INTRODUCTION TO SORTING

Data Structures and Algorithms

Waheed Iqbal

Department of Data Science, FCIT
University of the Punjab, Lahore, Pakistan

Socks Pairing Problem

Bubble Sort

9, 6, 2, 12, 11, 9, 3, 7

6, 9, 2, 12, 11, 9, 3, 7

6, 2, 9, 12, 11, 9, 3, 7

6, 2, 9, 12, 11, 9, 3, 7

6, 2, 9, 11, 12, 9, 3, 7

6, 2, 9, 11, 9, 12, 3, 7

6, 2, 9, 11, 9, 3, 12, 7

6, 2, 9, 11, 9, 3, 7, 12

Bubble Sort Example

First Pass	6, 2, 9, 11, 9, 3, 7, 12
Second Pass	2, 6, 9, 9, 3, 7, 11, 12
Third Pass	2, 6, 9, 3, 7, 9, 11, 12
Fourth Pass	2, 6, 3, 7, 9, 9, 11, 12
Fifth Pass	2, 3, 6, 7, 9, 9, 11, 12
Sixth Pass	2, 3, 6, 7, 9, 9, 11, 12

Bubble Sort: Few Questions

1. Which number is definitely in its correct position at the end of the first pass?

2. How does the number of comparisons required change as the pass number increases?

3. How does the algorithm know when the list is sorted?

4. What is the maximum number of comparisons required for a list of 10 numbers?

Bubble Sort: Few Questions

1. Which number is definitely in its correct position at the end of the first pass?

Answer: The last number must be the largest.

2. How does the number of comparisons required change as the pass number increases?

Answer: Each pass requires one fewer comparison than the last.

3. How does the algorithm know when the list is sorted?

Answer: When a pass with no exchanges occurs.

4. What is the maximum number of comparisons required for a list of 10 numbers?

Answer: 9 comparisons, then 8, 7, 6, 5, 4, 3, 2, 1 so total 45

Bubble Sort Code

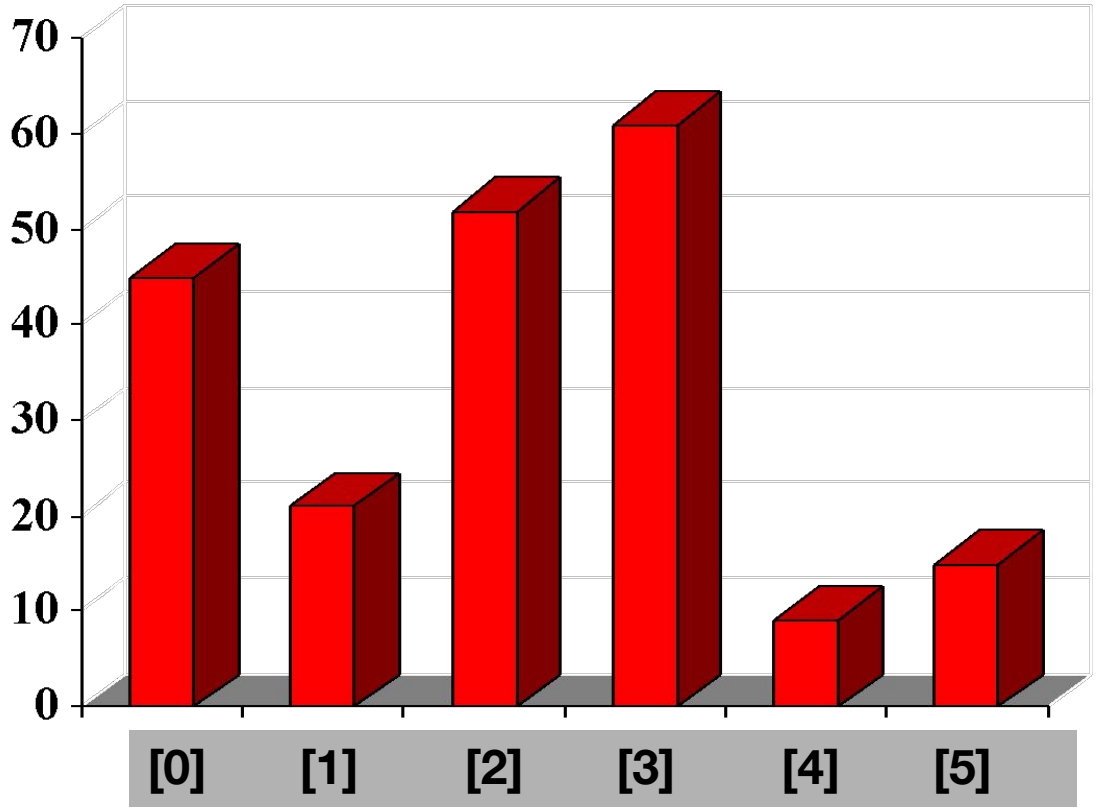
```python
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Example usage
arr = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(arr)
print("Sorted array is:", arr)
```

What is a Big-Oh of this code?

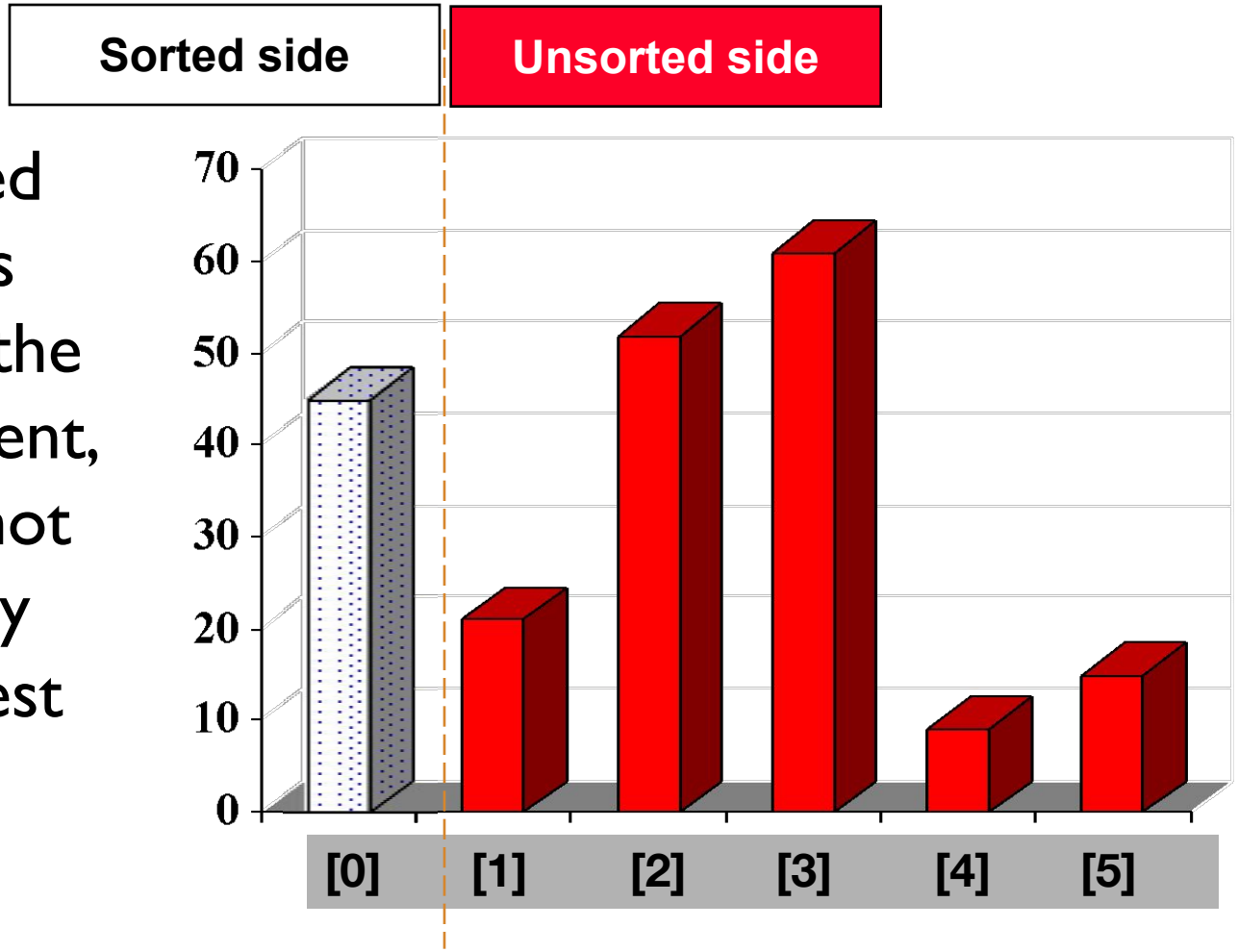
The Insertion Sort Algorithm

- The Insertion Sort algorithm also views the array as having a sorted side and an unsorted side.



