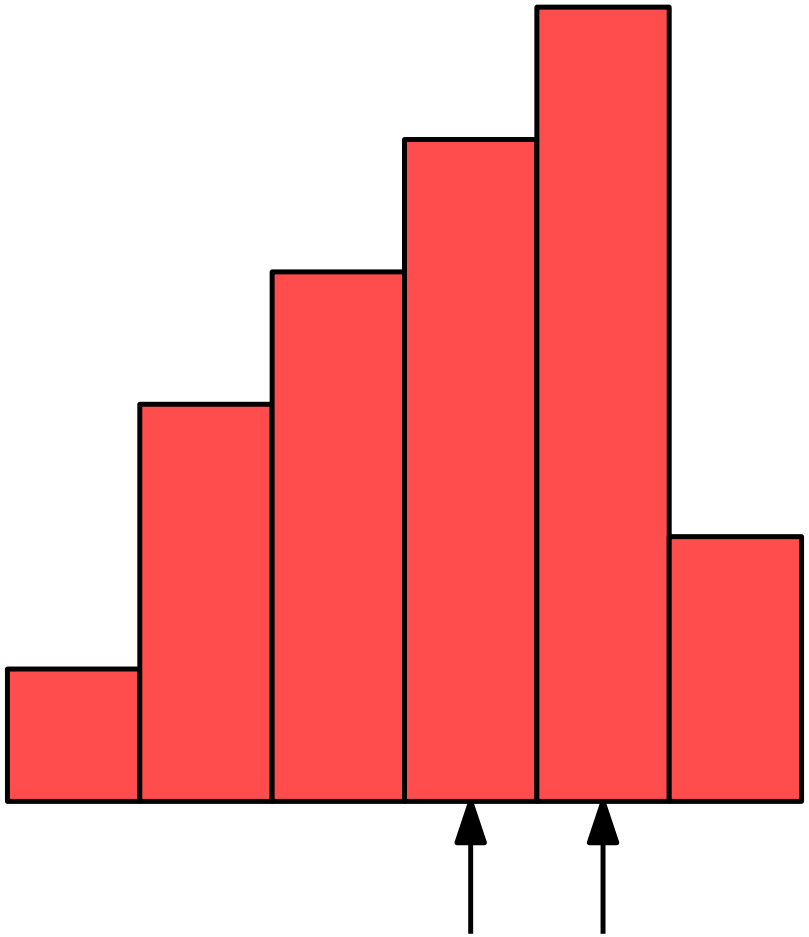
# Demo



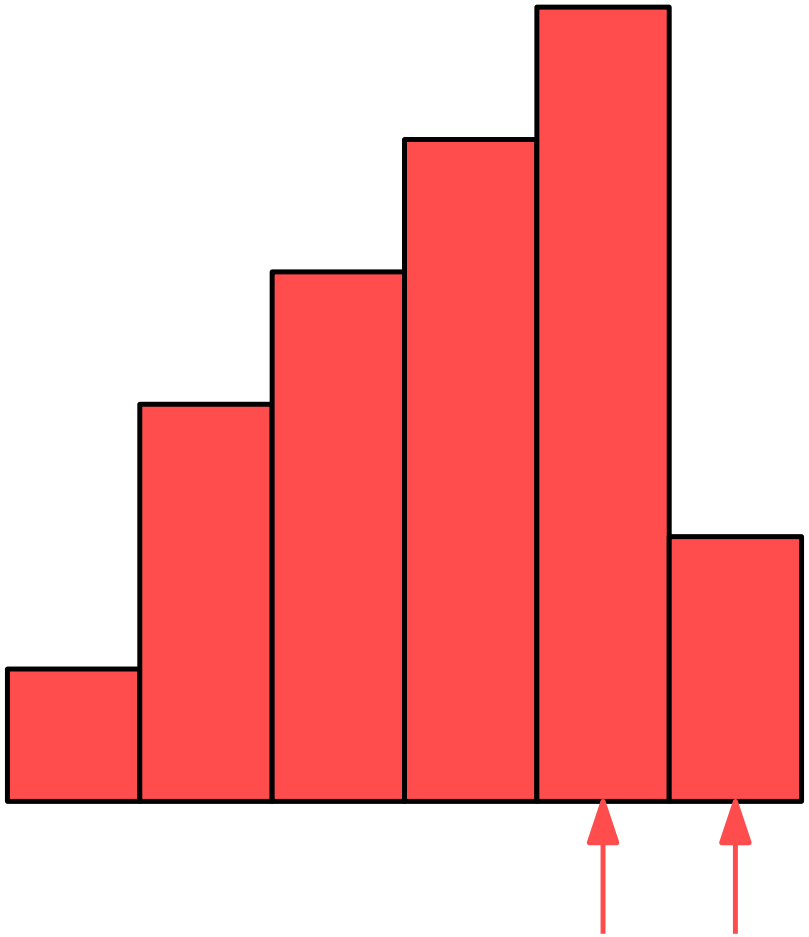
# Demo



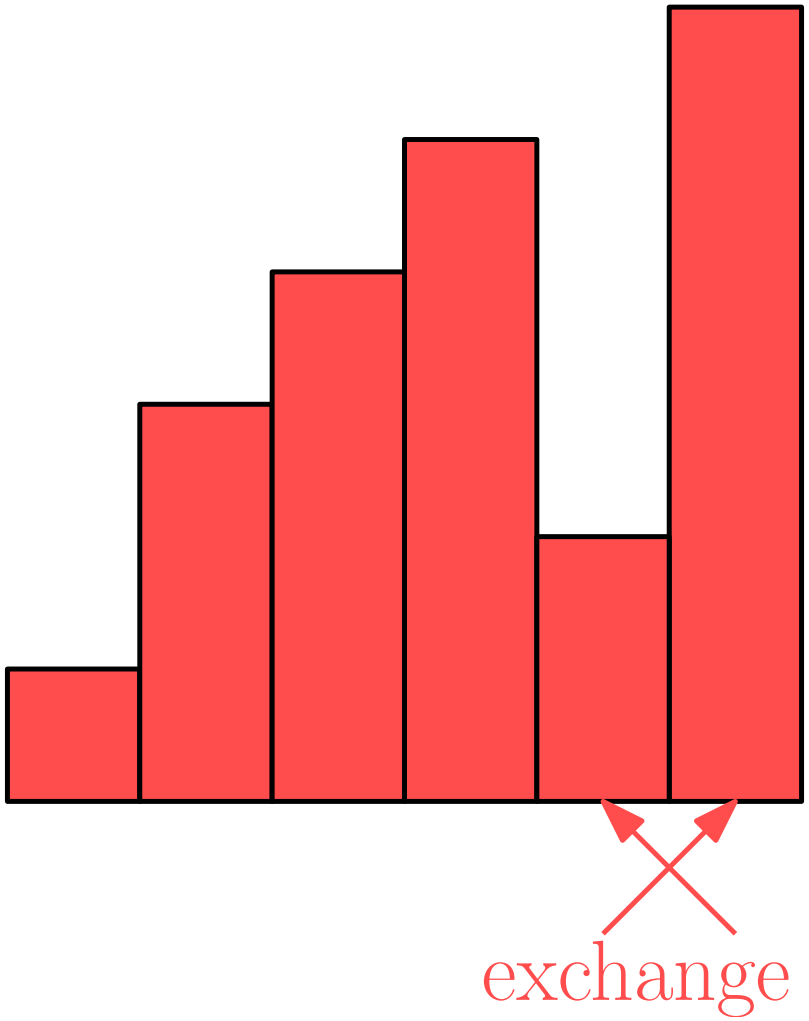
# Demo



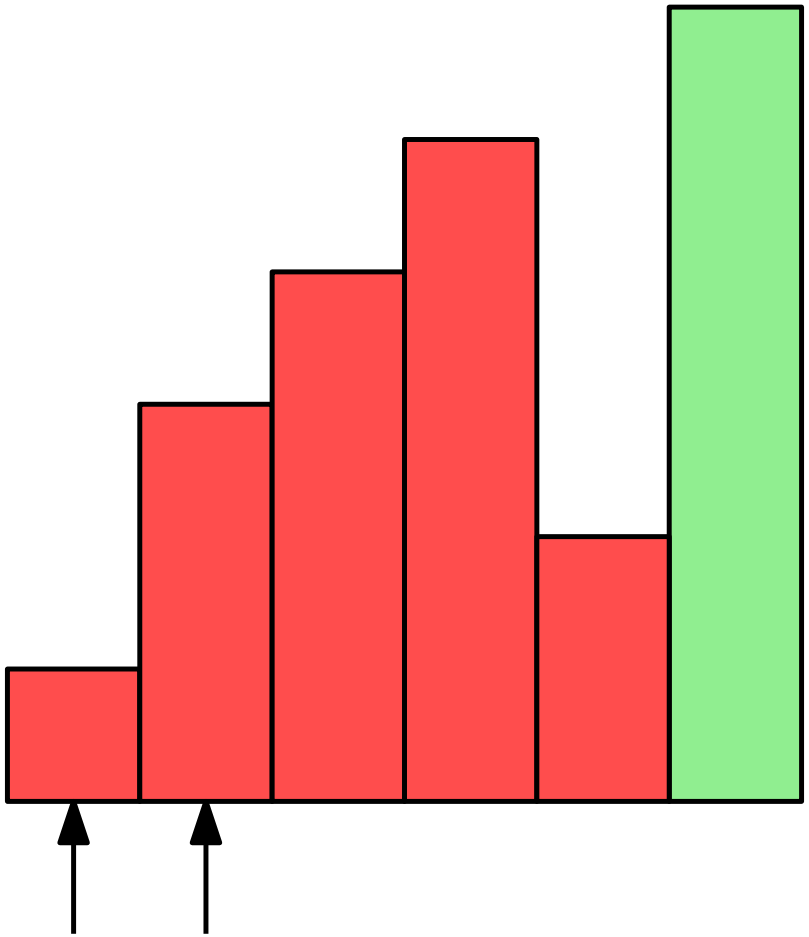
# Demo



# Demo

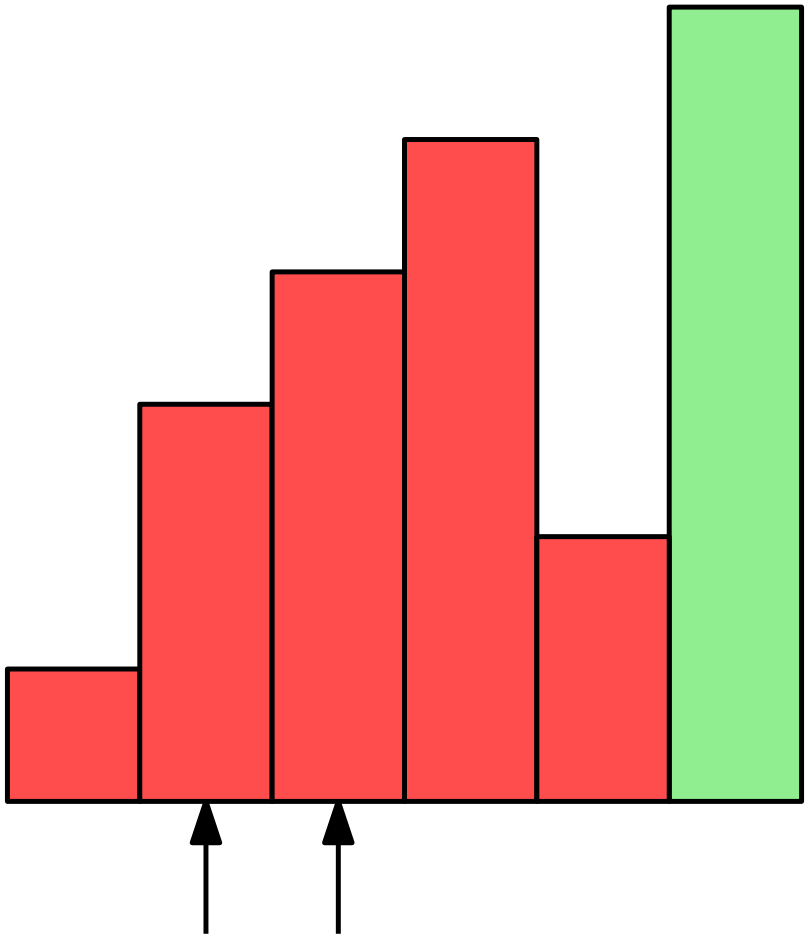


