Tutorial Week 2

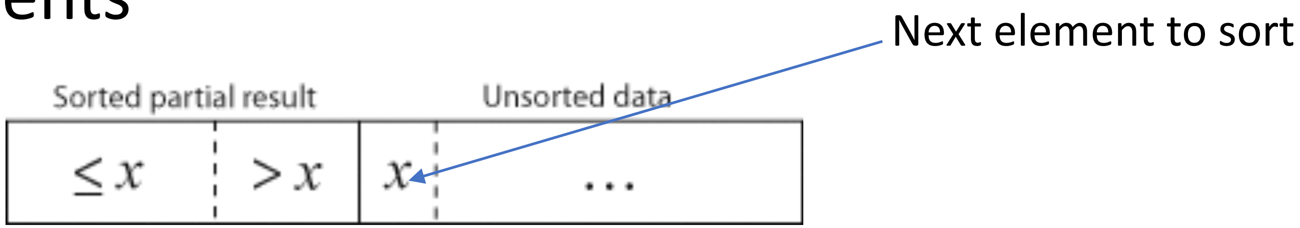
Robin Swanson (robin@cs.toronto.edu)

Tutorial Overview

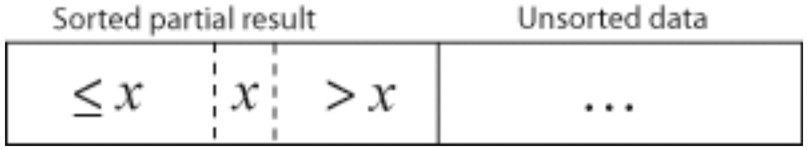
- Insertion Sort
  - Review
  - Example Array Sorting
  - Average Case Running Time
- Group Examples From Textbook
  - 6.1-1, 6.1-5
  - 6.2-4
  - 6.3-2

Insertion Sort Review

- Iteratively sort array from one end to the other
- Given the next element to sort find its new location among the already sorted elements



- Insert into the correct position, shift values > x to the right



- Choose the next element from the unsorted data and start again

Insertion Sort Pseudo Code

```
i ← 1
while i < length(A)
    x ← A[i]
    j ← i - 1
    while j >= 0 and A[j] > x
        A[j+1] ← A[j]
        j ← j - 1
    end while
    A[j+1] ← x
    i ← i + 1
end while
```