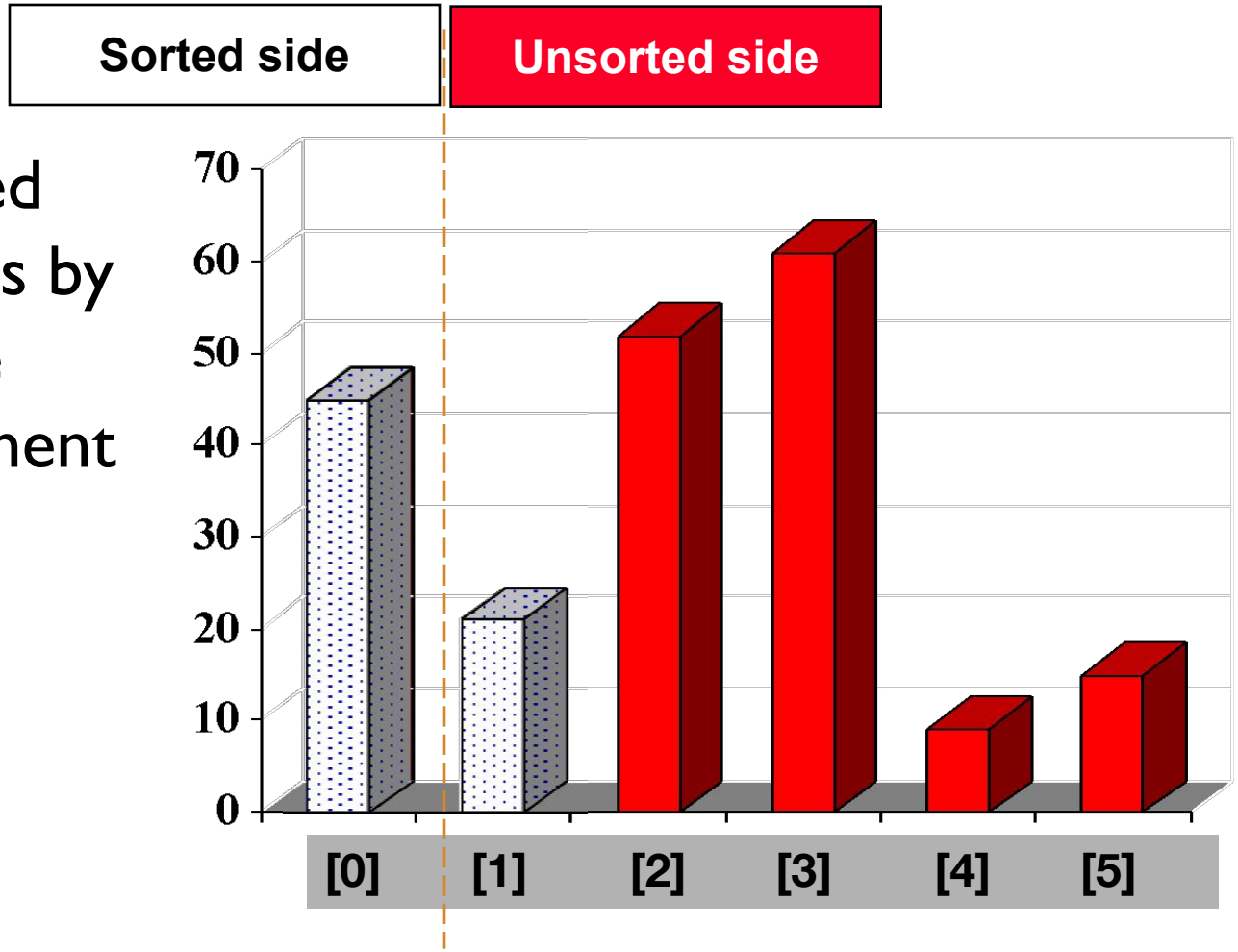
Insertion Sort Algorithm

- The sorted side starts with just the first element, which is not necessarily the smallest element.



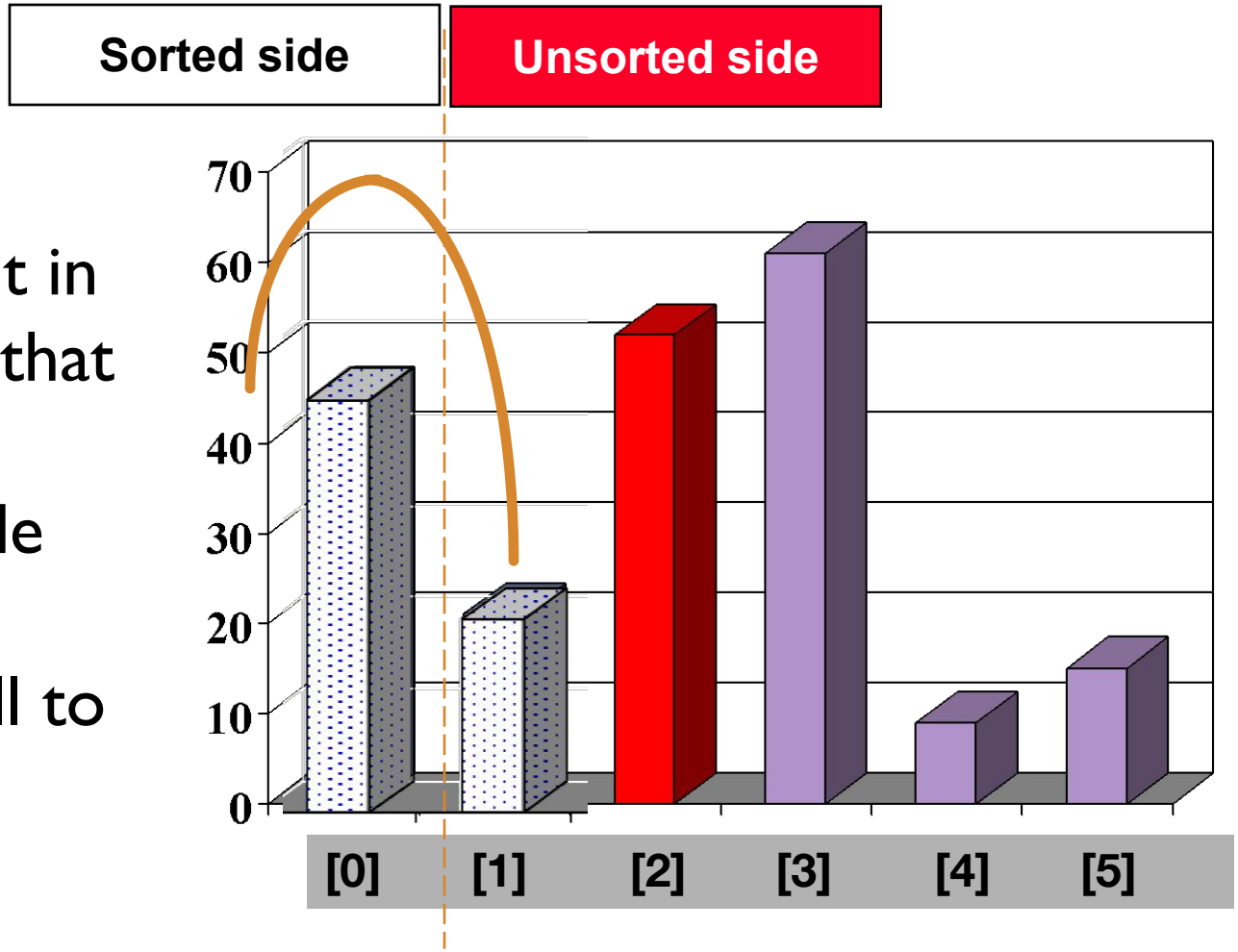
Insertion Sort Algorithm

- The sorted side grows by taking the front element from the unsorted side...

