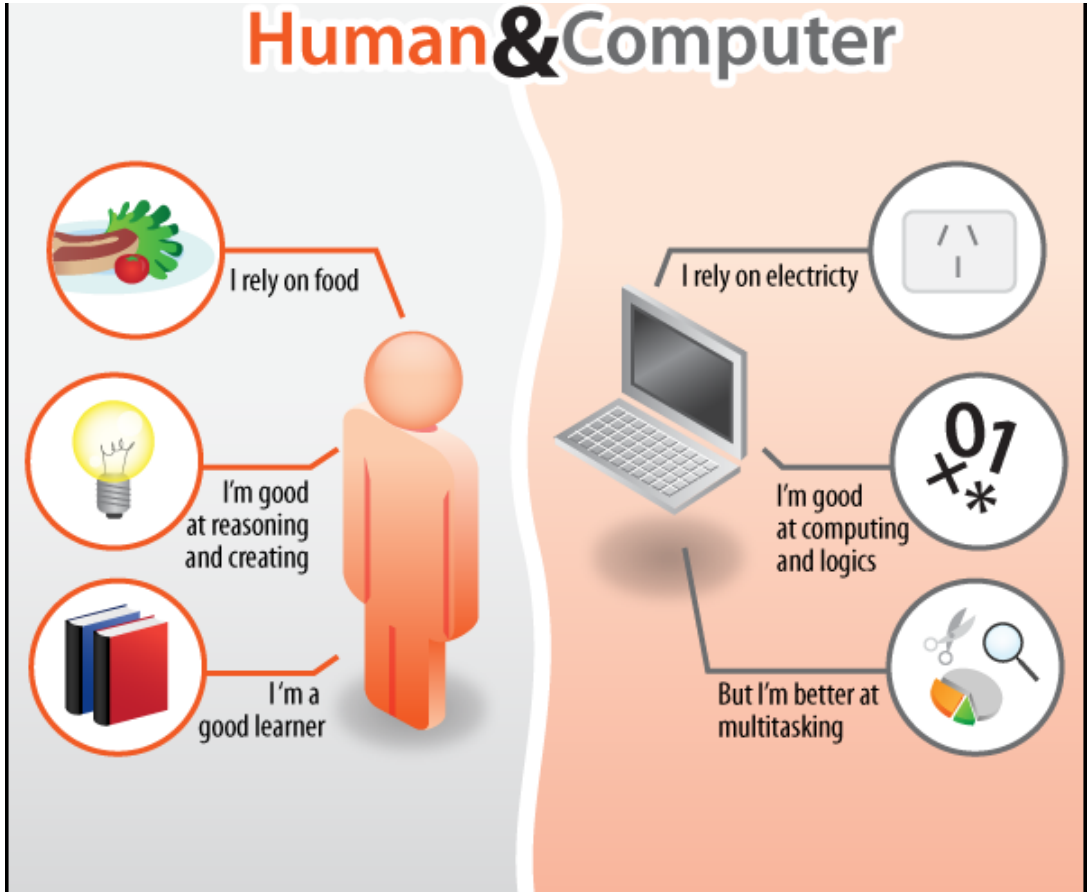
CIS 606 Analysis of Algorithms

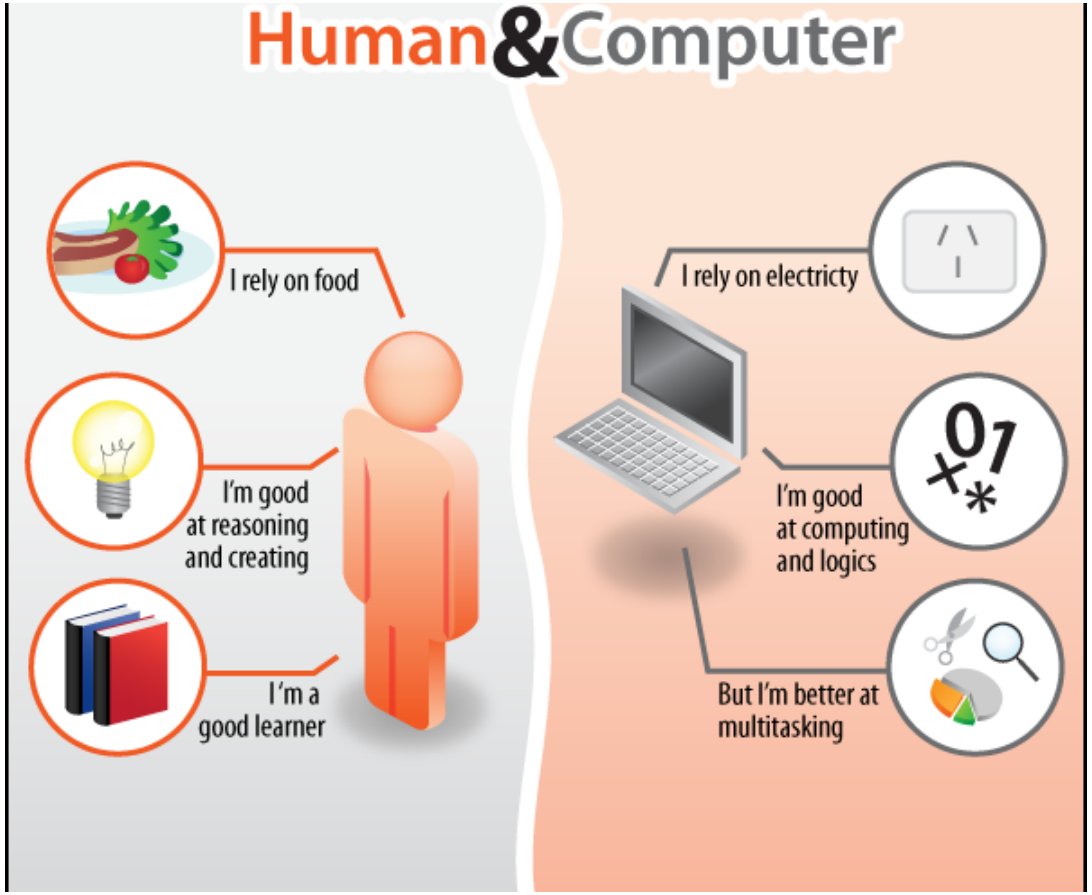
Introduction

Human vs Computer



reference: http://www.differencebetween.net/human-vs-computer/

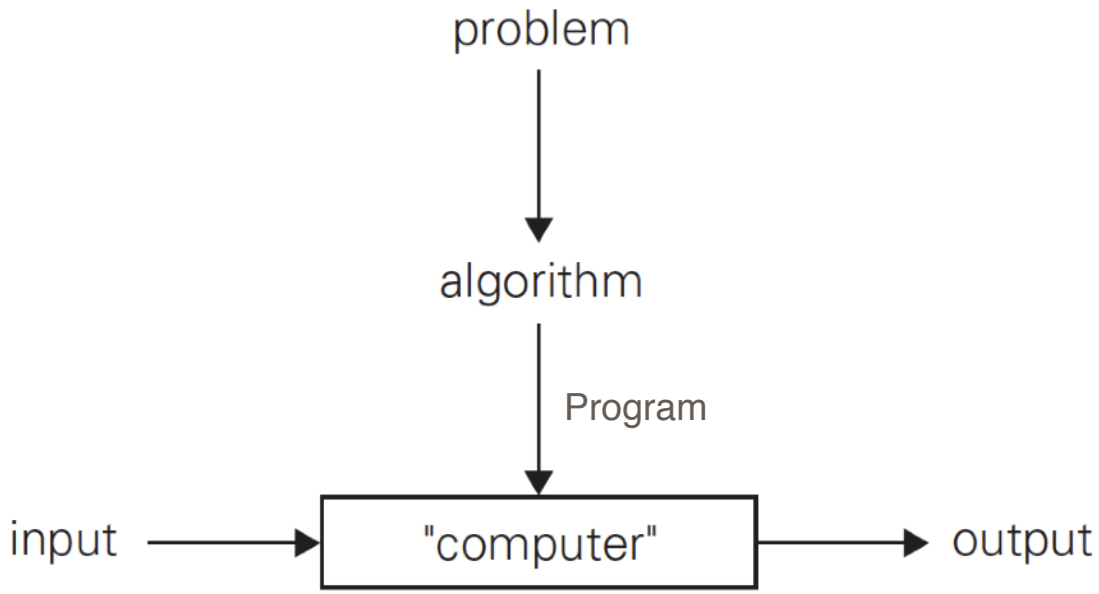
Human vs Computer



reference: http://www.differencebetween.net/human-vs-computer/

What Is An Algorithm?

- **An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.**



Computing Environment

- The random-access machine (RAM)
  - A single CPU with a fixed number of registers each with w bits.
    - No concurrent operations
    - Algorithm executed on RAM is called Sequential Algorithm.
  - An infinite sequence of cells for memory: each cell contains w bits.
    - Memory access takes one time step, and there is no shortage of memory.
  - Each atomic operation takes 1 time step.