Insertion Sort Example

7	***1***	***9***	***10***	***7***	***8***	***8***	***4***	***7***	***2***

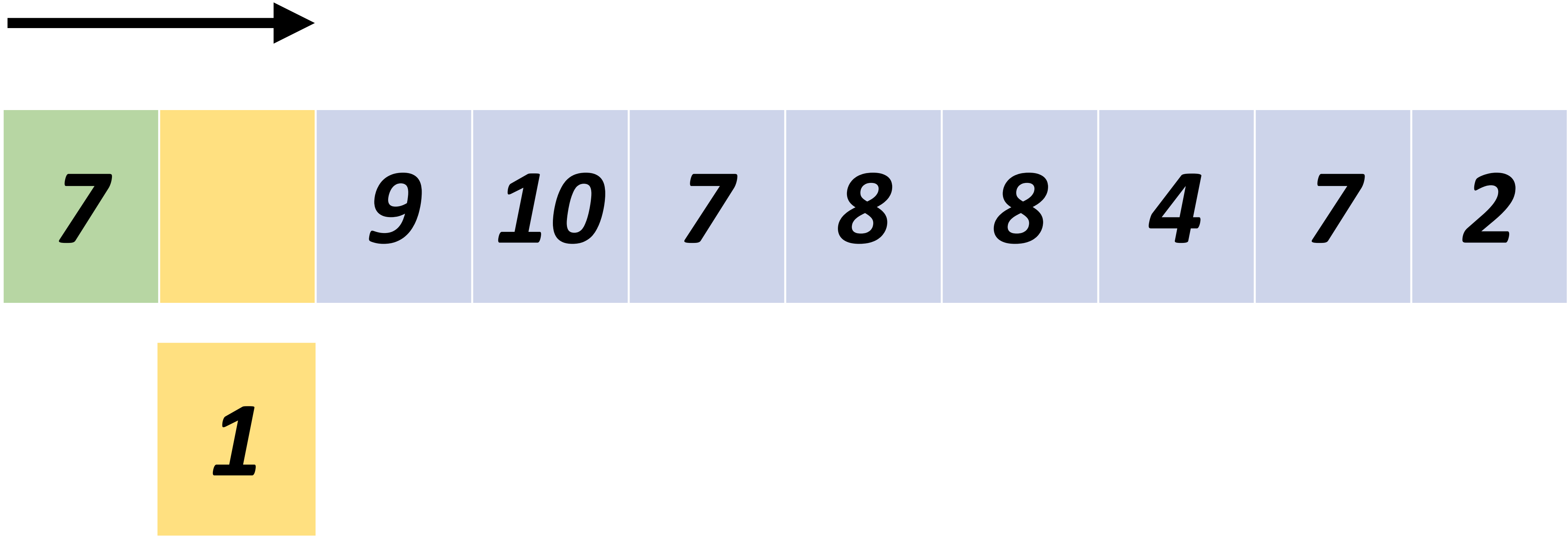
Index to Sort

Insertion Sort Example

7	1	9	10	7	8	8	4	7	2

Index to Sort

Insertion Sort Example



Insertion Sort Example

	7	*9*	*10*	*7*	*8*	*8*	*4*	*7*	*2*

1

Insertion Sort Example