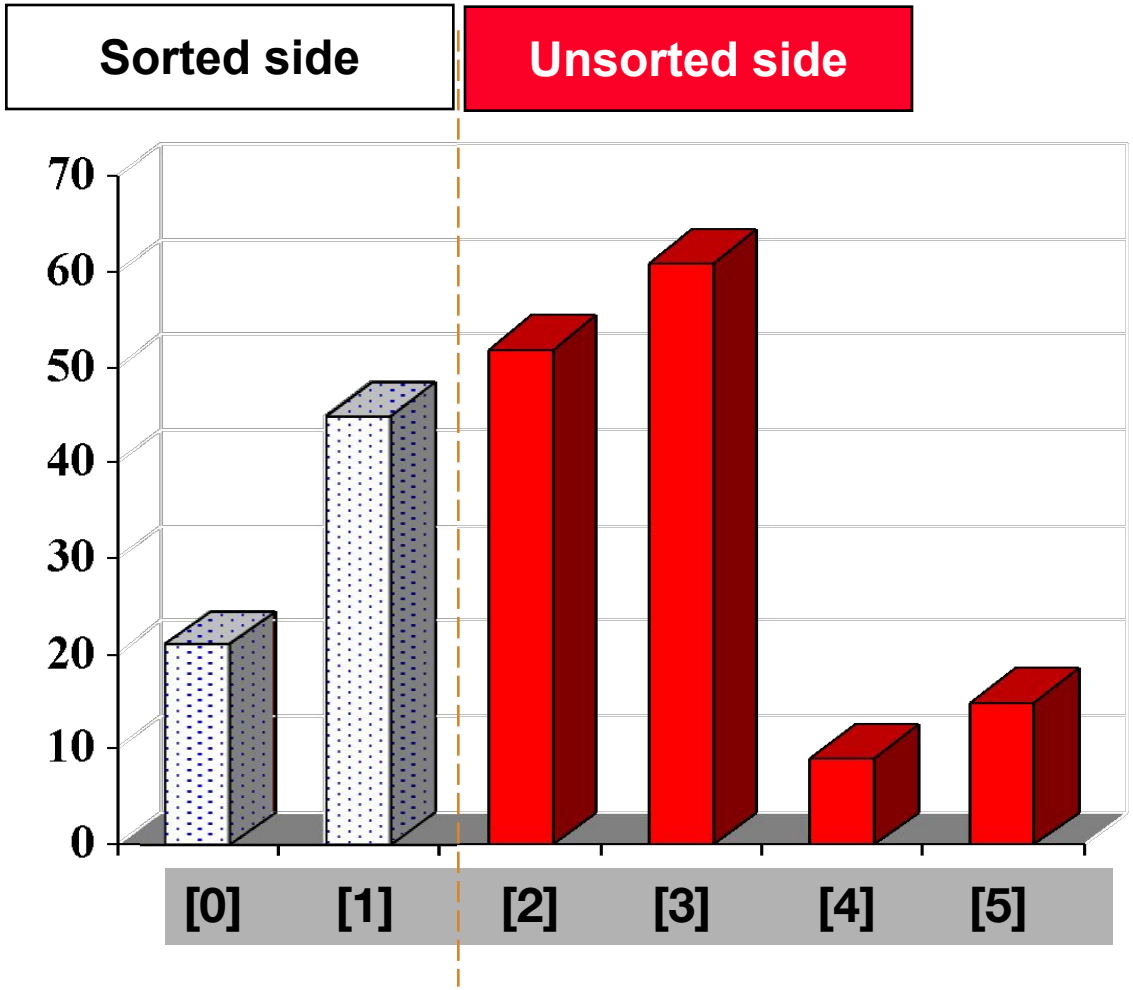


Insertion Sort Algorithm

- ...and inserting it in the place that keeps the sorted side arranged from small to large.