# Demo

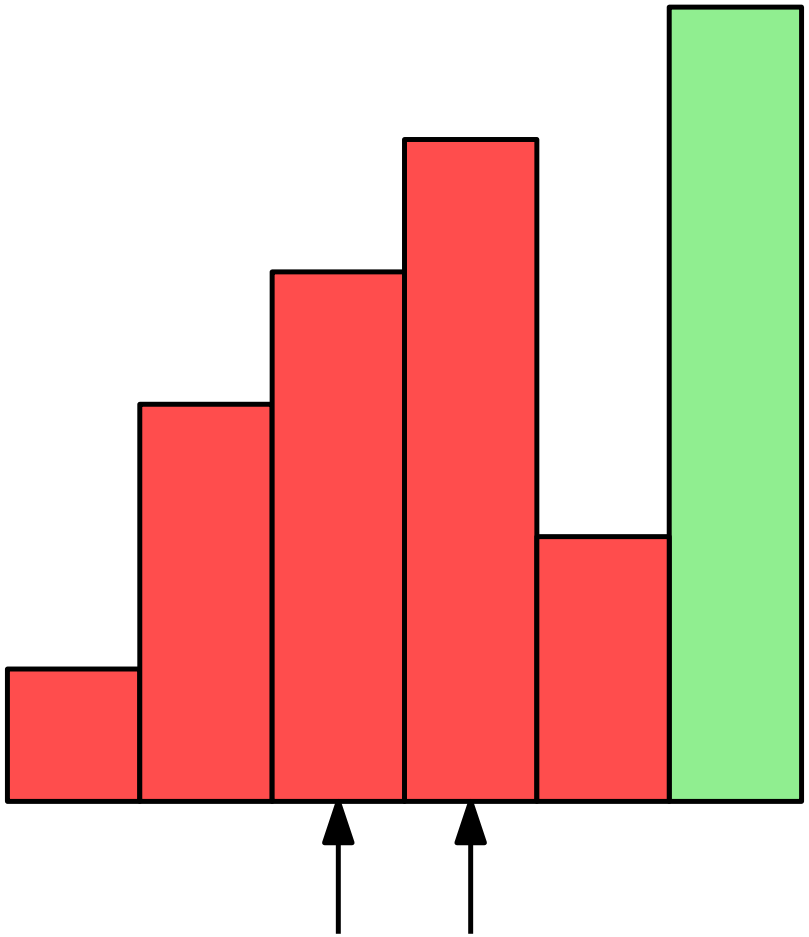


Pass 2 starts

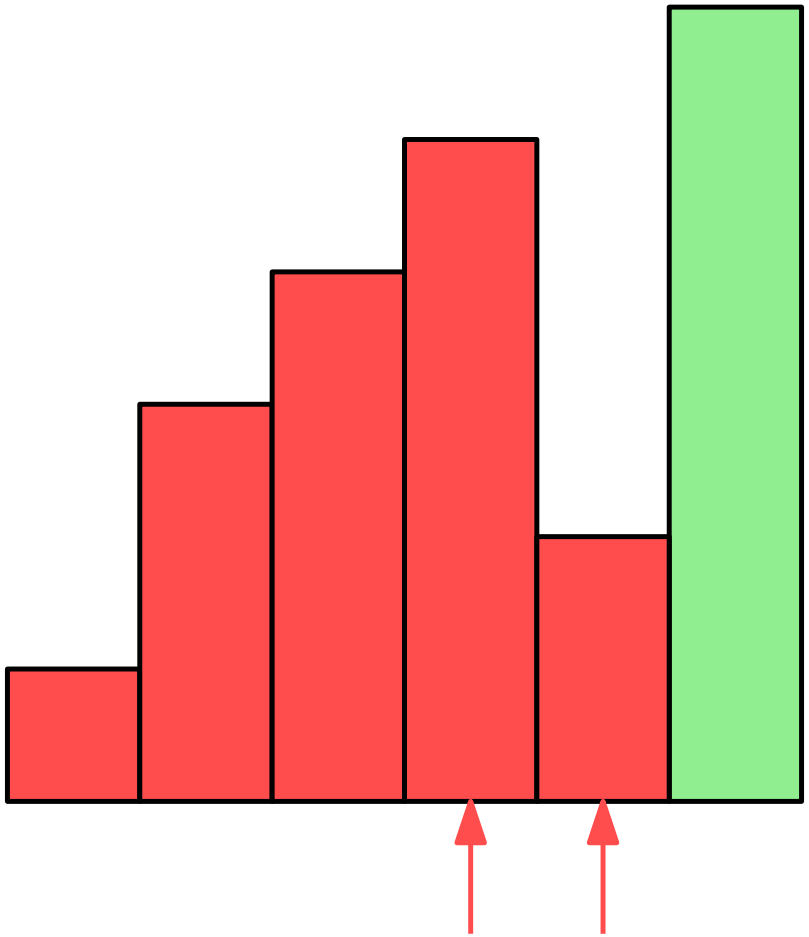
# Demo



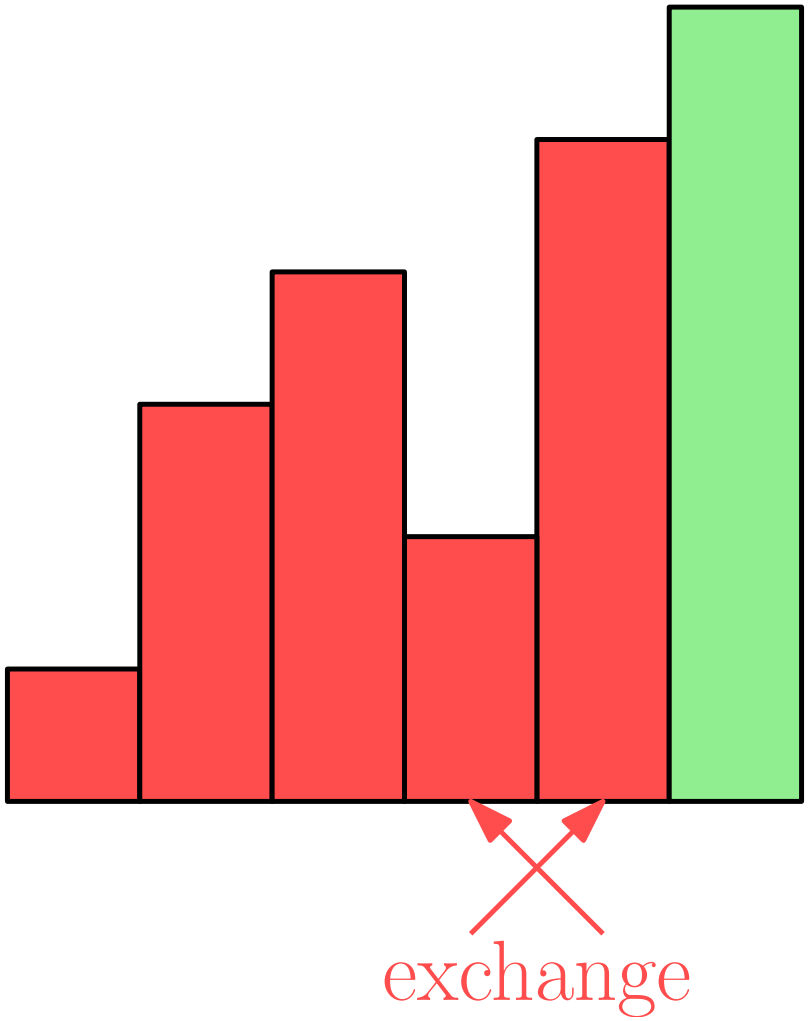
# Demo



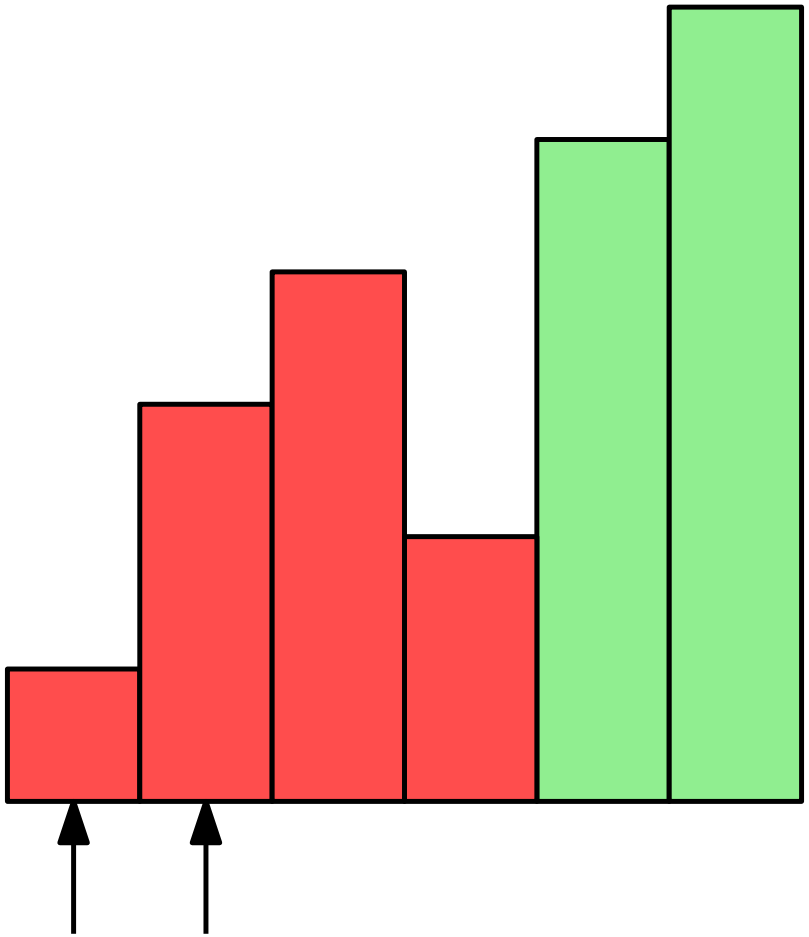