1	7	9	10	7	8	8	4	7	2

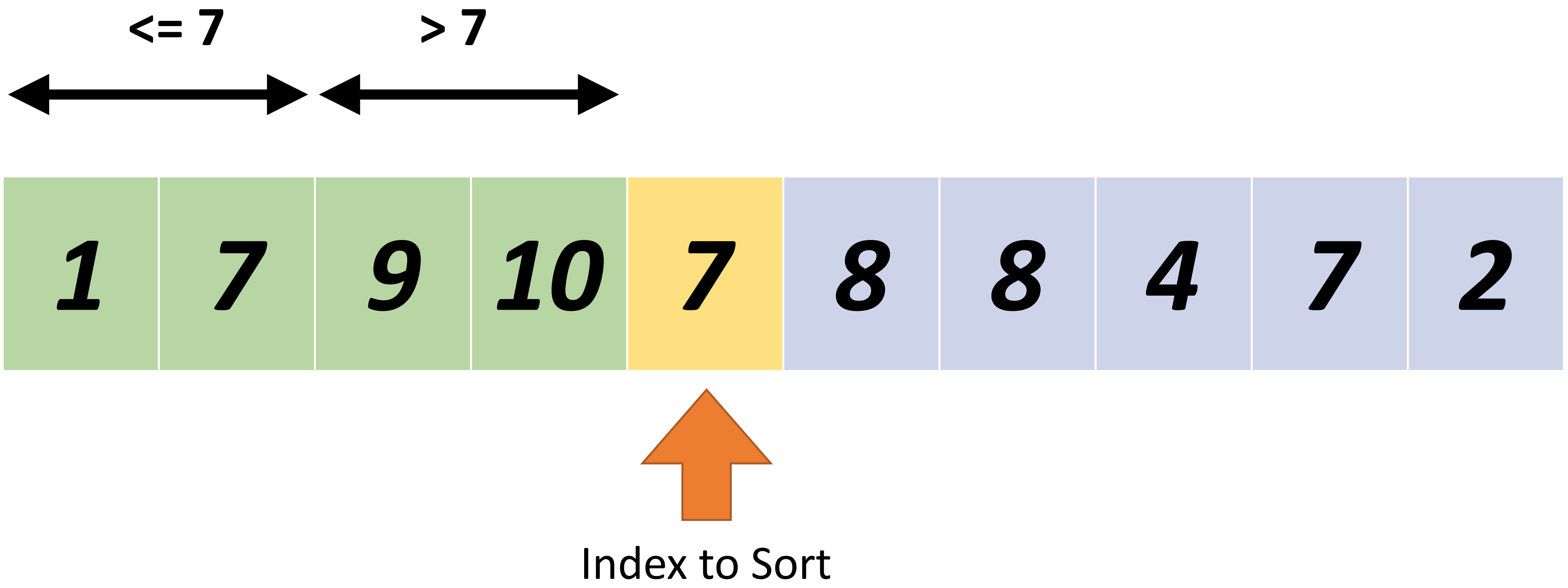
Index to Sort

Insertion Sort Example

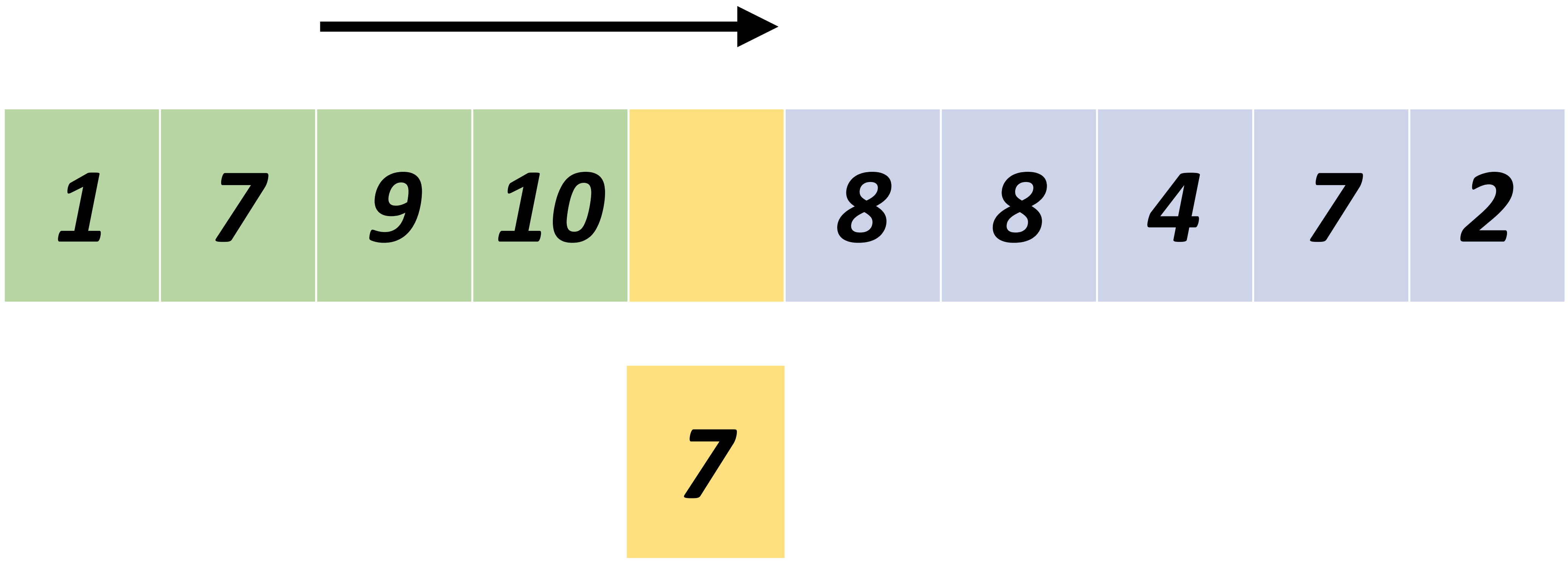
1	7	9	10	7	8	8	4	7	2

Index to Sort

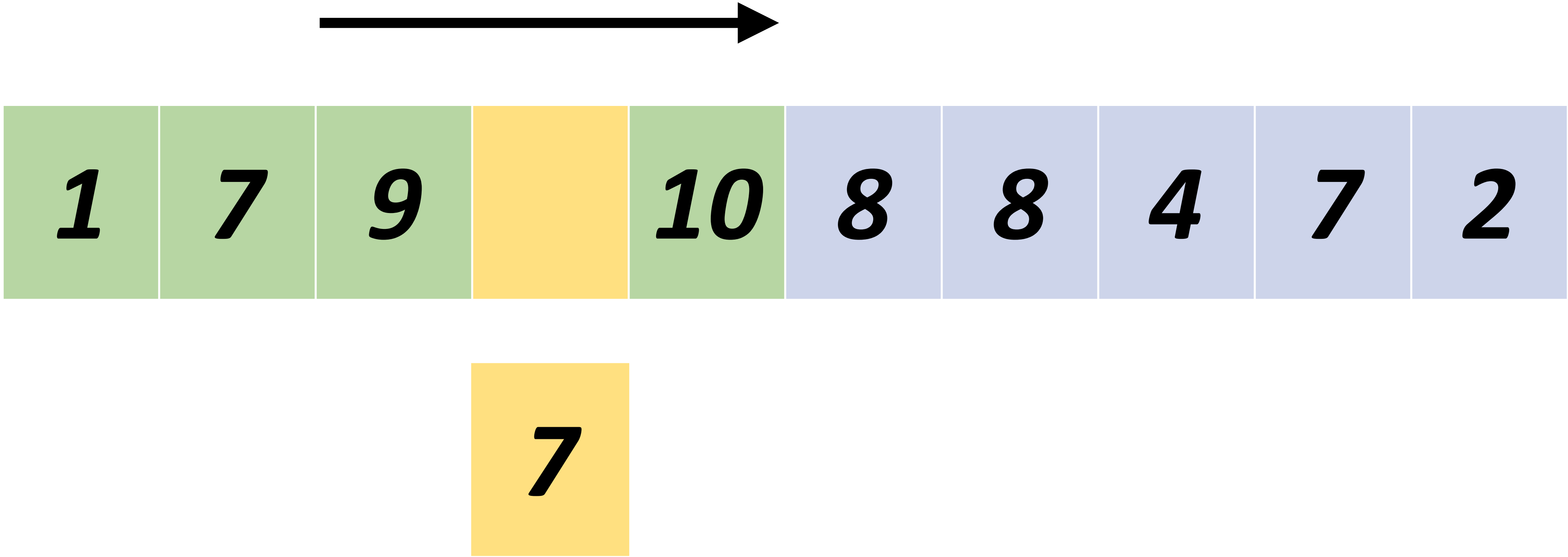
Insertion Sort Example



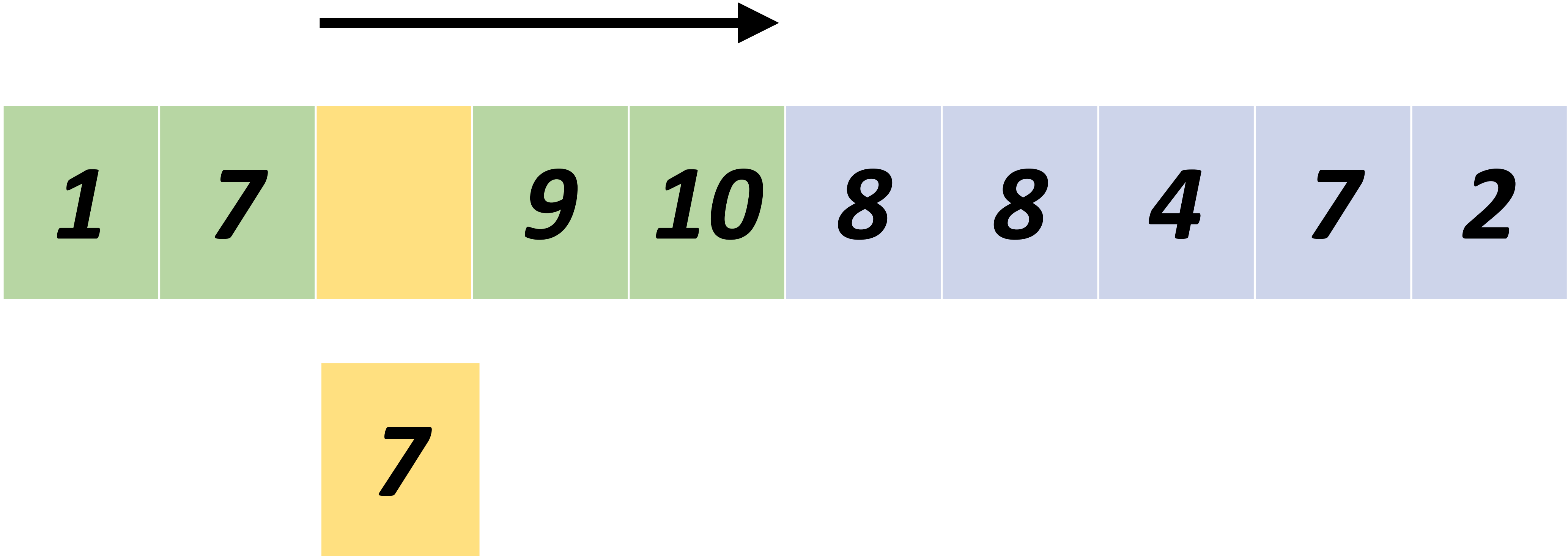
Insertion Sort Example



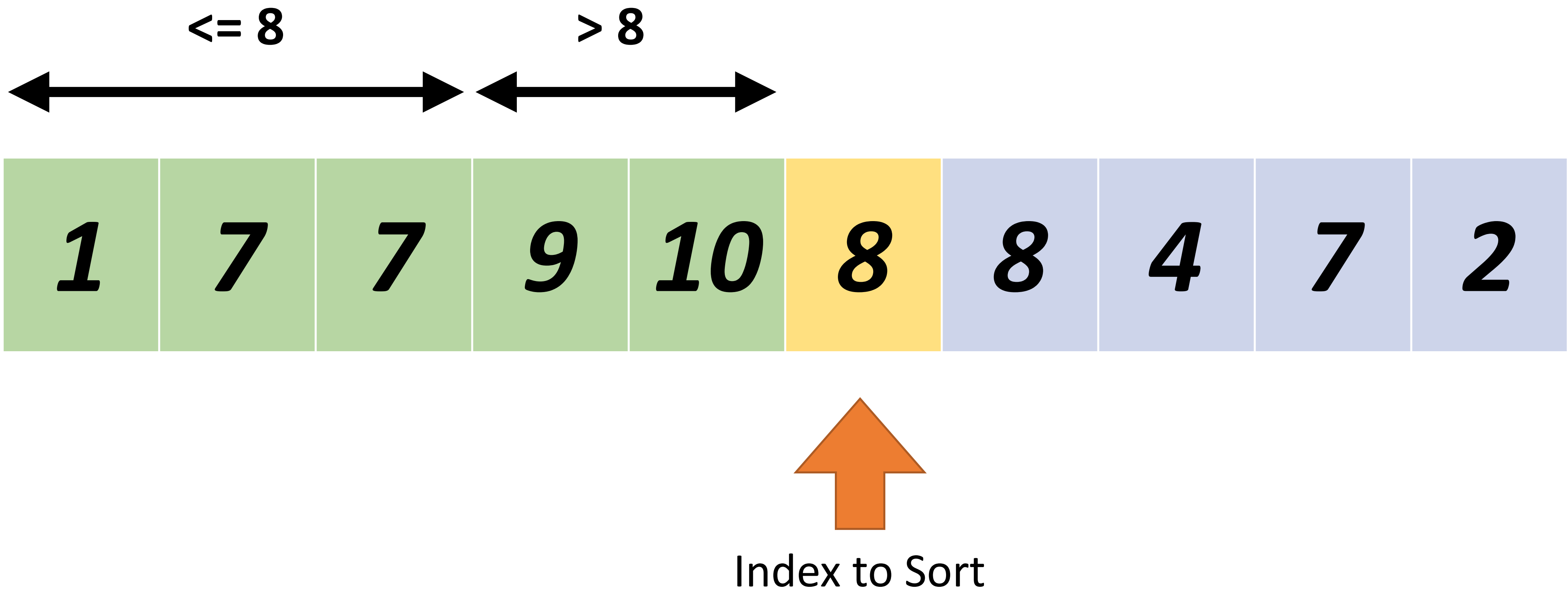