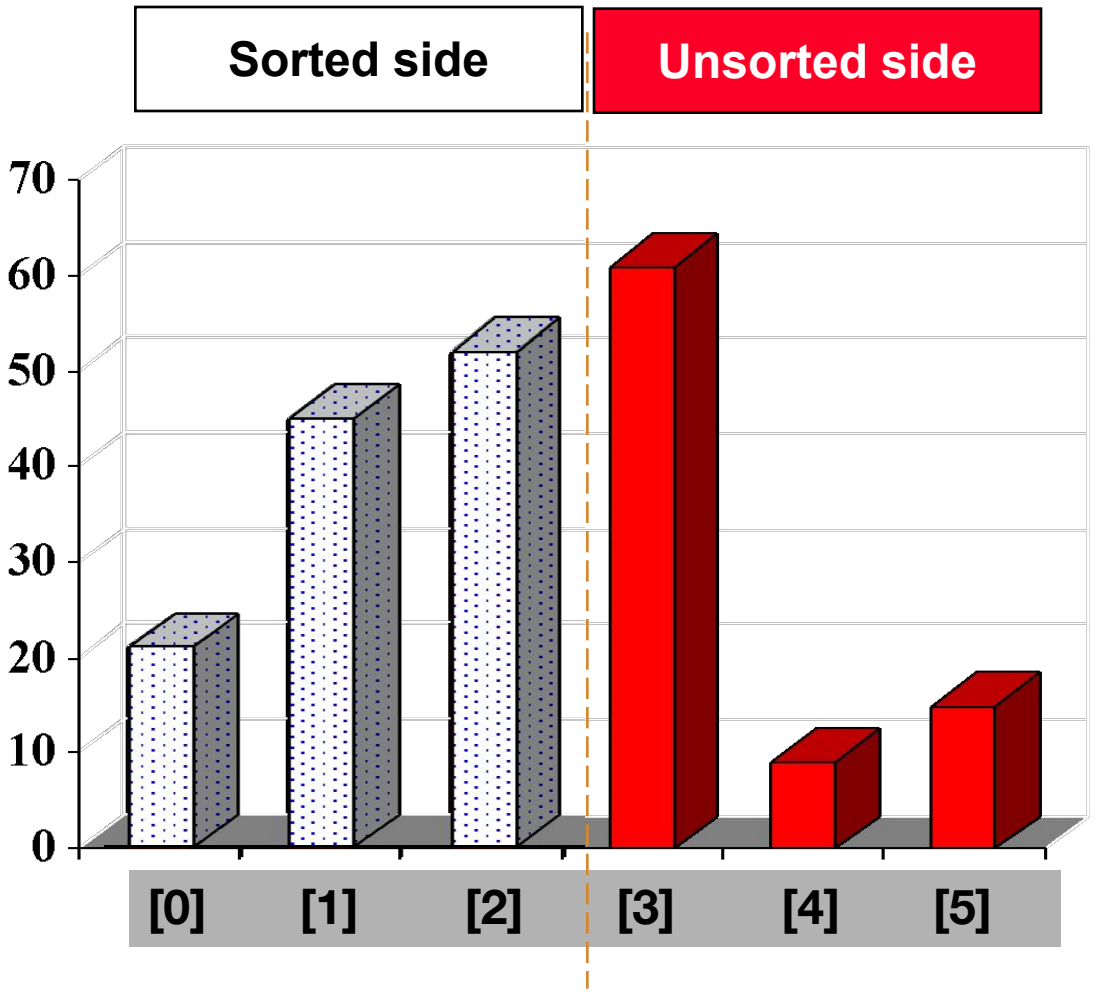


Insertion Sort Algorithm



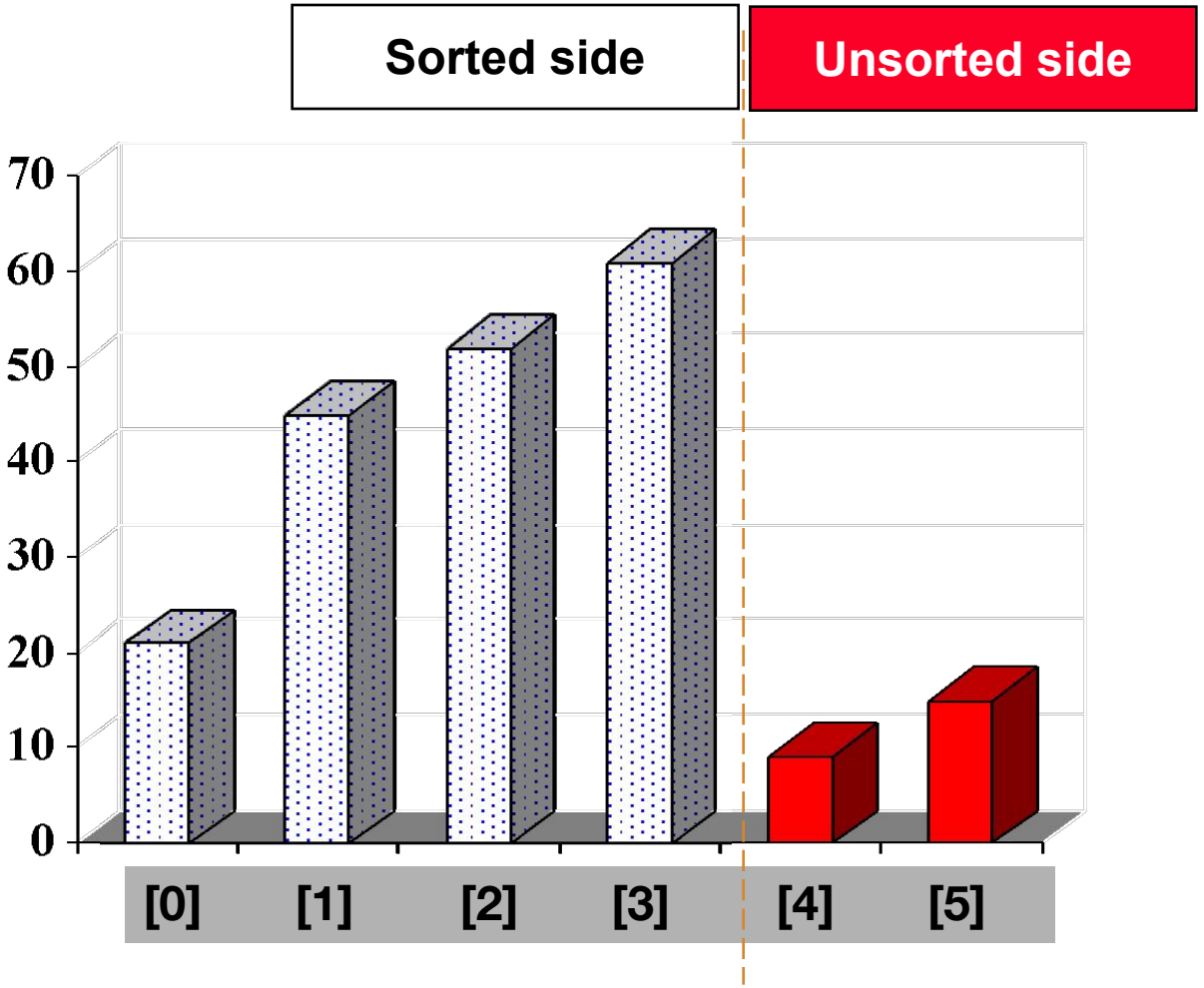
Insertion Sort Algorithm

- Sometimes we are lucky and the new inserted item doesn't need to move at all.



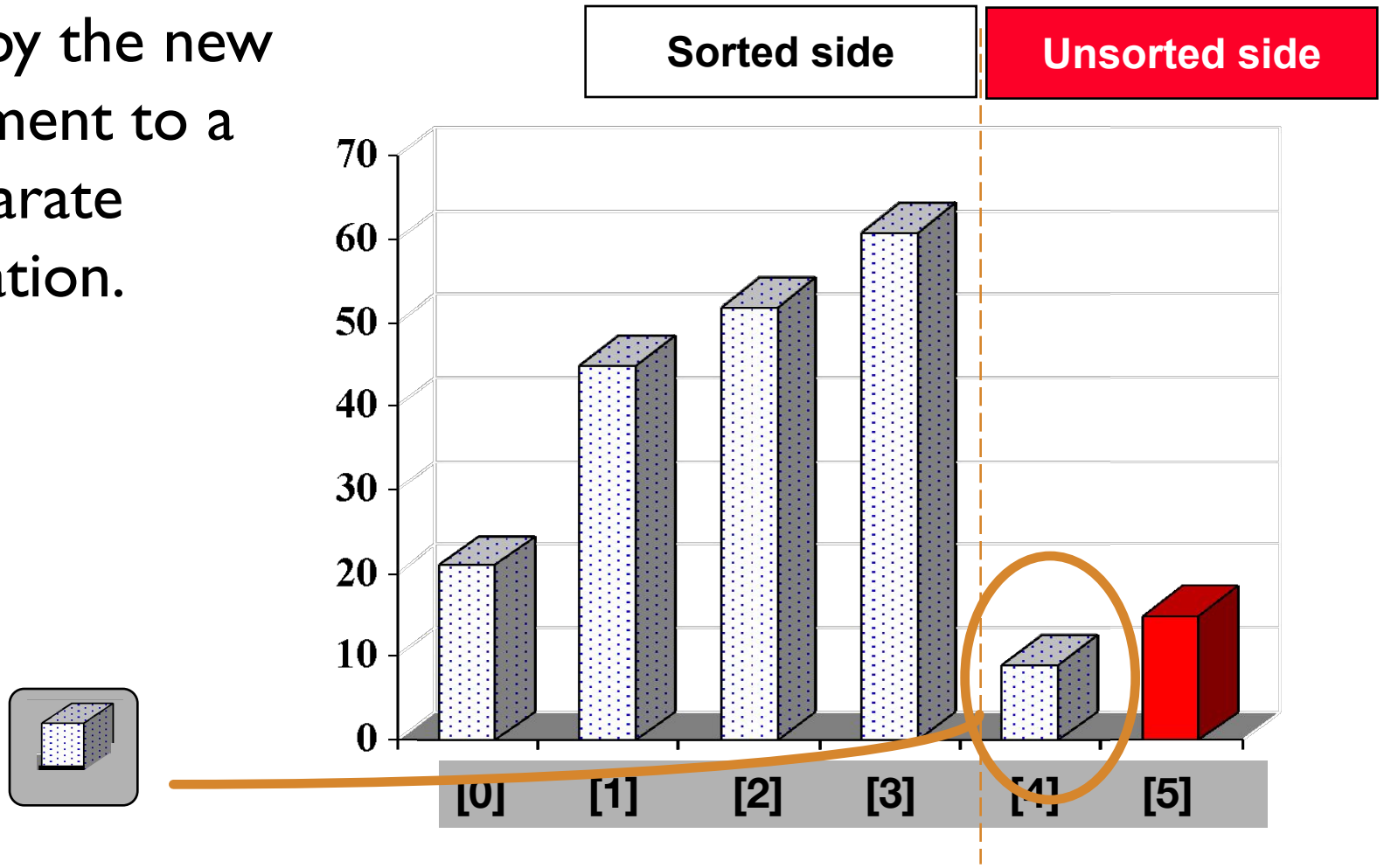
Insertion Sort Algorithm

- Sometimes we are lucky twice in a row.