# Demo



# Demo

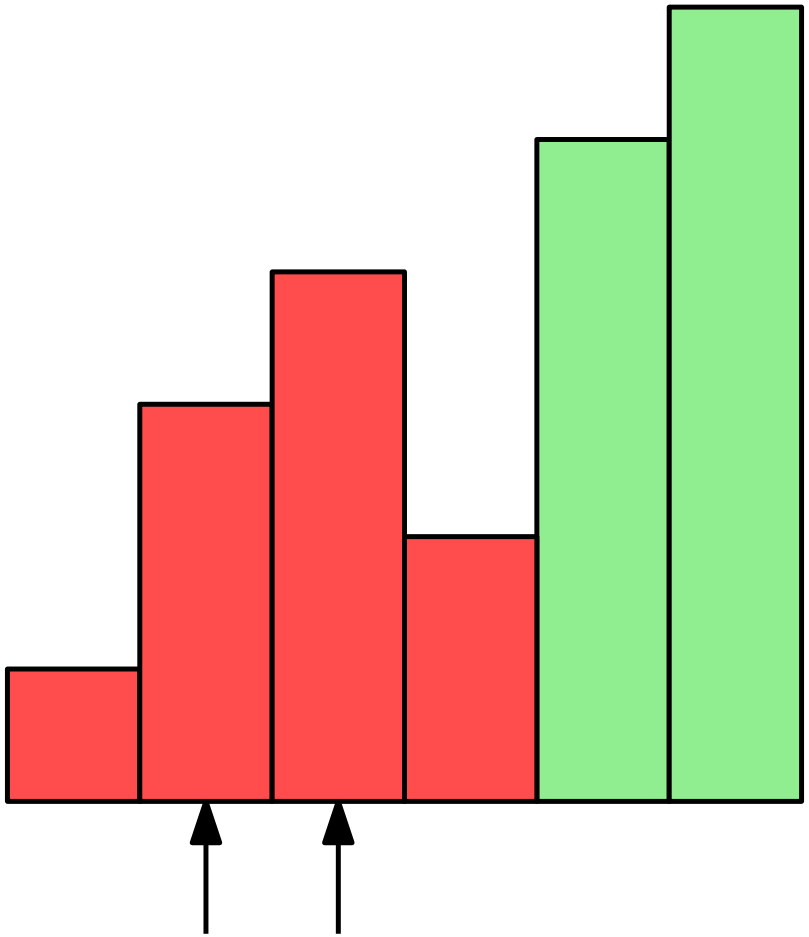


# Demo

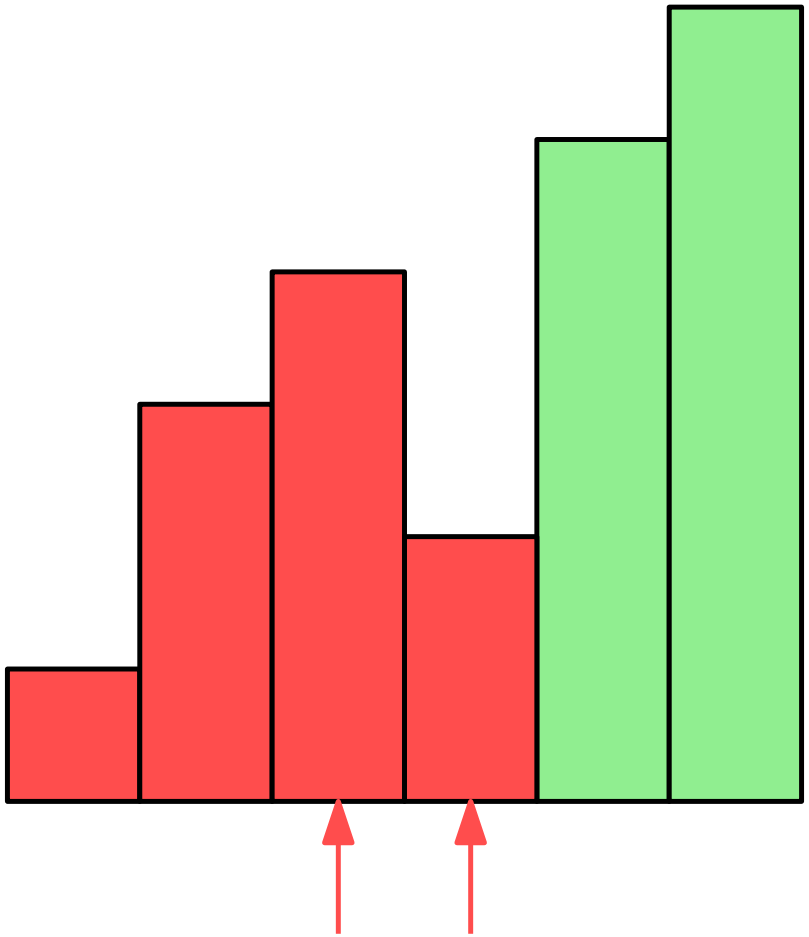


Pass 3 starts

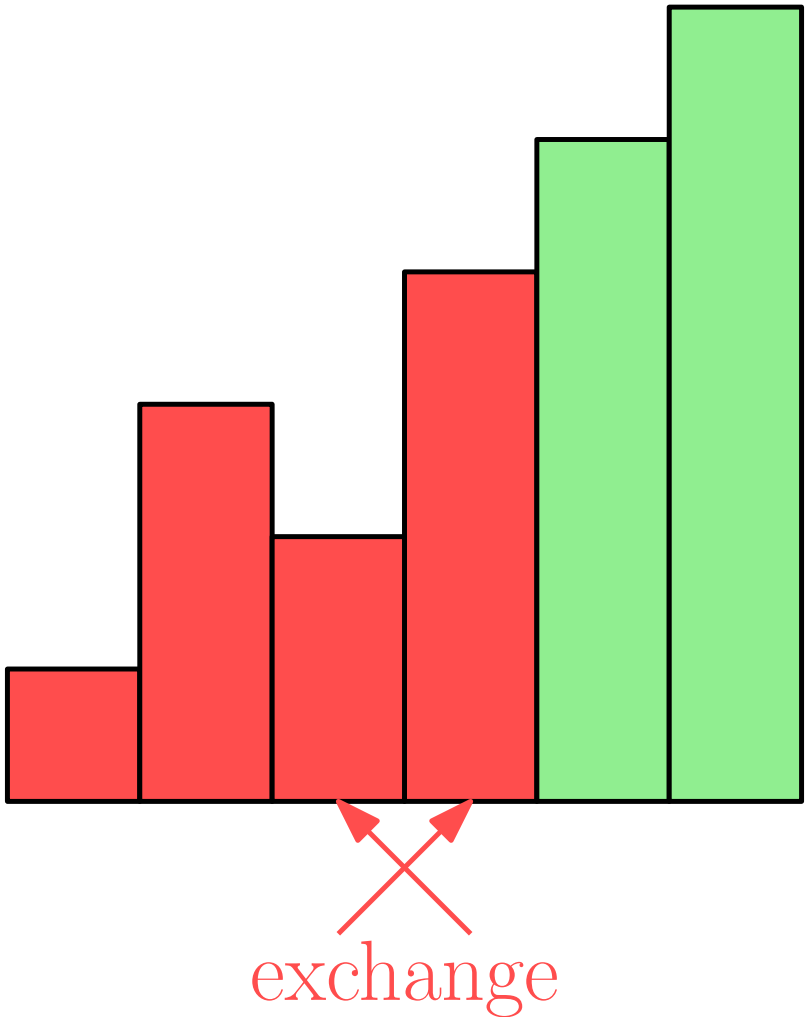
# Demo



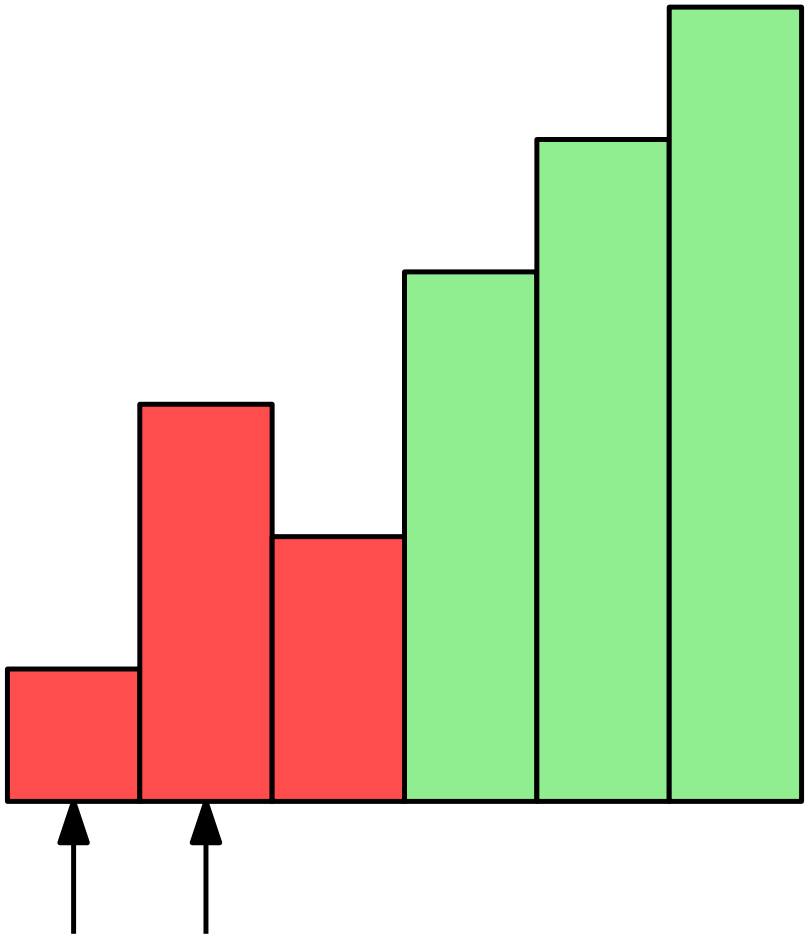
# Demo



# Demo

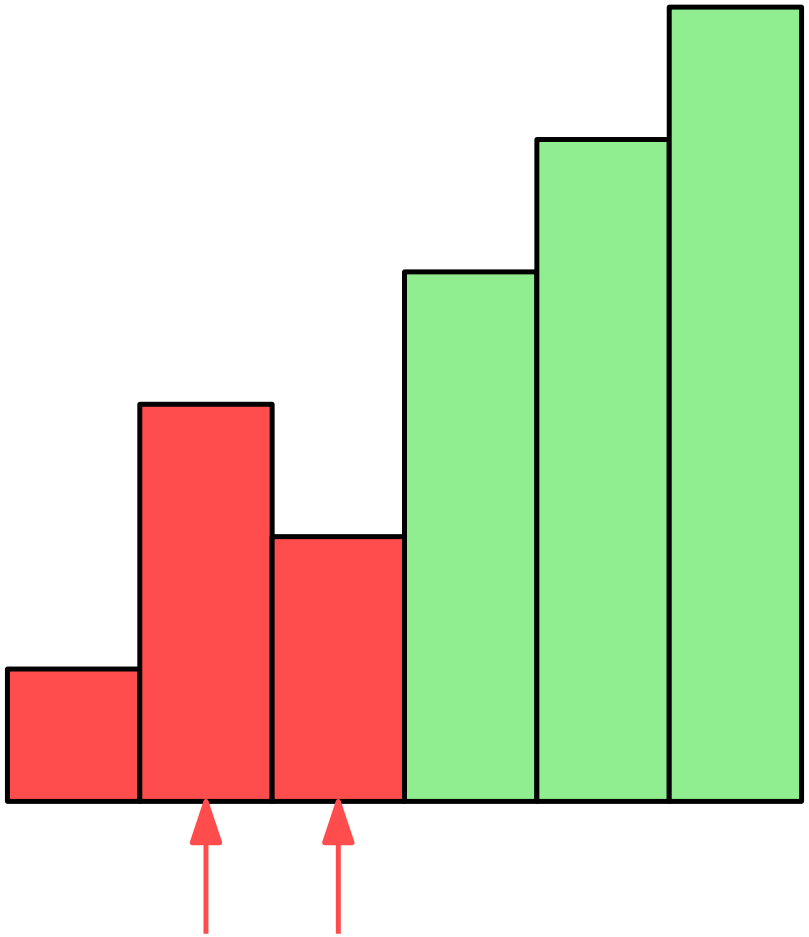