Insertion Sort Example



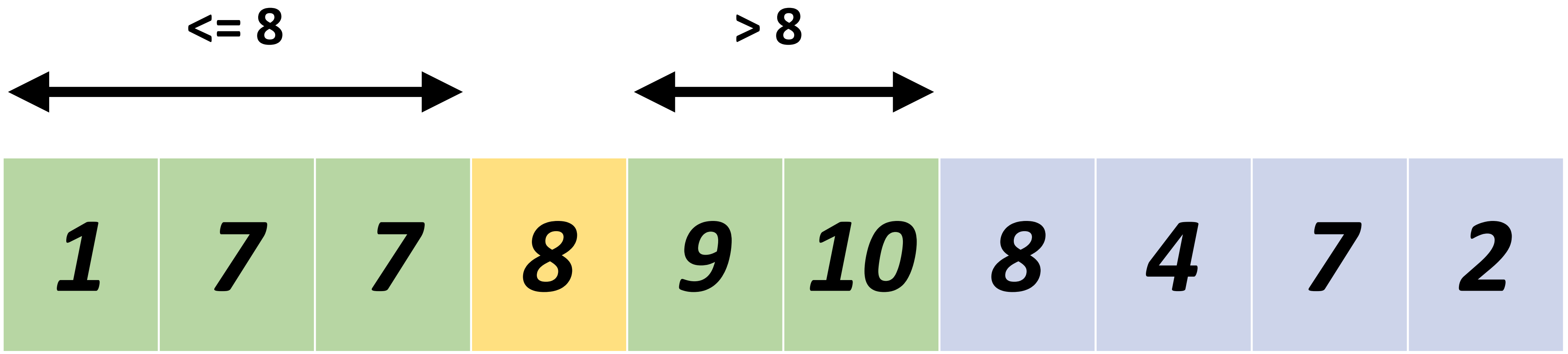
Insertion Sort Example



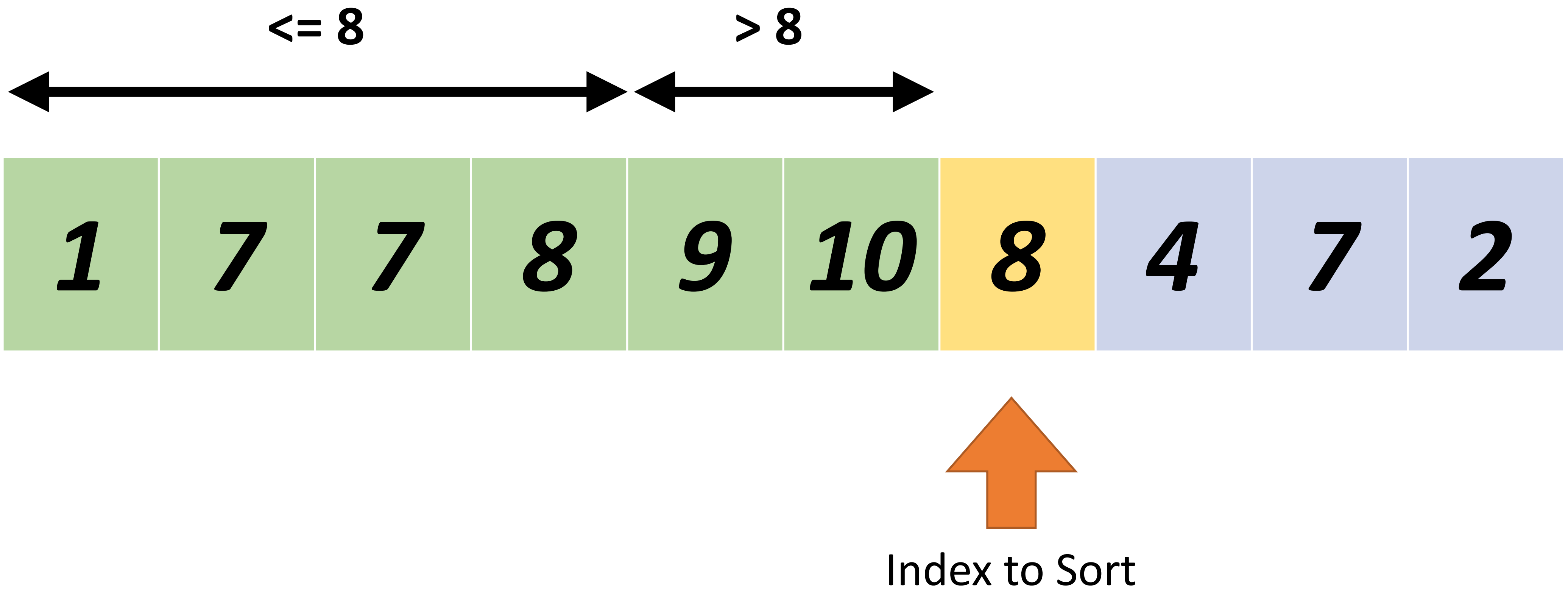
Insertion Sort Example



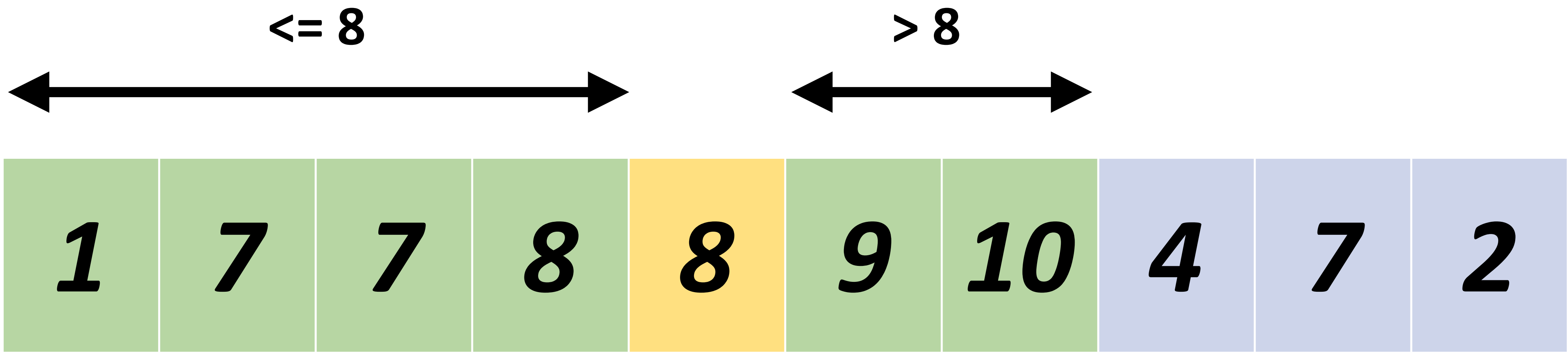
Insertion Sort Example



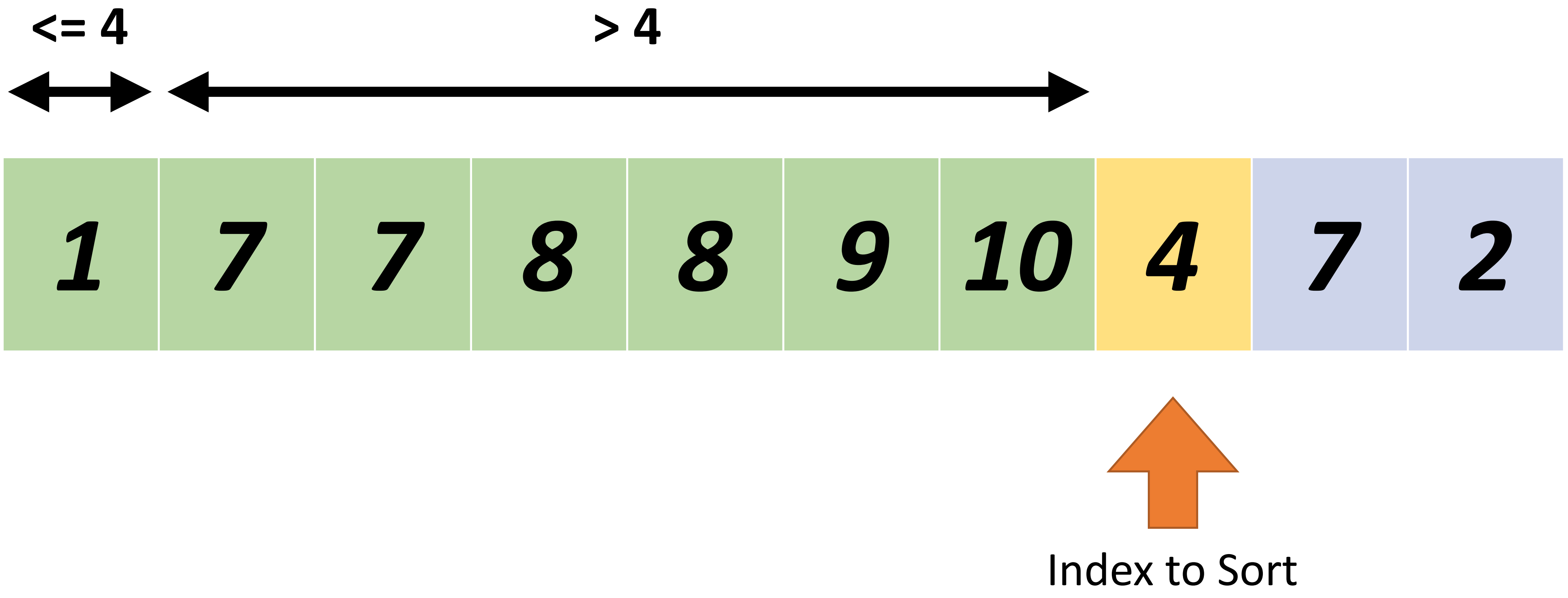
Insertion Sort Example



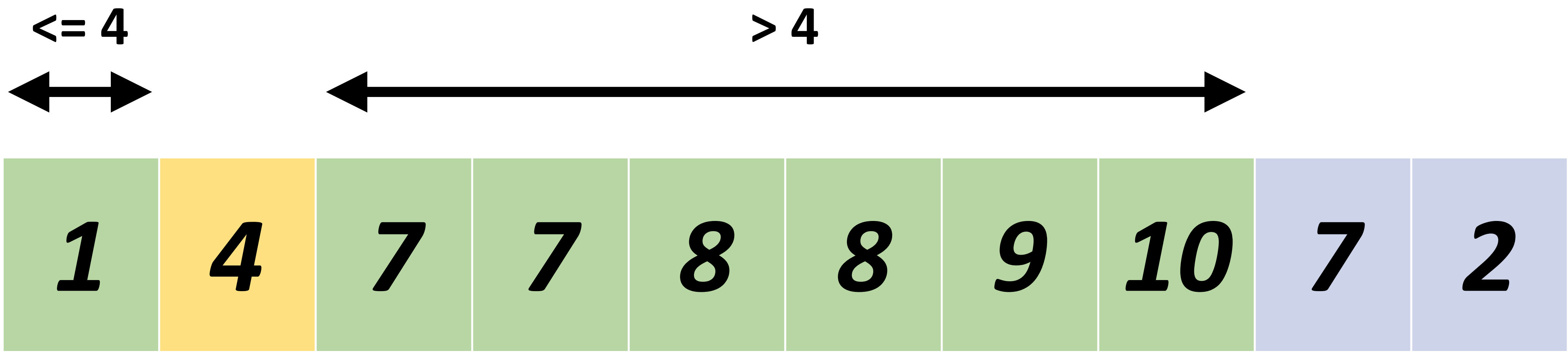