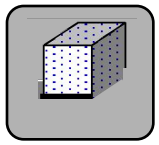
How to Insert One Element

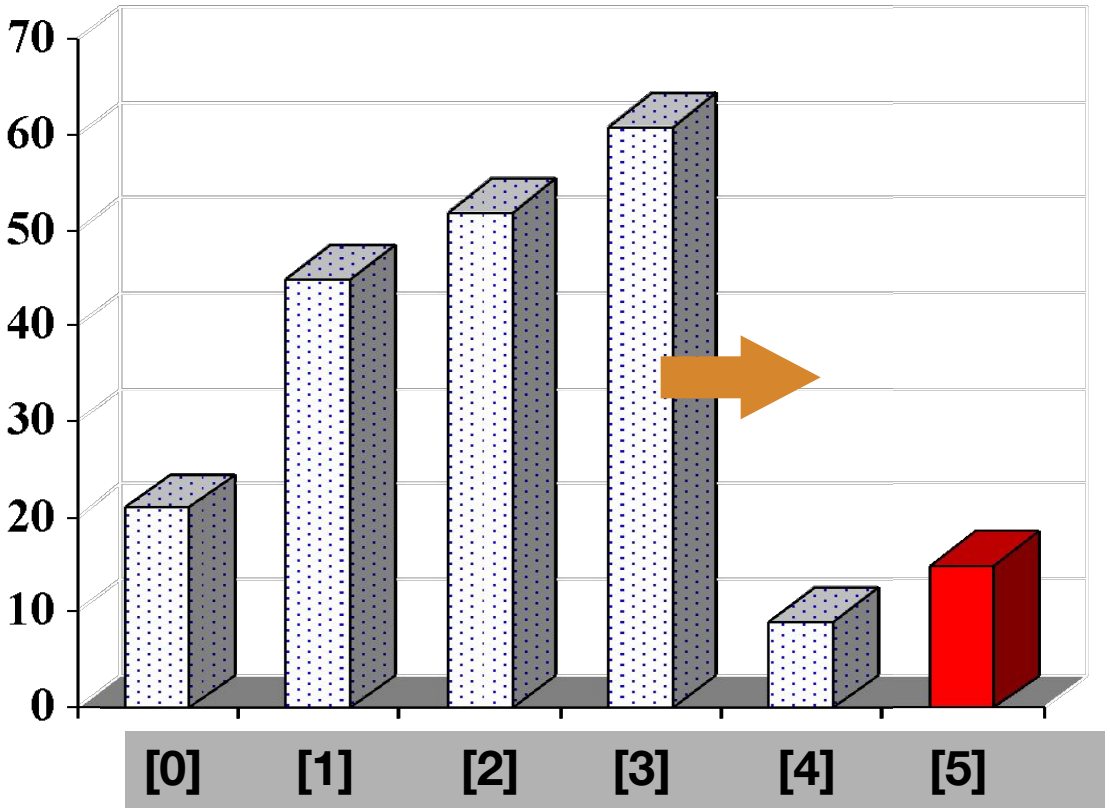
- Copy the new element to a separate location.



How to Insert One Element

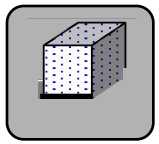
- Shift elements in the sorted side, creating an open space for the new element.

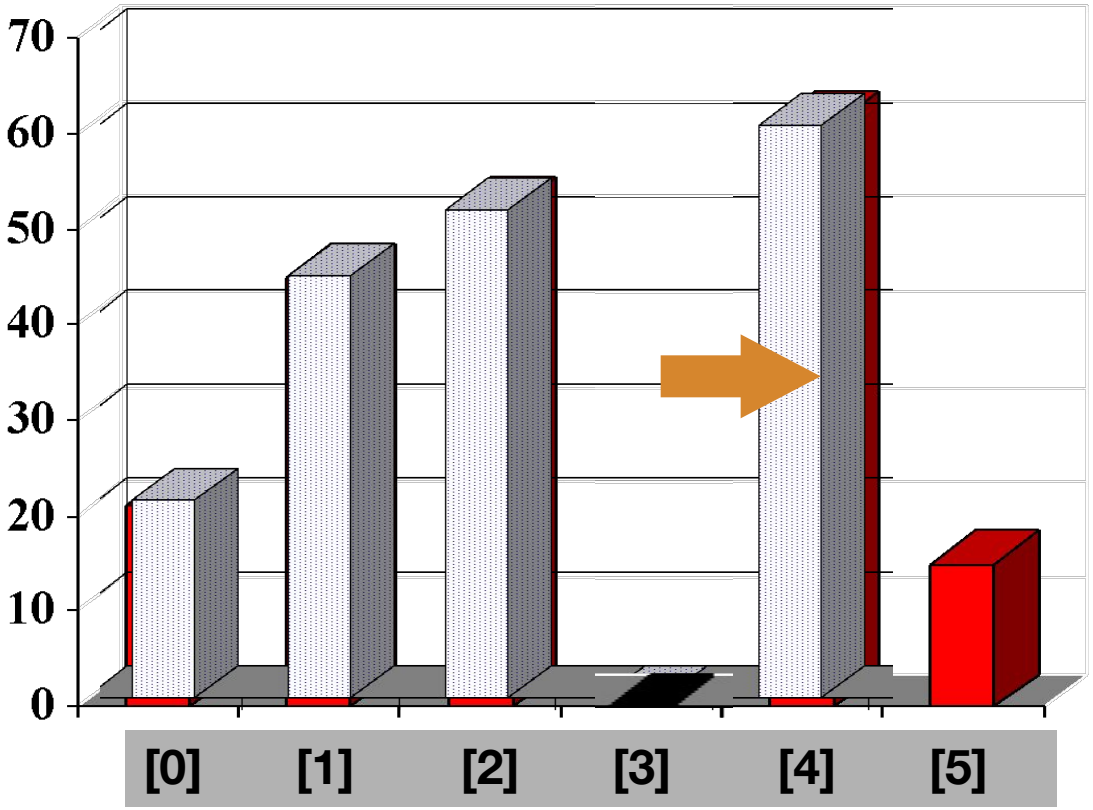




How to Insert One Element

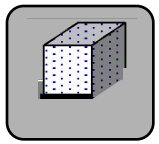
- Shift elements in the sorted side, creating an open space for the new element.

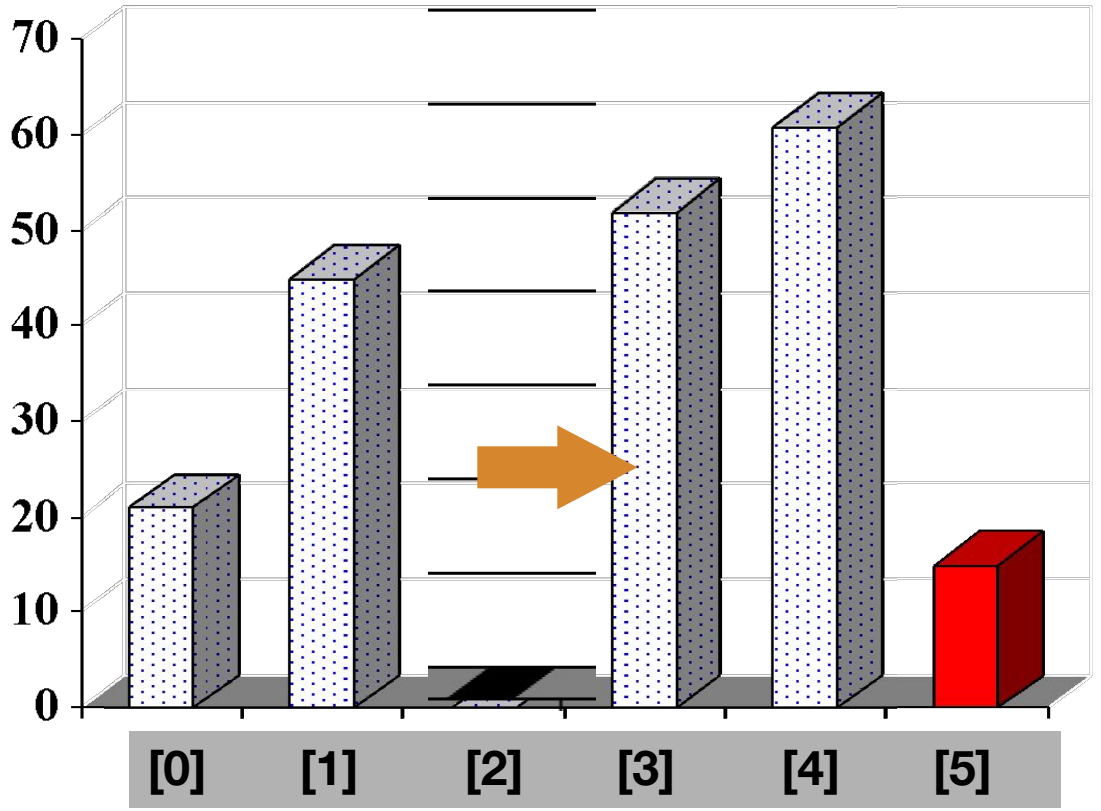




How to Insert One Element

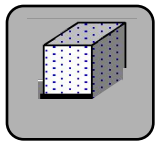
- Continue shifting elements...