# Demo

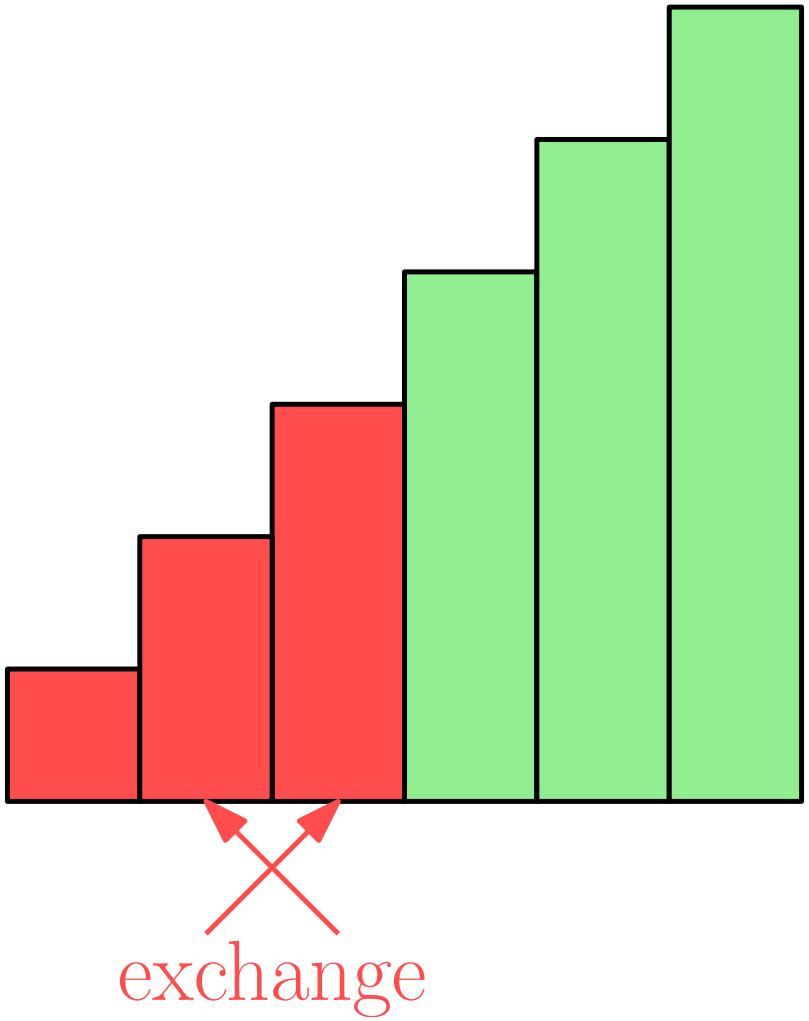


Pass 4 starts

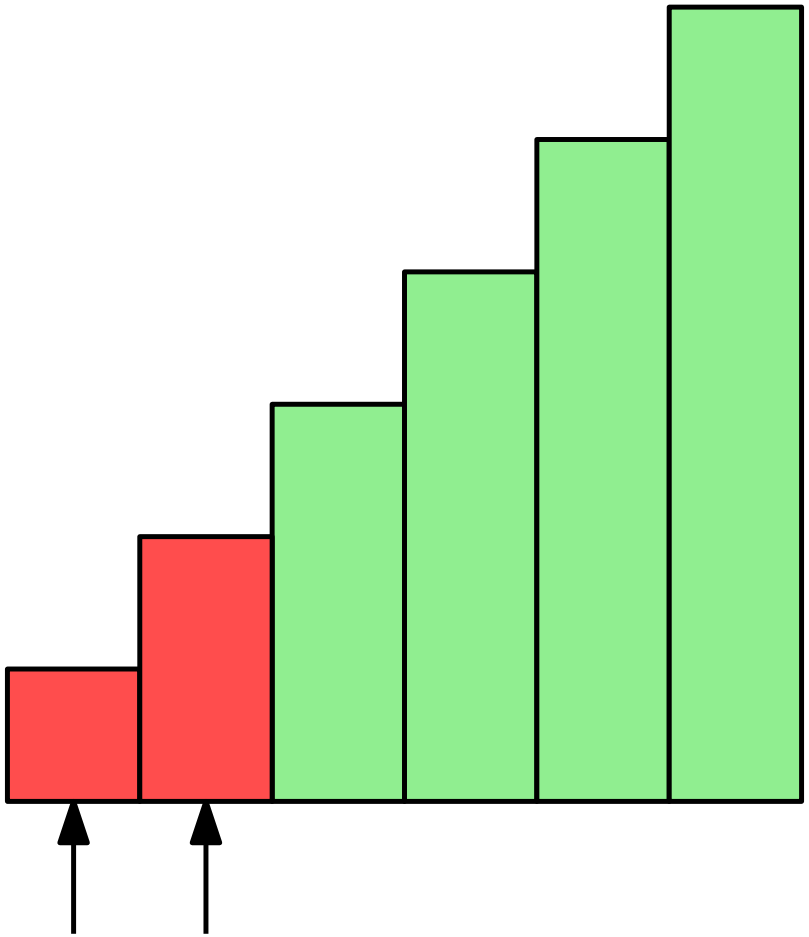
# Demo



# Demo

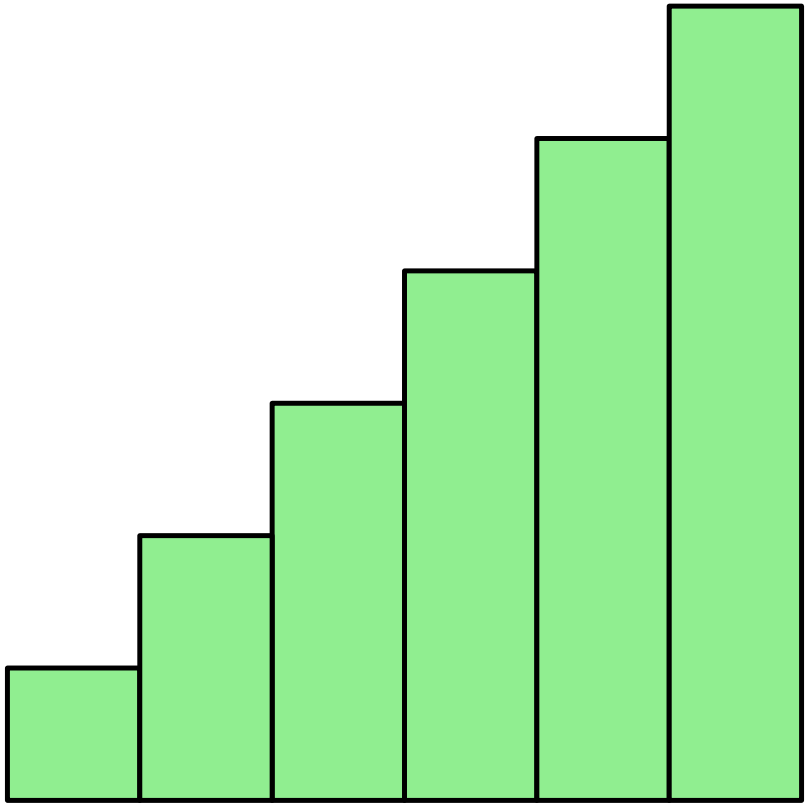


# Demo



Pass 5 starts

# Demo



End $\Rightarrow$ Sorted!