Insertion Sort Example



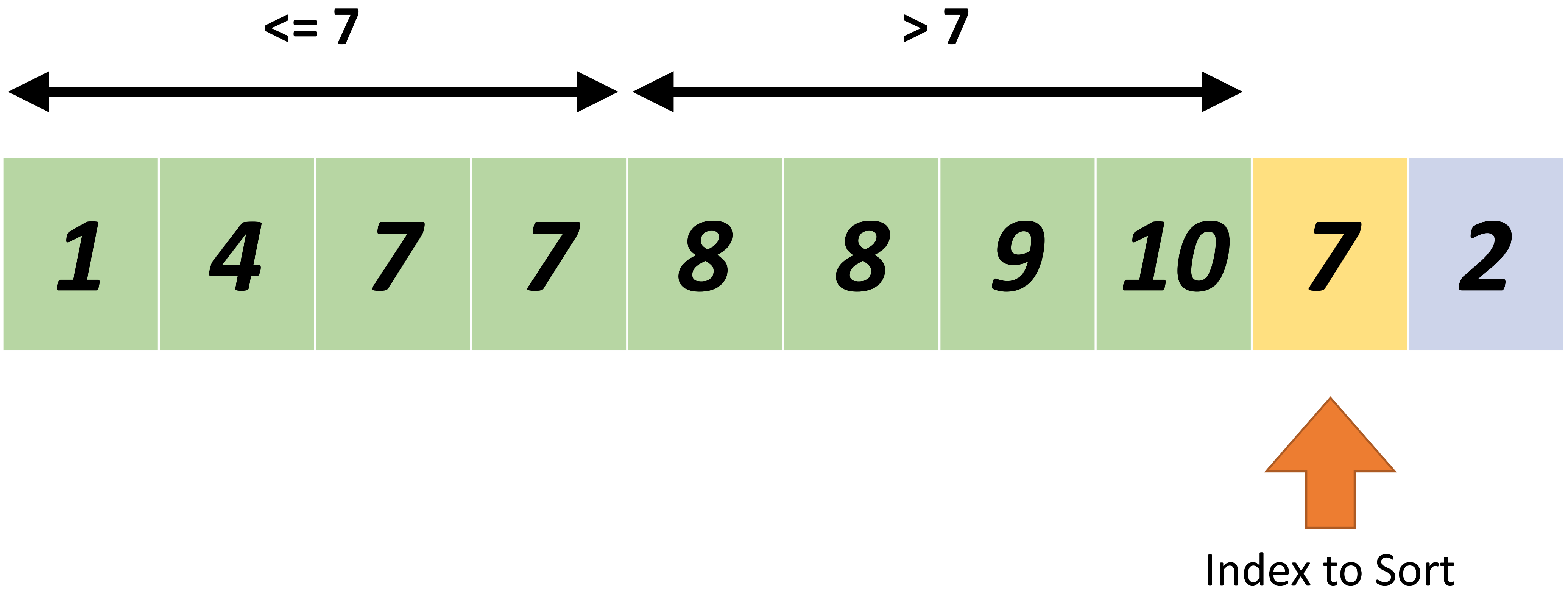
Insertion Sort Example



Insertion Sort Example



Insertion Sort Example

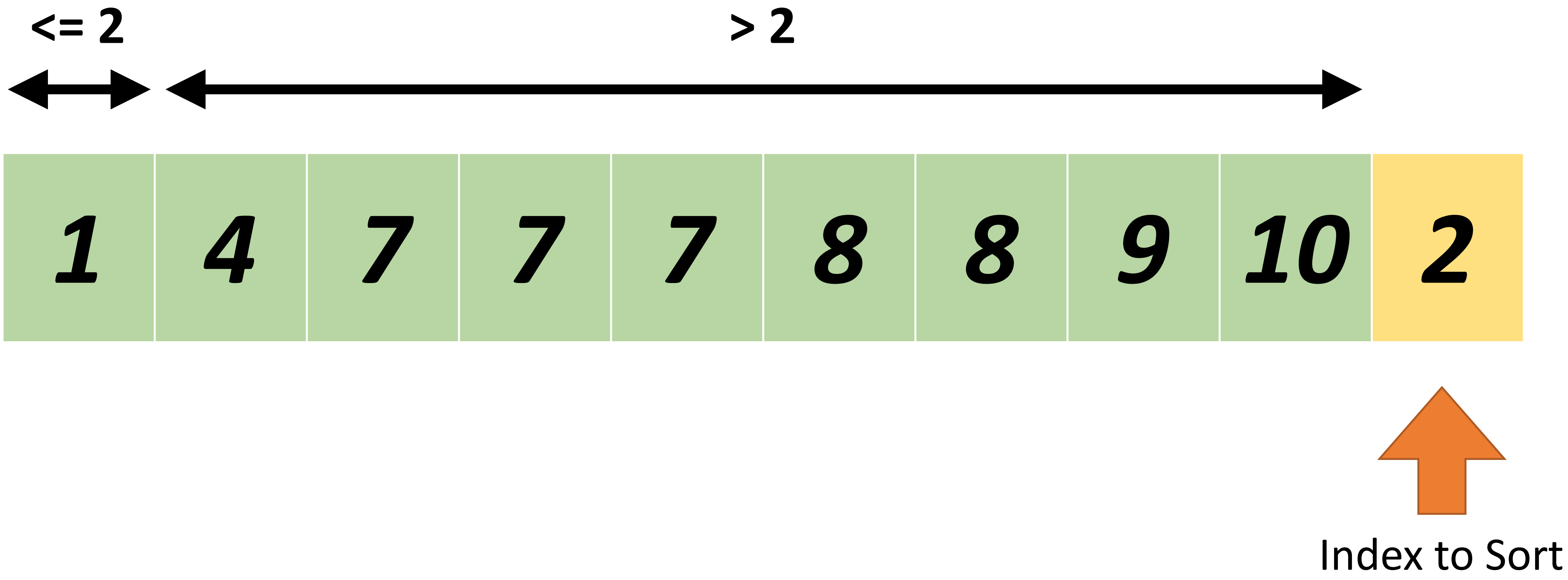


Insertion Sort Example

<= 7 > 7

1	4	7	7	7	8	8	9	10	2

Insertion Sort Example



Insertion Sort Example

<= 2 | > 2

1	2	4	7	7	7	8	8	9	10