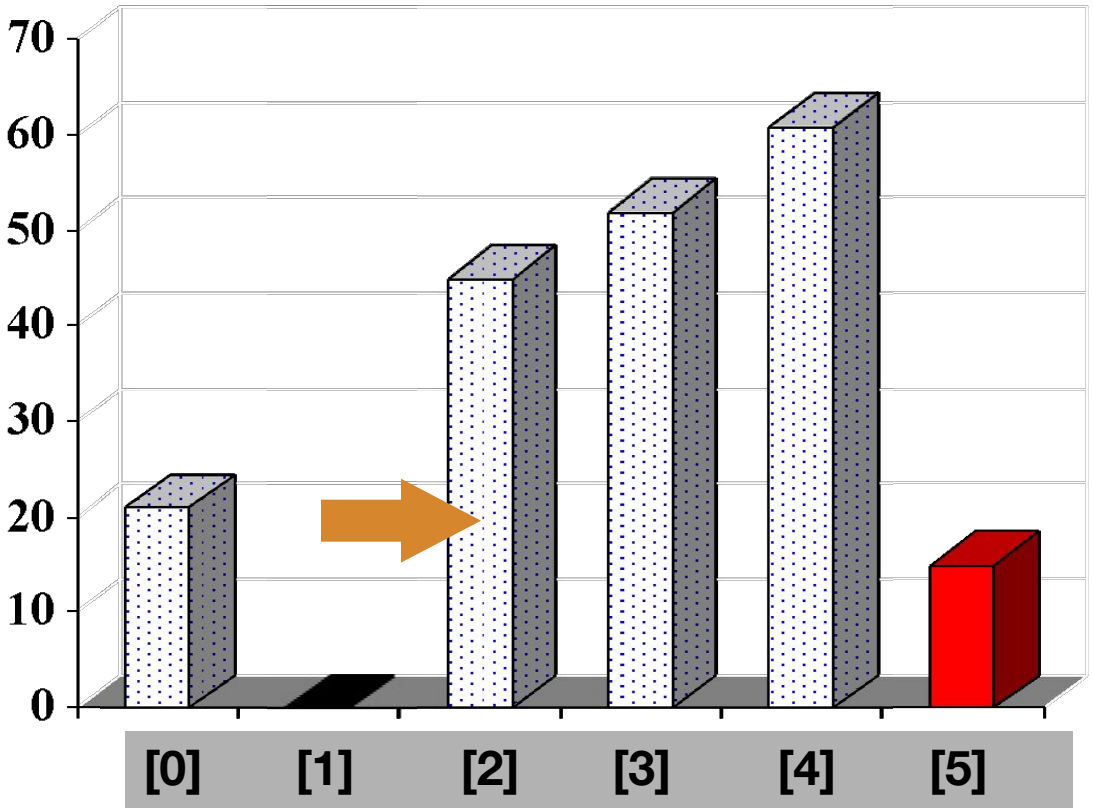




How to Insert One Element

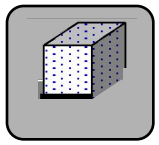
- Continue shifting elements...