# Principle of Bubble Sort (1)

- Works in $n-1$ *passes*.
- In each pass, every two neighboring elements are compared, and they are swapped if out of order.
- In each pass, the largest of the elements under consideration will bubble up to the top (i.e., right).
- Elements under consideration in each pass is one less than elements in the previous pass.

# Principle of Bubble Sort (2)

```
void swap(int *x, int *y){
  int tmp = *x; *x = *y; *y = tmp;}

void bubbleSort (int x[], int n){
  int i,j;
  for (i=n-1; i>0; i--)
    for (j=0; j<i; j++)
      if (x[j] > x[j+1])
        swap(&x[j], &x[j+1]);
}
```

# Principle of Bubble Sort (3)

```
int main(){
  int x[]={-45,89,-65,87,0,3,-23,19,56,
           21,76,-50};
  int i, n=12;
  printf("\nBefore: ");
  for(i=0;i<n;i++) printf("%d ",x[i]);

  bubbleSort(x,n);
  printf("\nAfter : ");
  for(i=0;i<n;i++) printf("%d ",x[i]);
  return 0;
}
```

# Principle of Bubble Sort (4)

```
Before:
-45 89 -65 87 0 3 -23 19 56 21 76 -50
After:
-65 -50 -45 -23 0 3 19 21 56 76 87 89
```

## Time Complexity

Measured in terms of *number of comparisons.*

*Worst case:*
$(n-1)+(n-2)+\ldots+1=\frac{1}{2}n(n-1)$.
*Best case:* Same.

# Principle of Bubble Sort



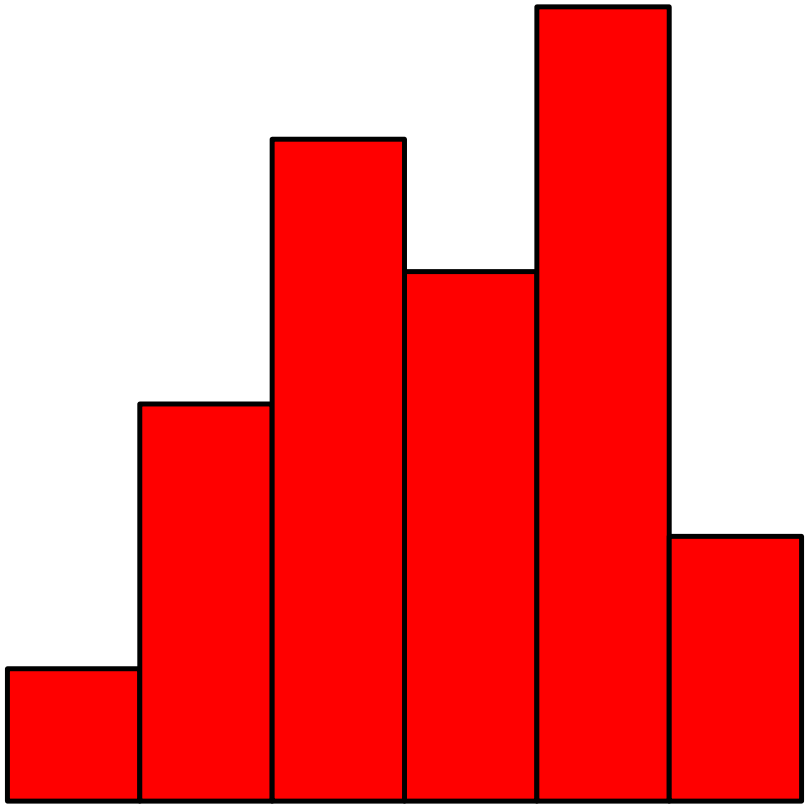
**How do you make best case with $n-1$ comparisons only?**

By maintaining a `flag`, to check if there has been any swap in a given pass. If not, the array is already sorted!

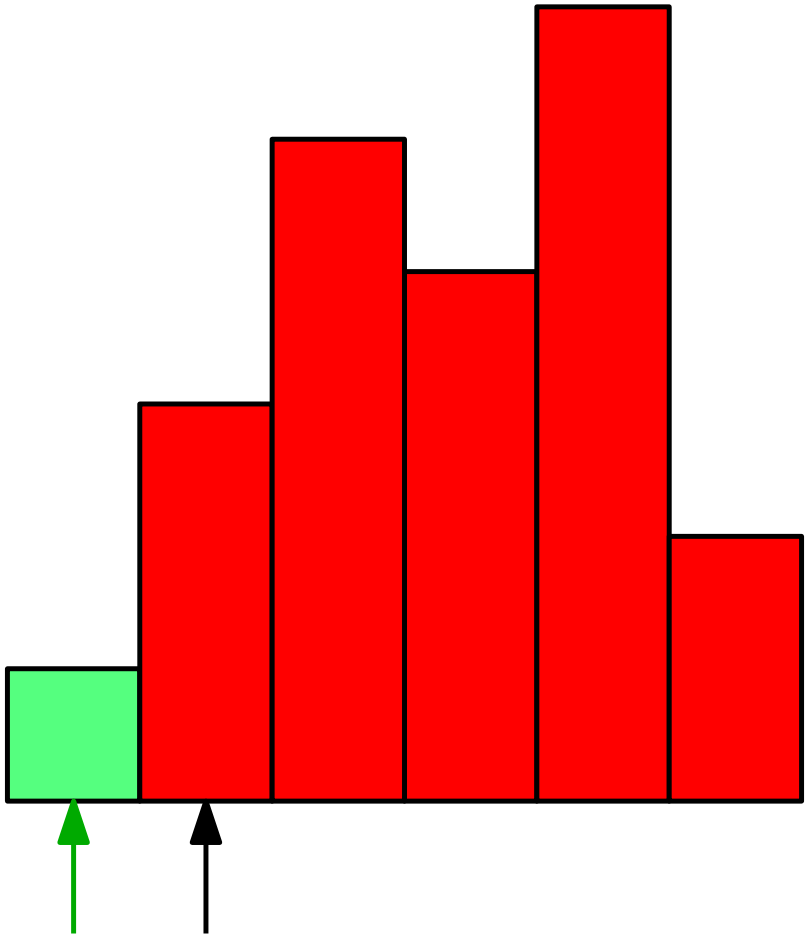
# Principle of Bubble Sort (6)

```
void bubbleSort(int x[], int n){
  int i, j, flag;
  for (i=n-1; i>0; i--){
    flag = 0;
    for (j=0; j<i; j++)
      if (x[j] > x[j+1]){
        swap(&x[j], &x[j+1]);
        flag = 1;}
    if (flag == 0) return;
  }//end outer for
}
```