Insertion Sort Example

1	***2***	***4***	***7***	***7***	***7***	***8***	***8***	***9***	***10***

Insertion Sort Running Time

- **Best Case?**

- **Worst Case?**

Insertion Sort Running Time

- **Best Case?** O(n)

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***

- **Worst Case?**

Insertion Sort Running Time

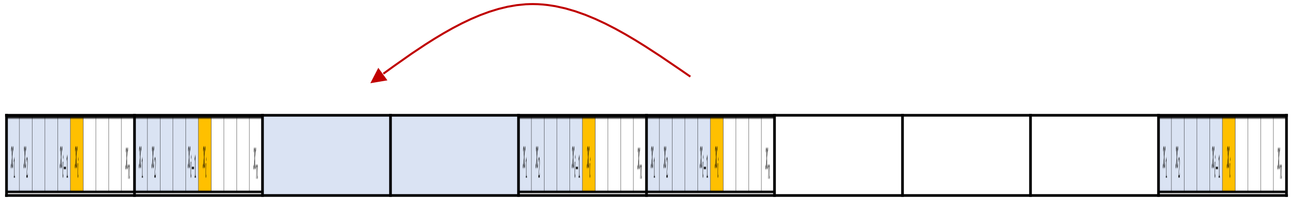
- **Best Case?** O(n)

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

- **Worst Case?** O(n^2)

10	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*

Insertion Sort Average Case



-

For $i = 2$ to n

Insert the element x_i in the partially sorted list $x_1, x_2, \ldots, x_{i-1}$.