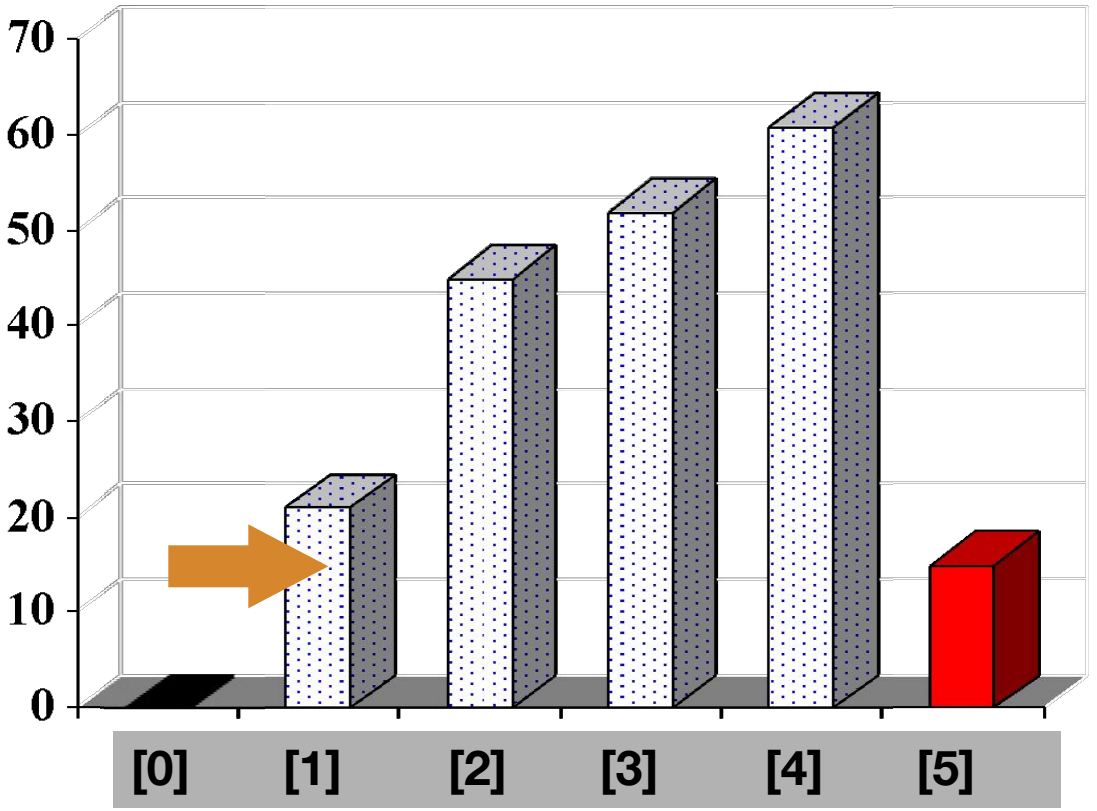




How to Insert One Element

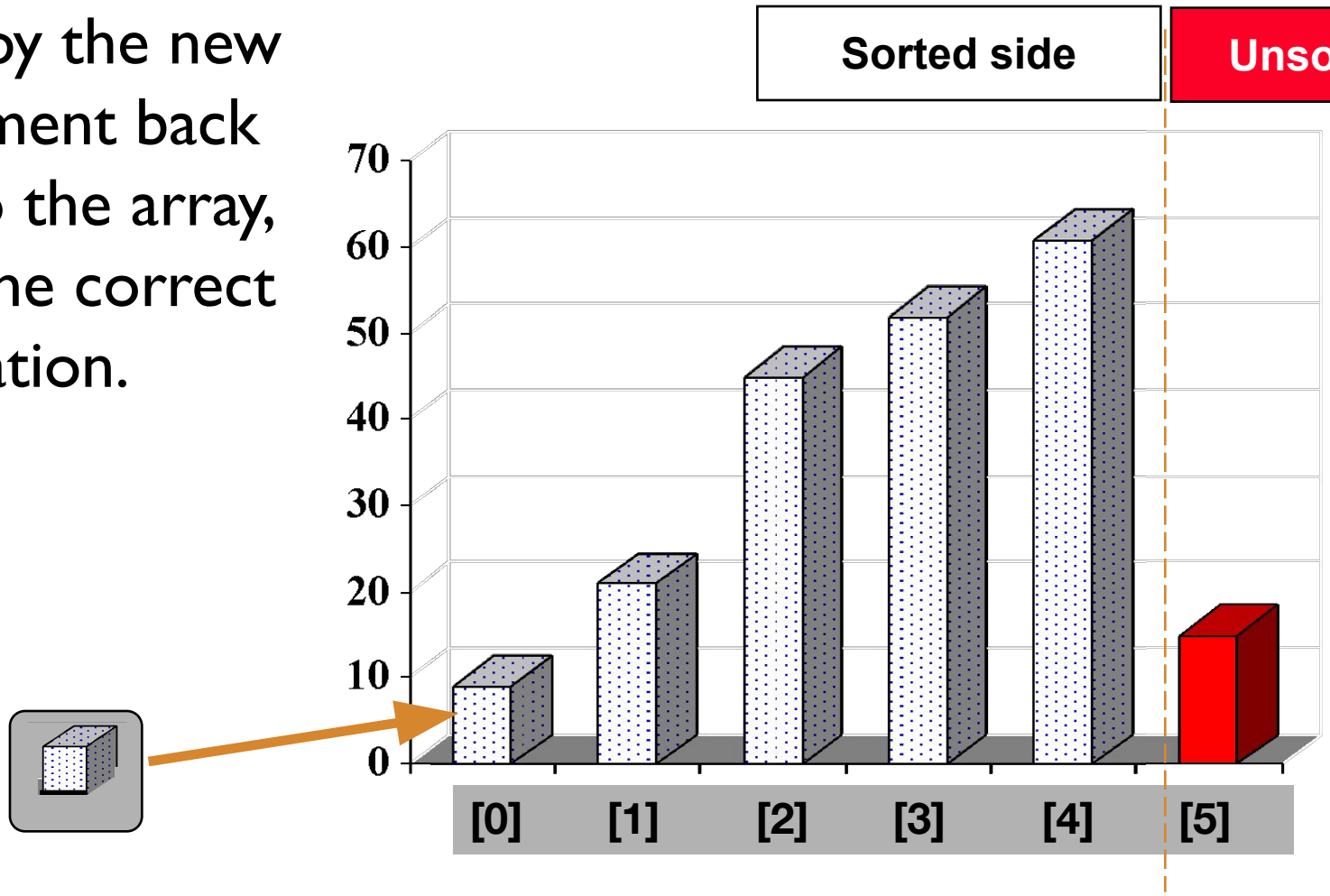
- ...until you reach the location for the new element.





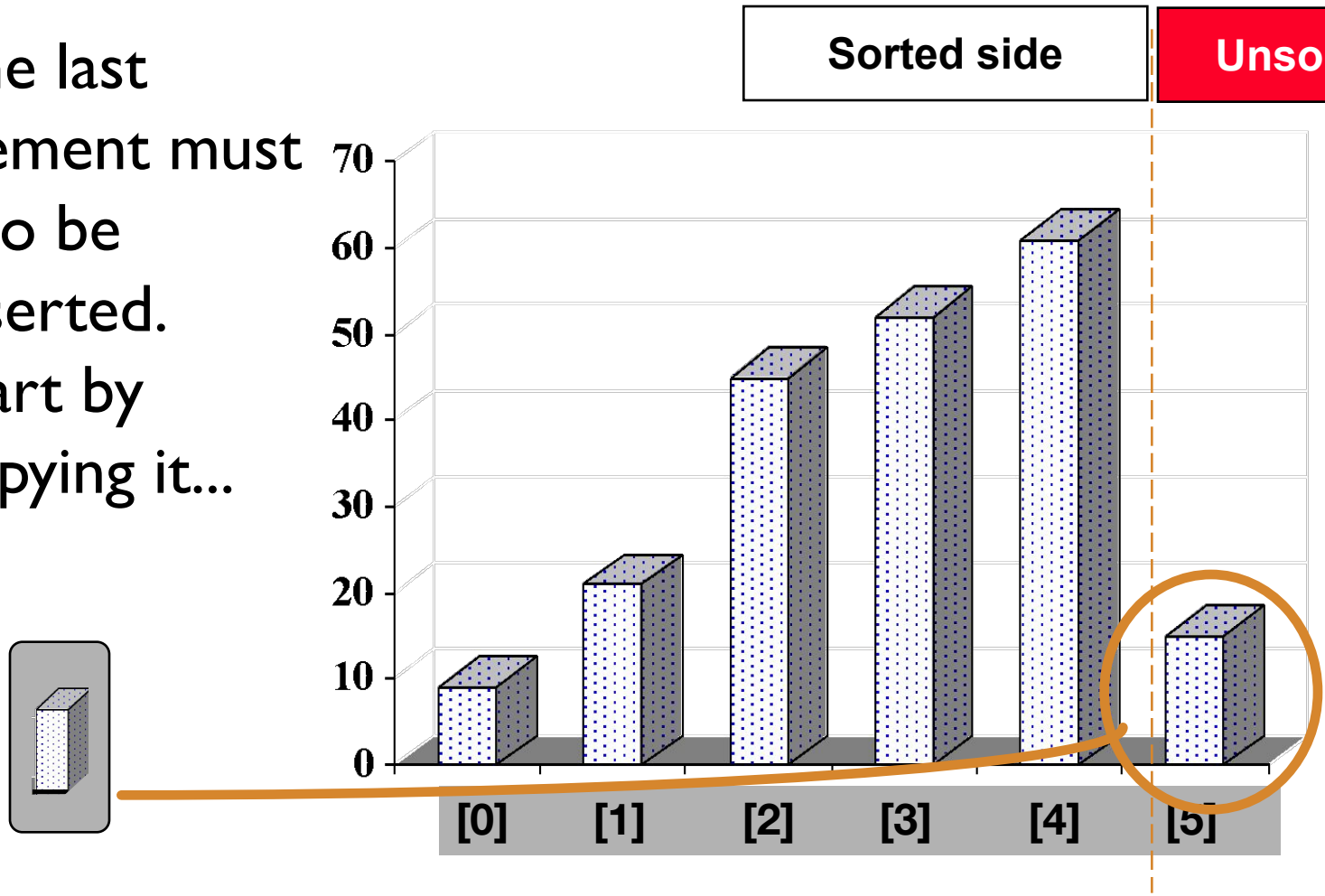
How to Insert One Element

- Copy the new element back into the array, at the correct location.