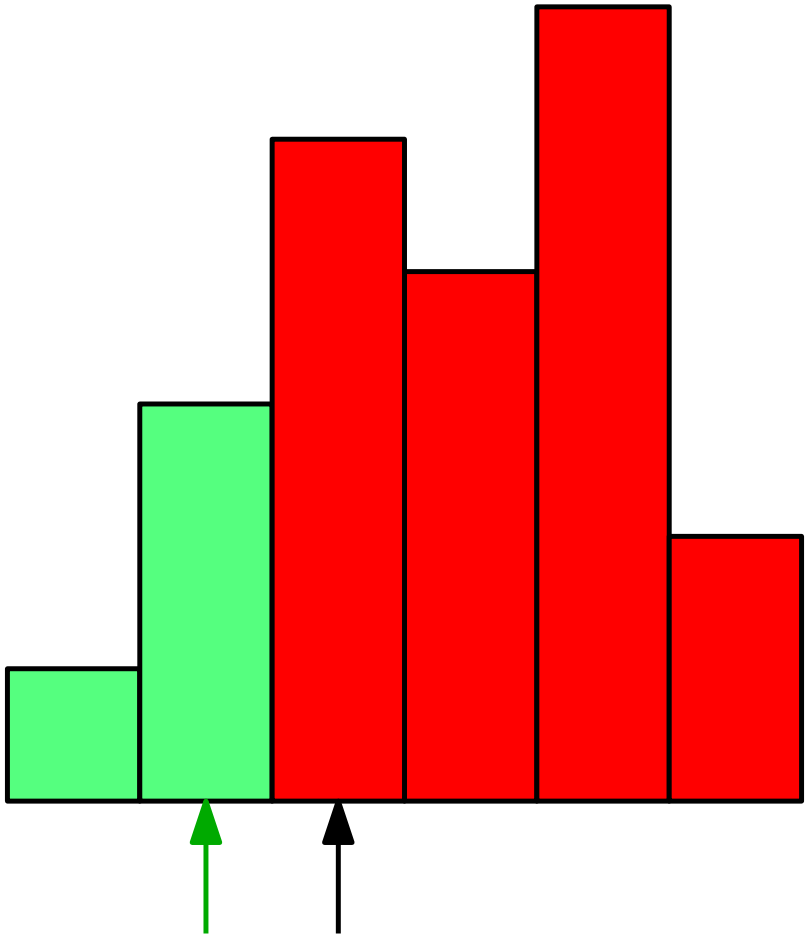
# Demo



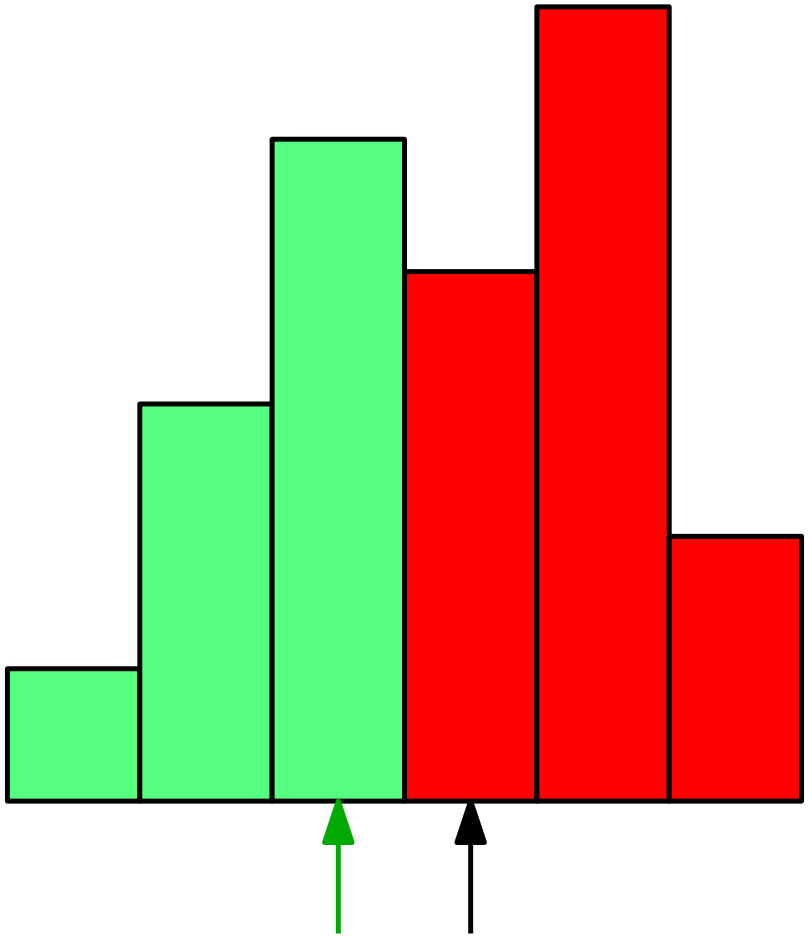
# Demo



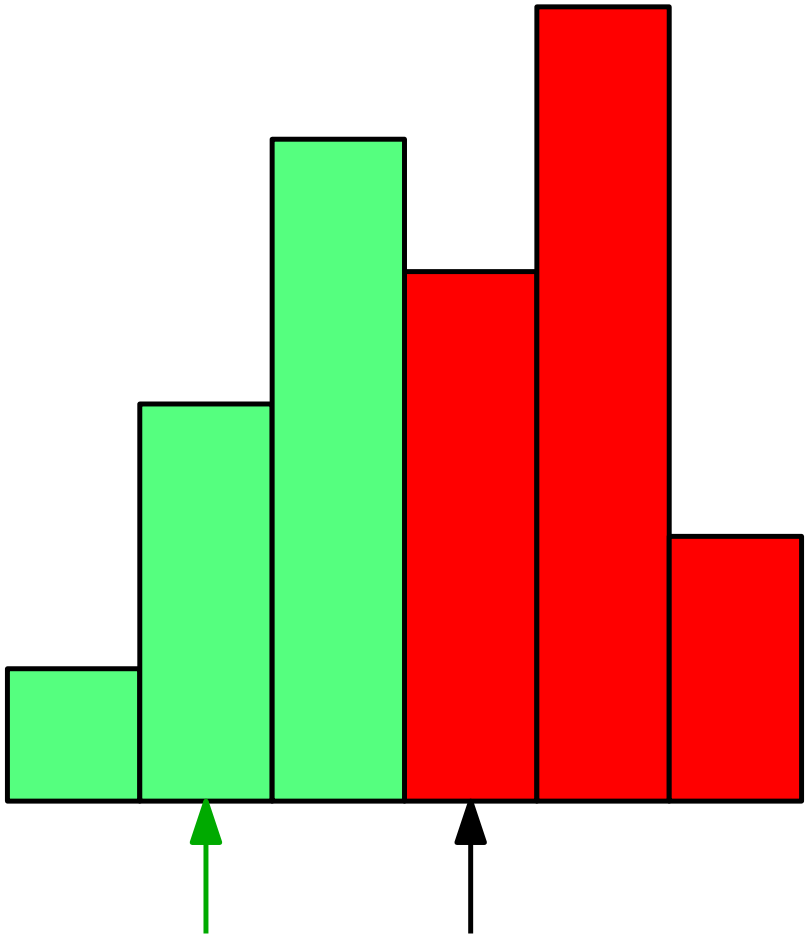
# Demo



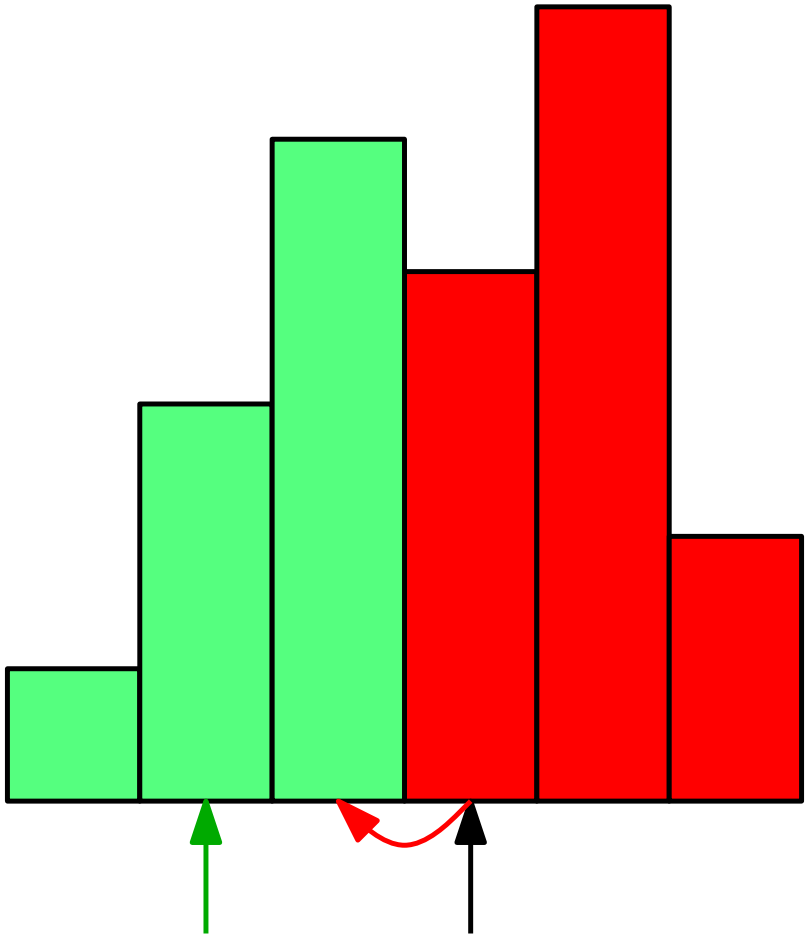
# Demo



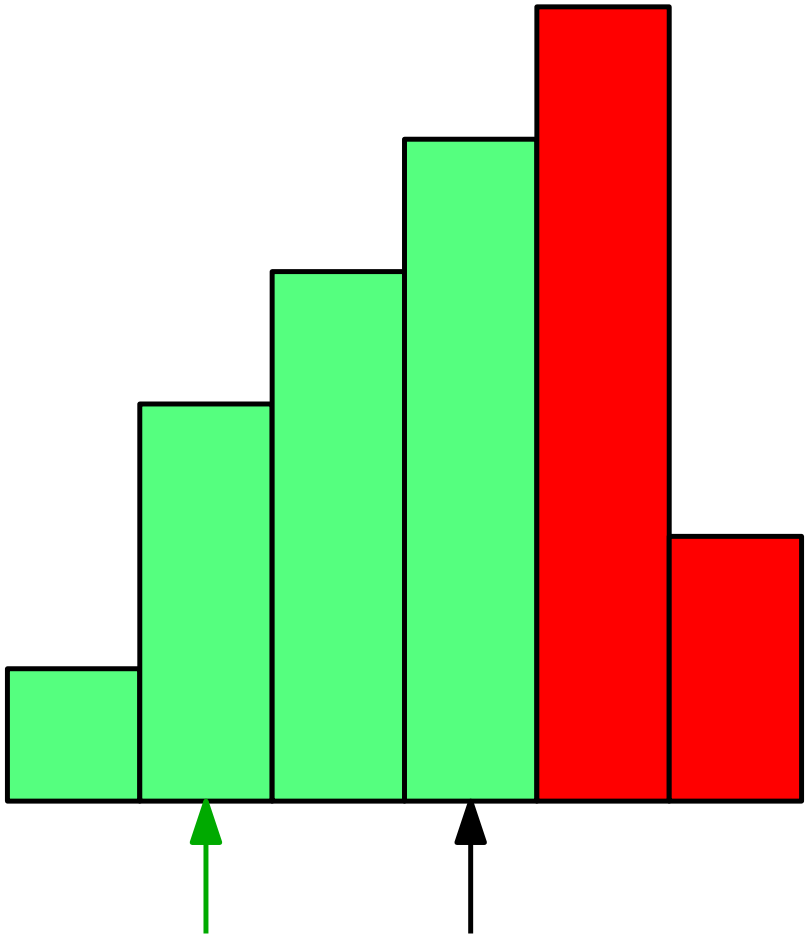
# Demo



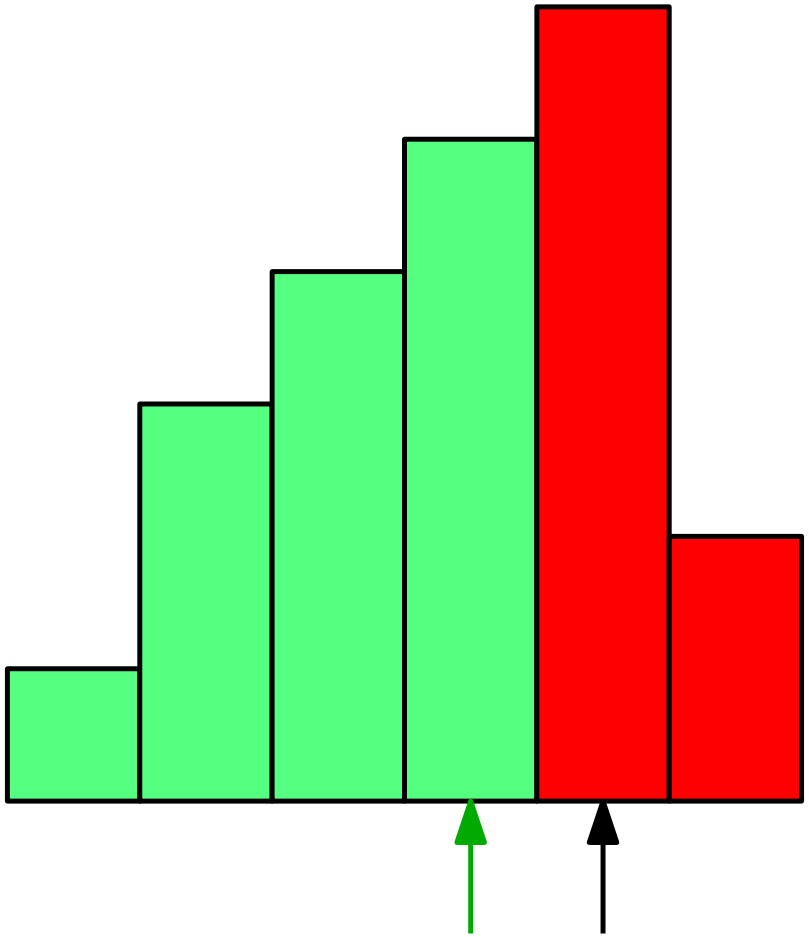
# Demo



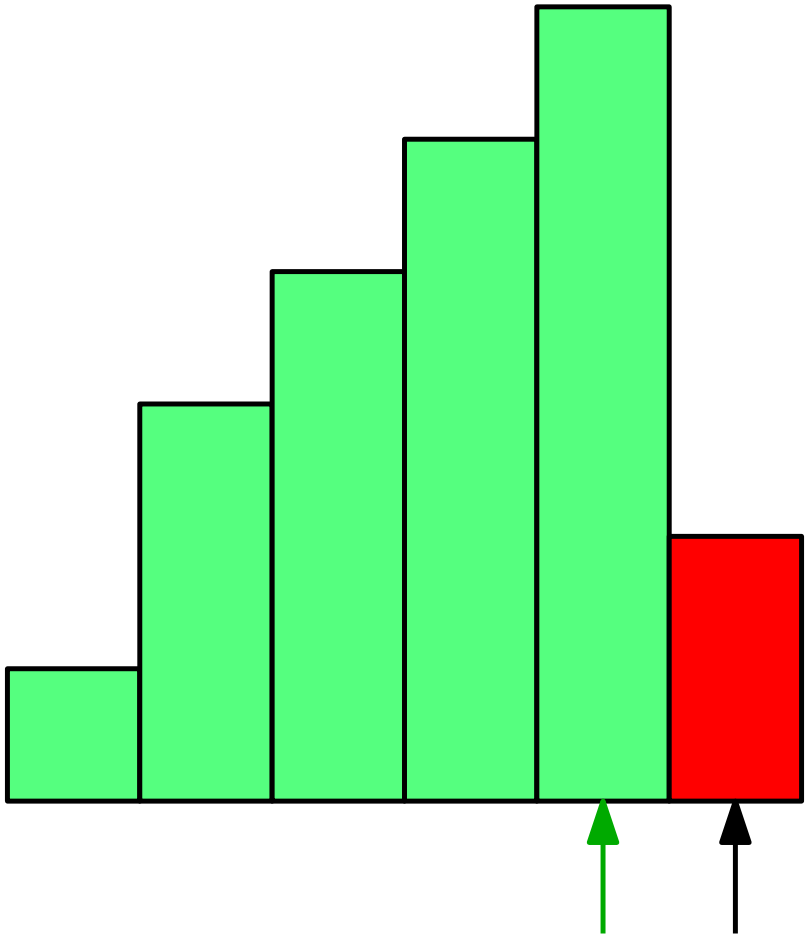
# Demo



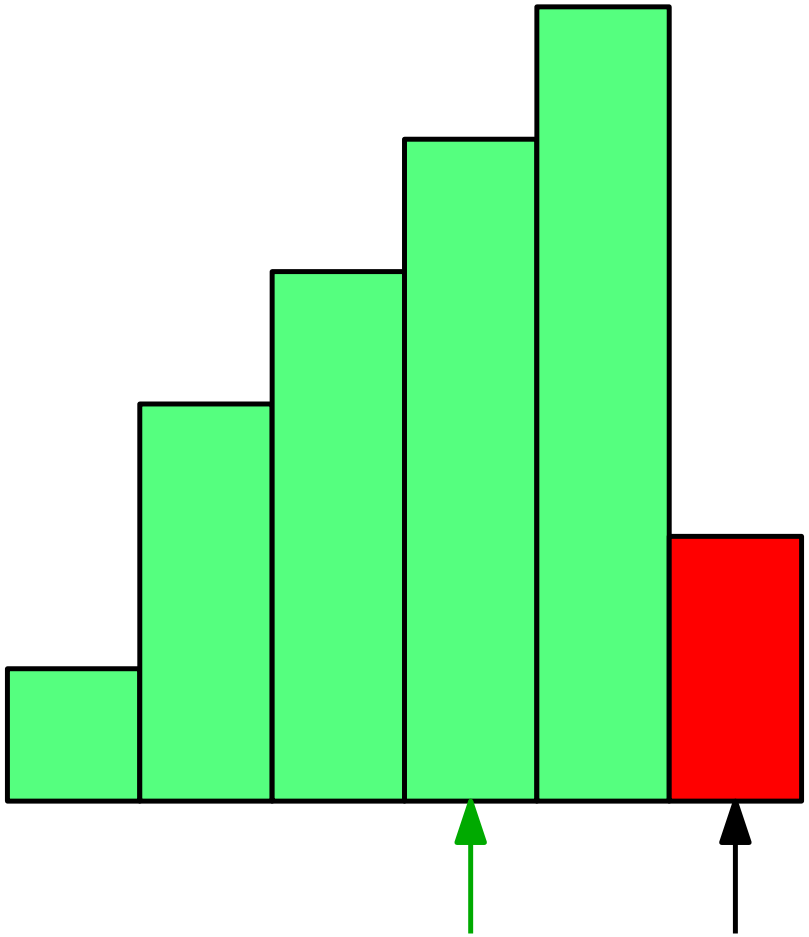
# Demo



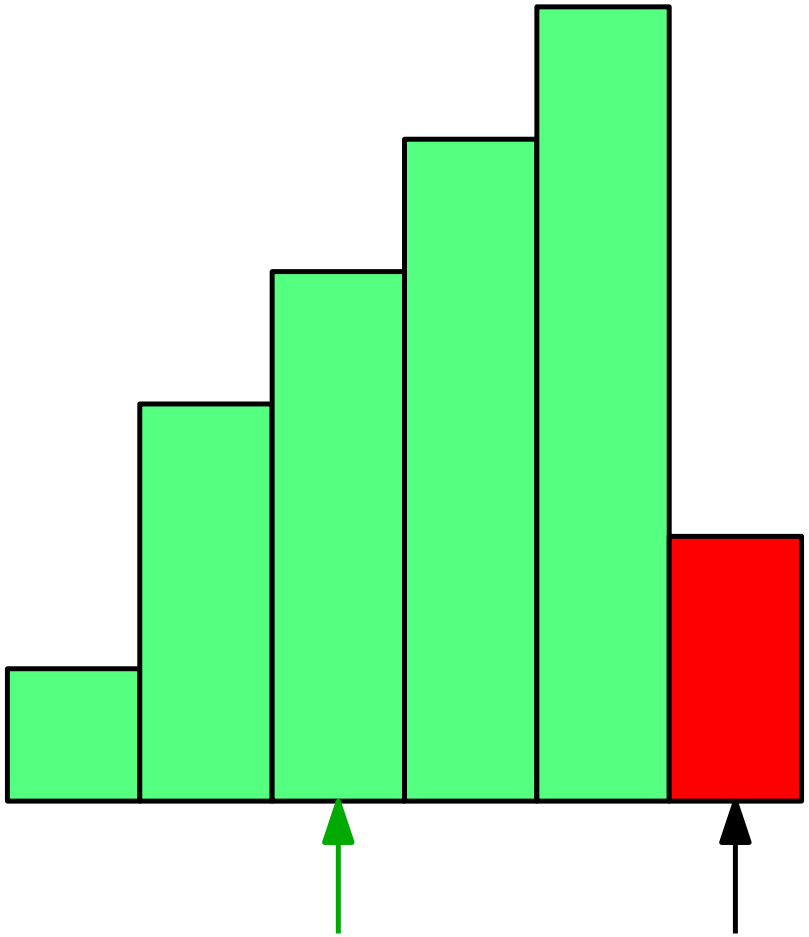
# Demo



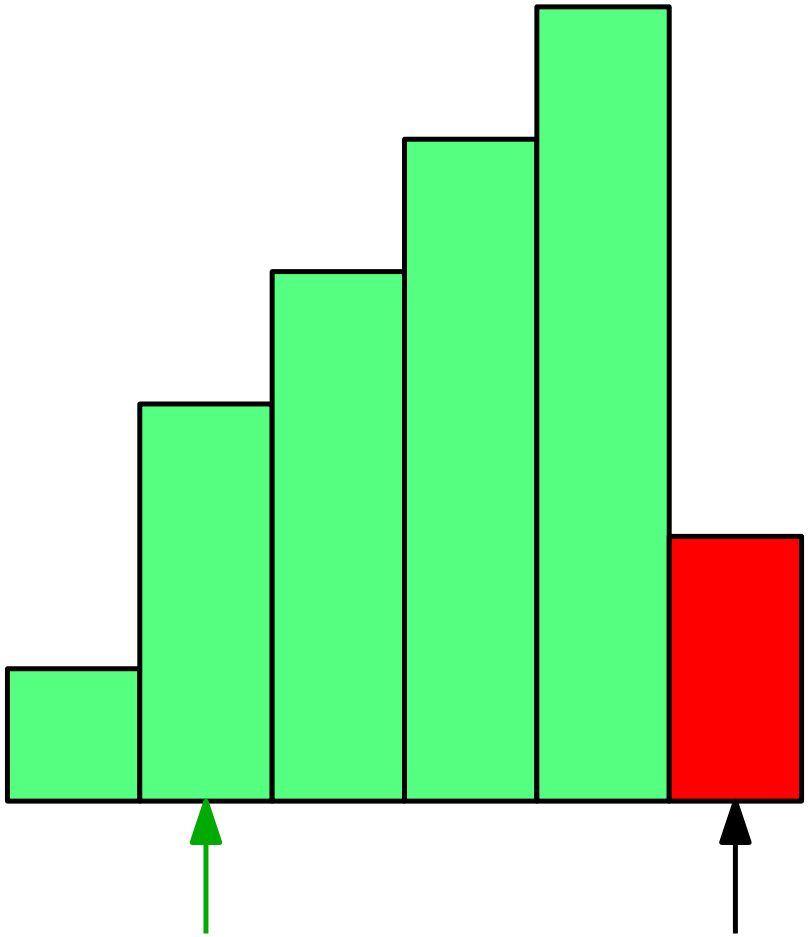
# Demo



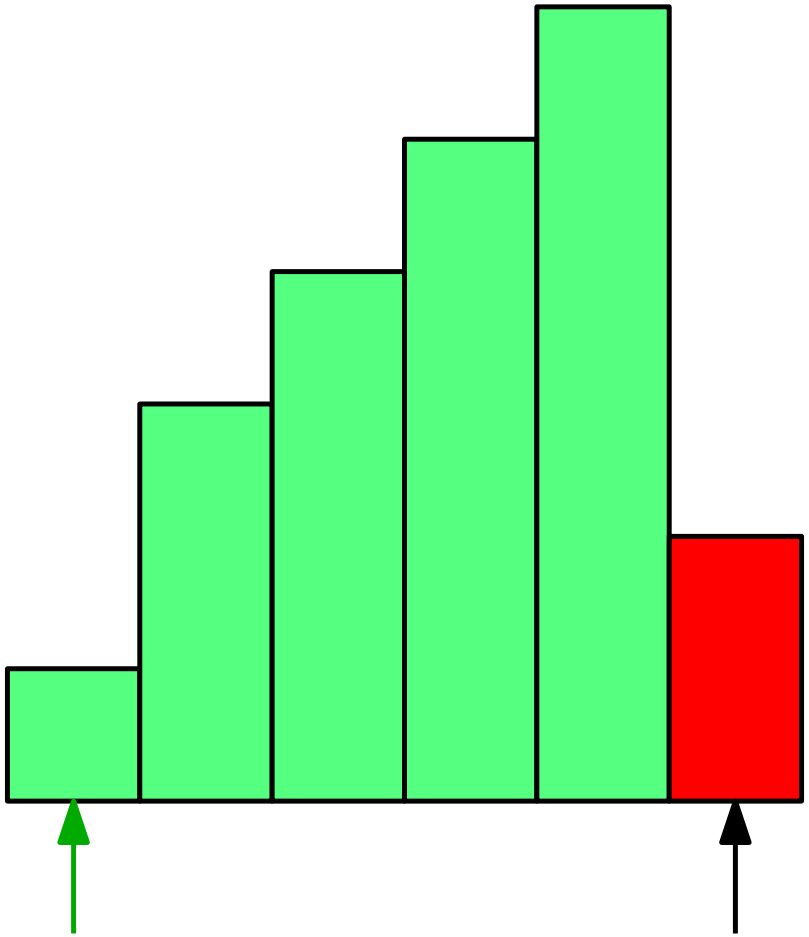
# Demo



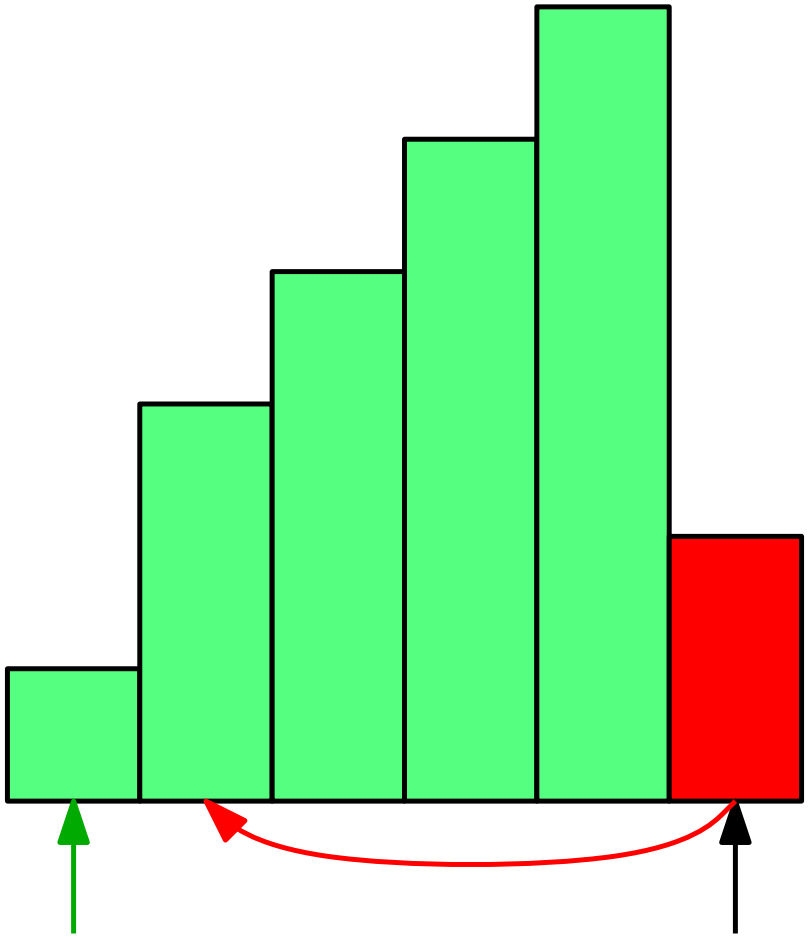
# Demo



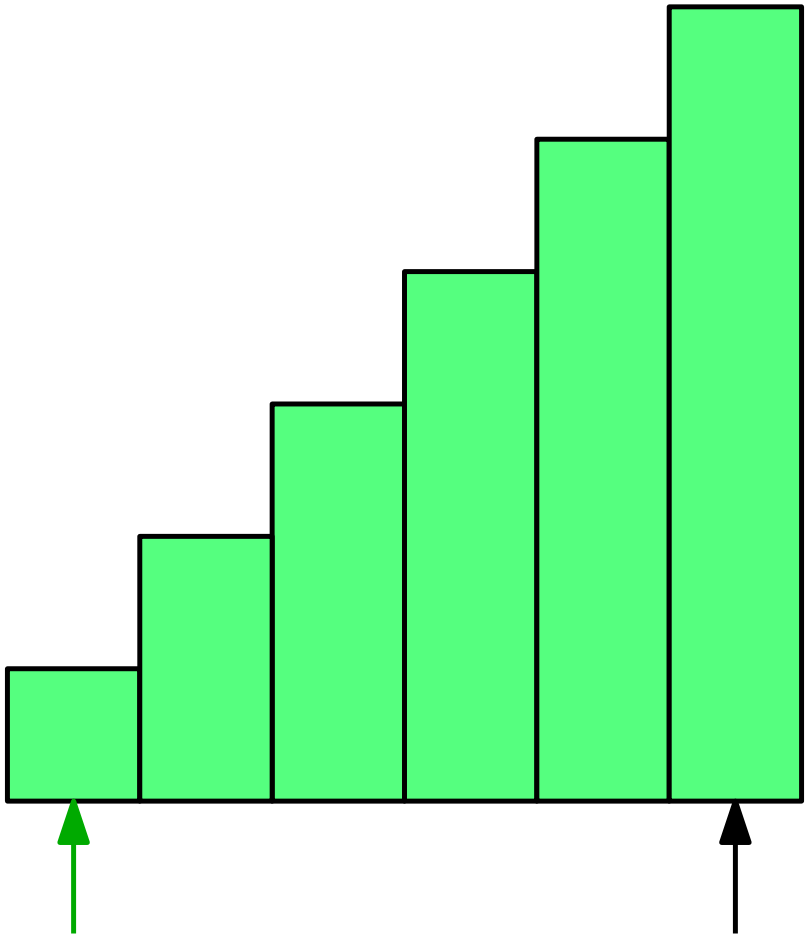
# Demo



# Demo



# Demo



End $\Rightarrow$ Sorted!

# Principle of Insertion Sort (1)

- Works in $n-1$ *passes*.
- On completion of `i`th pass, the first `i + 1` elements are sorted.
- In `i`th pass, the current element `x[i]` is compared with each `x[j]` for `j` $=$ `i` $-$ `1`, `i` $-$ `2`, $\ldots$, `0`, until some `x[j]` is smaller than `x[i]`. Then `x[i]` is inserted at (`j + 1`)th position.
- Elements under consideration in each pass is one more than elements in the previous pass.