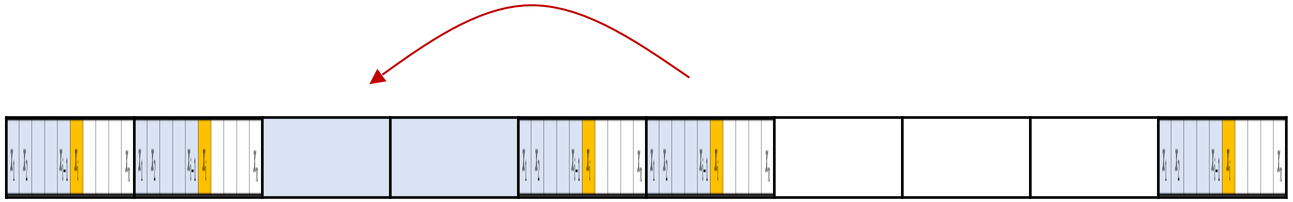
-
  - Let X_i be the random variable which represents the number of comparisons required to insert i^{th} element of the input array in the sorted sub array of first $i-1$ elements.

$$E(X_i) = \sum_{j=1}^{i} x_{ij} p(x_{ij})$$

where $E(X_i)$ is the expected value X_i

and, $p(x_{ij})$ is the probability of inserting x_i in the j^{th} position $1 \leq j \leq i$

How many comparisons it makes to insert ith element in jth position?



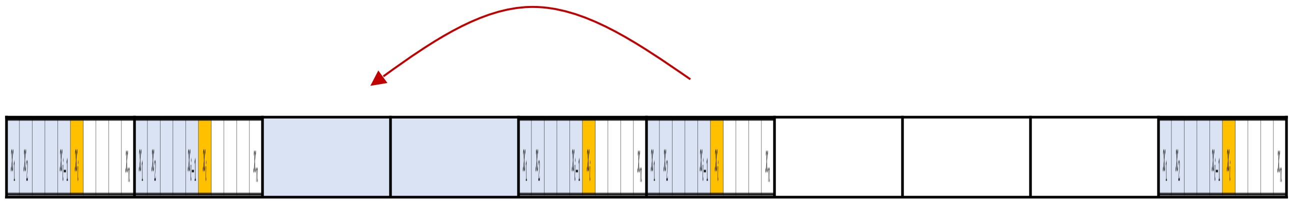
# of Comparisons	Position
1	i
2	$i-1$
3	$i-2$
.	.
.	.
.	.
$i-1$	2
$i-1$	1

Note: Here, both position 2 and 1 have # of Comparisons equal to $i-1$. Why? Because to insert element at position 2 we have to compare with previously first element and after that comparison we know which of them come first and which at second.

•

$$E(X_i) = \sum_{j=1}^{i} x_{ij} p(x_{ij})$$

probability to insert at j^{th} position in the i possible positions: $P(x_{ij} = j) = \frac{1}{i}$



Thus, $E(X_i) = \sum_{j=1}^{i} j * \frac{1}{i} = (i+1)/2$

For n elements, $E(X_2 + \ldots + X_n) = \sum_{i=2}^{n} E(X_i) = \sum_{i=2}^{n} (i+1)/2 = \frac{(n+4)(n-1)}{4} = \frac{n^2}{4} + \frac{3n}{4} - 1$

Therefore average case of insertion sort takes $\Theta(n^2)$

Problems From The Book (6.1-1)

- **Q**: What are the minimum and maximum numbers of elements in a heap of height h?

Problems From The Book (6.1-1)

- **Q**: What are the minimum and maximum numbers of elements in a heap of height *h*?
- **A**: **At least 2^h and at most 2^(h+1)-1**
- A full binary tree of height h-1 has at most 2^h-1 elements

Problems From The Book (6.1-4)

- **Q**: Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

Problems From The Book (6.1-4)

- **Q**: Where in a max-heap might the smallest element reside, assuming that all elements are distinct?
- **A:** The smallest element must be a leaf node. Recall that by the max-heap property a child must be less than or equal to its parent. Because every element is distinct in this heap this becomes a strict inequality. Therefore the smallest element must be somewhere in a leaf node.

Problems From The Book (6.2-4)

- **Q:** What is the effect of calling Max_Heapify(A, i) if i > A.heap_size/2?

Problems From The Book (6.2-4)

- **Q:** What is the effect of calling Max_Heapify(A, i) if i > A.heap_size/2?
- **A: Nothing**. i must be a leaf node, therefore the recursive call will never be made and the heap will not be changed.

Problems From The Book (6.3-2)

- **Q:** Why do we want the loop index i in line 2 of Build_Max_Heap to decrease from floor(A.length/2) to 1 rather than increase from 1 to floor(A.length/2)?

```
Build_Max_Heap(A):
    A.heap_size = A.length
    for i = floor(A.length/2) -> 1
        Max_Heapify(A,i)
```

Problems From The Book (6.3-2)

- **Q:** Why do we want the loop index i in line 2 of Build_Max_Heap to decrease from floor(A.length/2) to 1 rather than increase from 1 to floor(A.length/2)?
- **A:** If we begin at element 1, higher value child nodes may not be swapped up to the top of the heap.
  - Example: [2,1,1,3]
    1. 2 larger than 1 & 1, no swap
    2. 1 < 3 so swap 1 & 3
    3. Finished. Result: [2,3,1,1]