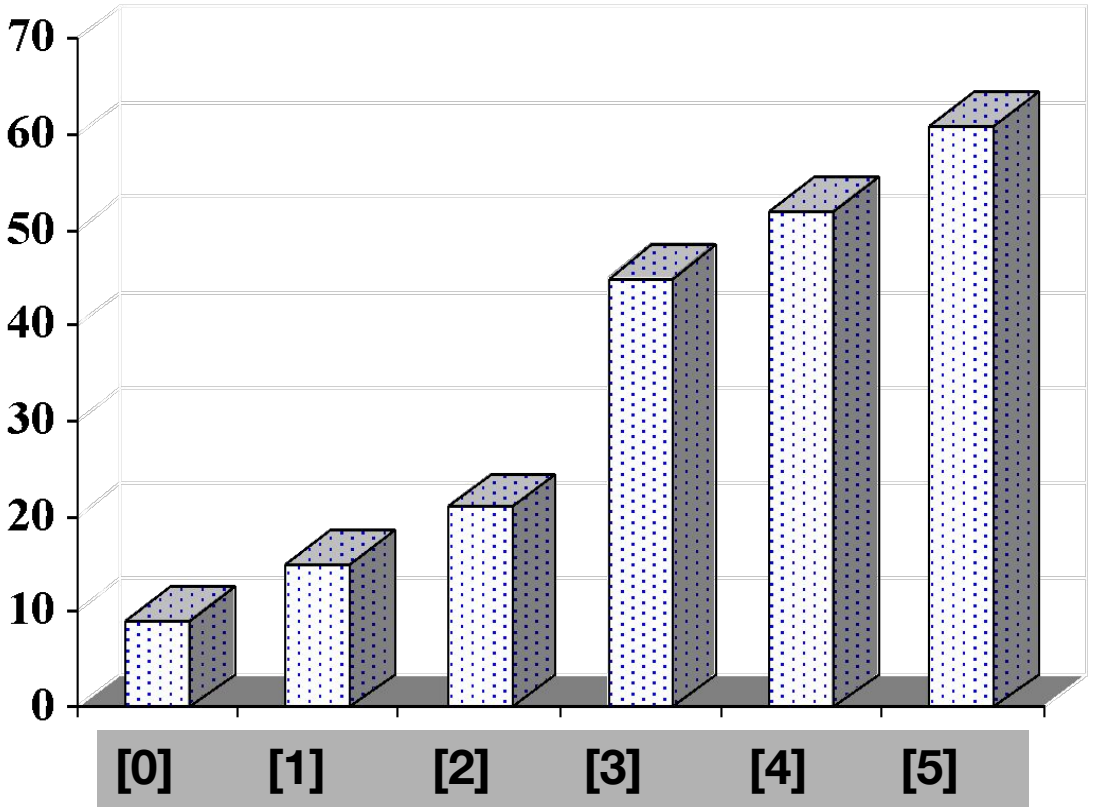


How to Insert One Element

- The last element must also be inserted. Start by copying it...



Sorted Result



Insertion Sort Code

```python
def insertion_sort(arr):
    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        # Move elements of arr[0..i-1], that
        j = i-1
        while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = key

# Example usage
arr = [12, 11, 13, 5, 6]
insertion_sort(arr)
print("Sorted array is:", arr)
```

Merge Sort (Introduction)

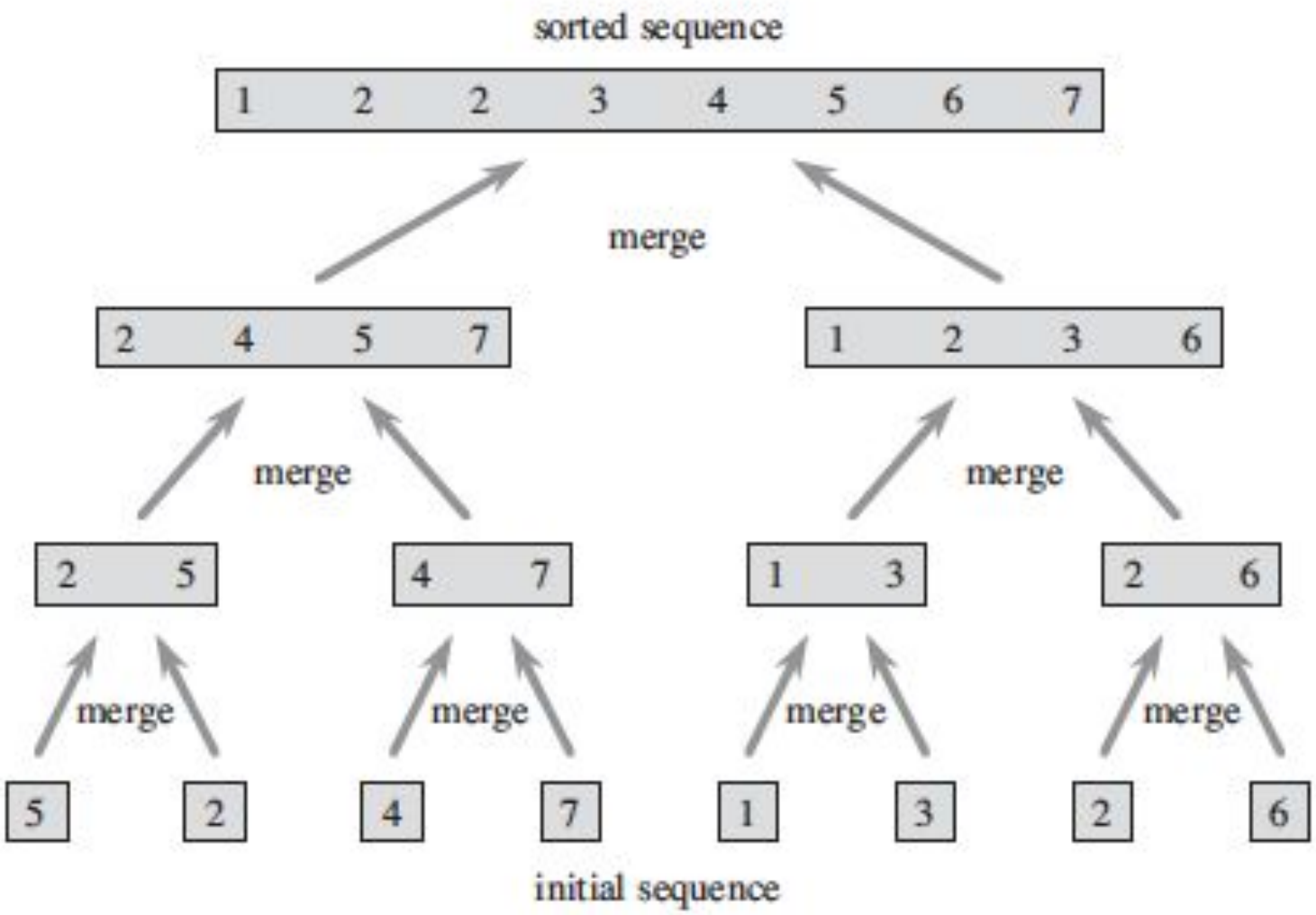
Divide-and-conquer based algorithm with O(nlogn) running time complexity.

Divide: divide the input into number of subparts

Conquer: solve sub-problems recursively

Combine: combine the solution of sub problems into the solution for the original problem

Merge Sort (Concept)



Merge Sort (Algorithm)

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Merge Sort (Merge Call)

```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

Merge Sort (Merge Call Simplified)

Pseudocode for Merge:

```
C = output [length = n]
A = 1st sorted array [n/2]
B = 2nd sorted array [n/2]
i = 1
j = 1
```

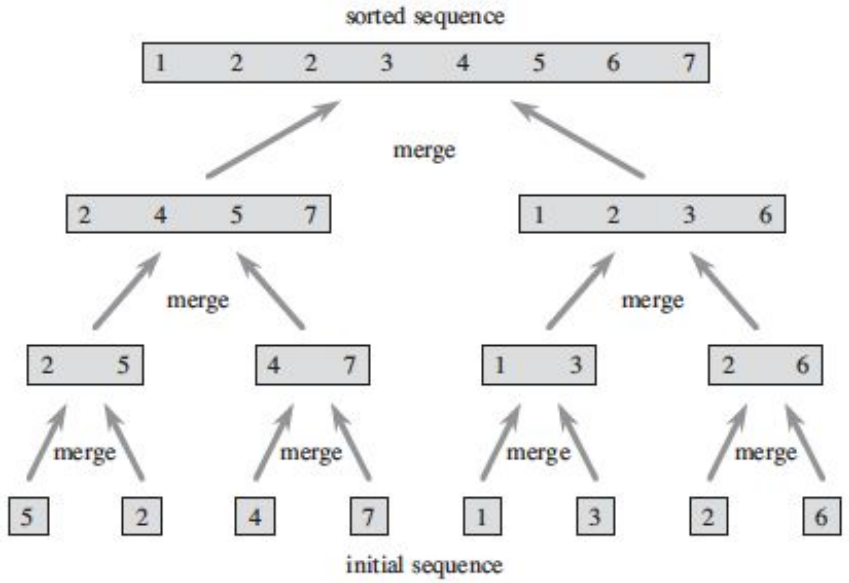
```
for k = 1 to n
        if A(i) < B(j)
                C(k) = A(i)
                i++
        else [B(j) < A(i)]
                C(k) = B(j)
                j++
end
```

(ignores end cases)

Source: Coursera course "*Design and Analysis"* course available at coursera by Tim Roughgarden, Stanford University.

Merge Sort Running Time Complexity

- Lets discuss how Merge Sort running time complexity is O(nlogn)!



1. How many levels of this recursion tree have as a function of n (size of input array)?
2. Running time complexity of merge call?