# Principle of Insertion Sort (2)

```
#include <stdio.h>

void insSort (int x[], int n){
  int i, j, cur;
  for (i=1; i<n; i++){
    cur = x[i];
    for (j=i-1; (j>=0) && (x[j]>cur); j--)
      x[j+1] = x[j];
    x[j+1] = cur;
  }
}
```

# Principle of Insertion Sort (3)

```
int main(){
  int x[]={-45,89,-65,87,0,3,-23,19,56,
           21,76,-50}, i;
  printf("Before:\n");
  for(i=0;i<12;i++) printf("%d ",x[i]);
  printf("\n");
  insSort(x,12);
  printf("After:\n");
  for(i=0;i<12;i++) printf("%d ",x[i]);
  printf("\n");
  return 0;
}
```

# Principle of Insertion Sort (4)

```
Before:
-45 89 -65 87 0 3 -23 19 56 21 76 -50
After:
-65 -50 -45 -23 0 3 19 21 56 76 87 89
```

## Time Complexity

Measured in terms of *number of comparisons.*

*Worst case:* $1 + 2 + \ldots + (n-1) = \frac{1}{2}n(n-1)$.
*Best case:* $1 + 1 + \ldots n - 1$ times $= n - 1$.

# Other Sorting Algorithms (1)

- **Selection Sort:** Finds the maximum of `n` elements and sends it to `x[n-1]`, then finds the maximum of first `n-1` elements and sends it to `x[n-2]`, and so on.

# Other Sorting Algorithms (2)

- **Merge Sort:** Compares the elements *pairwise* (`x[0]` with `x[1]`, `x[2]` with `x[3]`, ...) and swaps them if not in order. It then recursively *merges* each sorted sub-list of size 2 into lists of size 4, then sub-lists of size 4 into sub-lists of size 8, and so on, to get the final sorted list.

  Requires extra space of size $n$.

  *Worst-case runtime:* $O(n \log n)$.

# Other Sorting Algorithms (3)

- **Quicksort:** A *divide and conquer* algorithm.
  Based on *partition* using a *pivot*.
  All elements smaller than the pivot are moved before it and all greater elements are moved after it.
  The lesser and greater sub-lists are then recursively sorted.

  *Average-case runtime:* $O(n \log n)$.

# Exercise (1)

- Write a C program to read `struct` data from a file and store it in a linked list, so that the stored data is sorted in non-decreasing order of the *key* values.
- Write a C program to first read `struct` data from a file and store it in a linked list. Now apply *insertion sort* on the linked list to sort the data in non-decreasing order of the *key* values.