Atomic Operations on RAM

- Set a register a to a value: a=10.
- An arithmetic operation
  - a+b, a-b, a*b, a/b(integer division: 6/3=2, 7/3=2)
- A comparison/branching operation
  - a<b, a=b, a>b
- A memory access
  - Read the memory cell at address A into a register
  - Write the value of a register into the memory cell at address A

Algorithm for Computing the Sum

```
Sum(Array A : list of integers)
{
     n = A.size
     sumA = 0
     for i = 1 to n
          sumA = sumA + A[i]
     return sumA
}
```

The cost of an algorithm:
Time Complexity and Space Complexity

An algorithm is the step-by-step procedure (i.e., a sequence of atomic operations) to compute the solution.

Time and Space

- **Time complexity**: How long does an algorithm take to compute the solution?
  - For any given problem the running time of an algorithms is the number of time steps, i.e., the number of atomic operations.
- **Space complexity**: How much memory an algorithm use in its execution?
  - The space used by an algorithm is assumed to be the number of RAM memory cells.

What is a good Algorithm?

- Two standards:
  - Correctness
  - Use less resources: time and space

Methods of Specifying an Algorithm

- Natural Language
- Pseudocode
- Flowchart
- Computer Program

Computing Time Complexity of an Algorithm

- Run the code on computer with different input and time it.

```
Sum(Array A : list of integers)
{
    n = A.size
    sumA = 0
    for i = 1 to n
        sumA = sumA + A[i]
    return sumA
}
```

Computing Time Complexity of an Algorithm(cont)

- Analyze the time on the paper

```
Sum(Array A : list of integers)
{
    n = A.size
    sumA = 0
    for i = 1 to n
        sumA = sumA + A[i]
    return sumA
}
```

What is the running time?

Time Complexity T(n)

- The time complexity of an algorithm is a function with respect to the input size n.
- Constant time: O(1)
  - T(n) = 4, T(n)=10000000
- Linear: O(n)
  - T(n) = 300n+1, T(n) = 1024n+100000000
- Quadratic: O(n^2)
  - T(n) = $2n^2$+1, T(n) = $1024n^2$+20
- Exponential:O(2^n)
  - T(n) = 2^n

How Do We Measure the Running Time?

- Depends on the input size n
  - n=6, n= 6000, n=$6*10^6$
- Depends on the input pattern
  - Worst case
  - Best case
  - Average case
- Depends on the machine
  - Super computer, phone

Example: Sorting Problem - Bubble Sort

- Input: a sequence of numbers {71, 32, 65, 4, 45, 67, 20} in an array A
- Output: the ascending order of A = {4, 20, 32, 45, 65, 67, 71}
- Algorithm: Bubble Sort
  - Repeat to step through the list
  - Every pass, compare adjacent elements and swap them if they are out of order
  - End until no swaps are needed

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

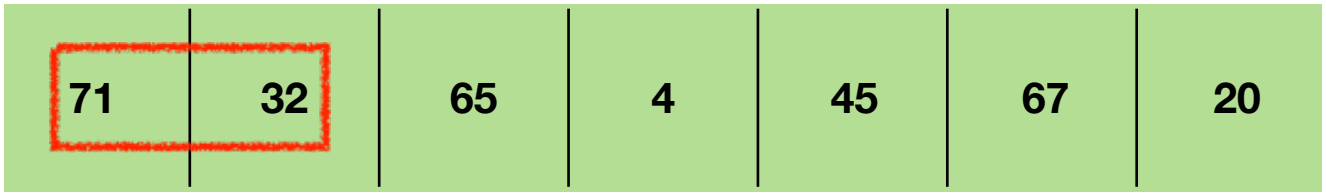
First Pass:

71	32	65	4	45	67	20

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

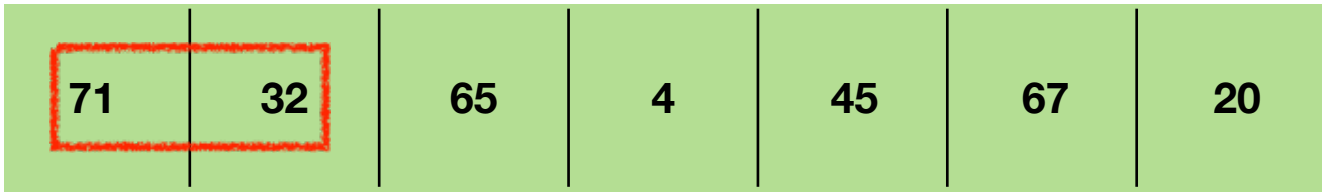
First Pass:



Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

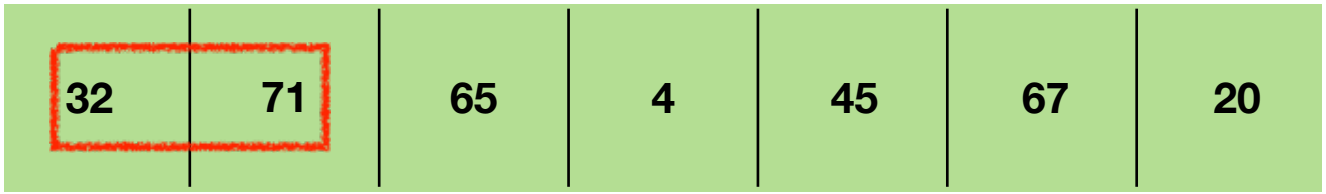


71>32

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

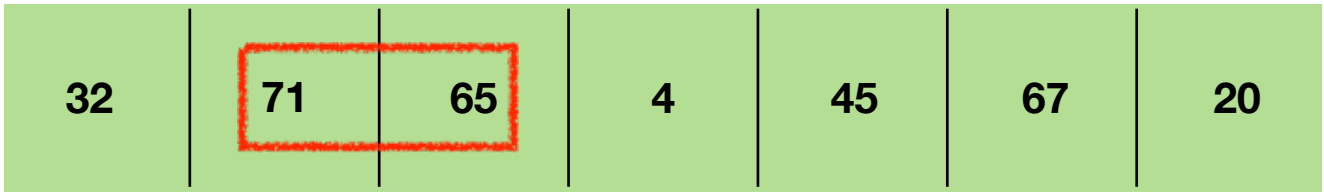


71>32

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

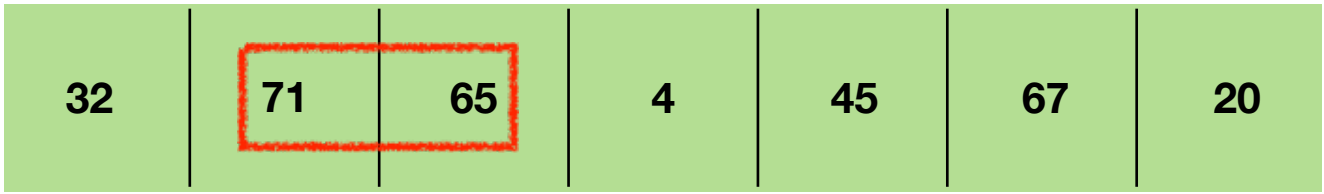
First Pass:



Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

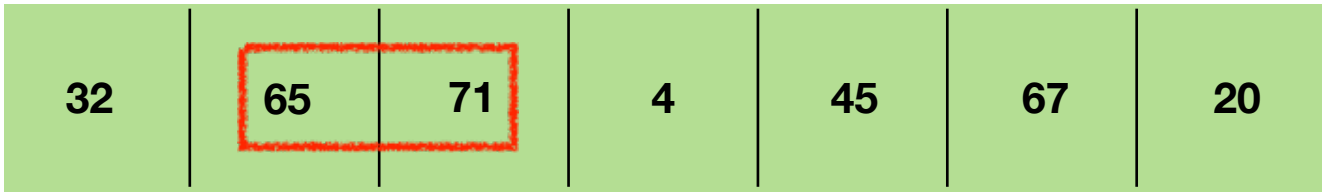


71>65

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

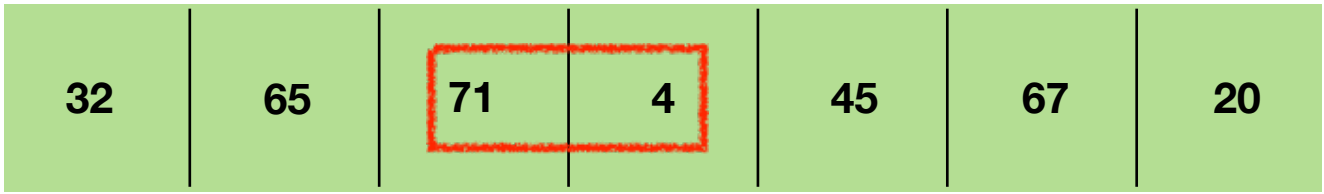


71>65

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

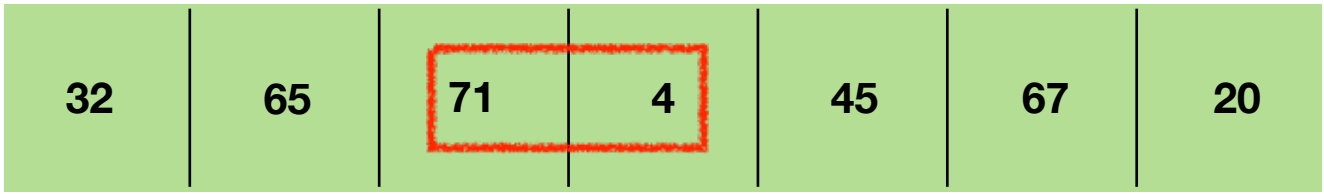
First Pass:



Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

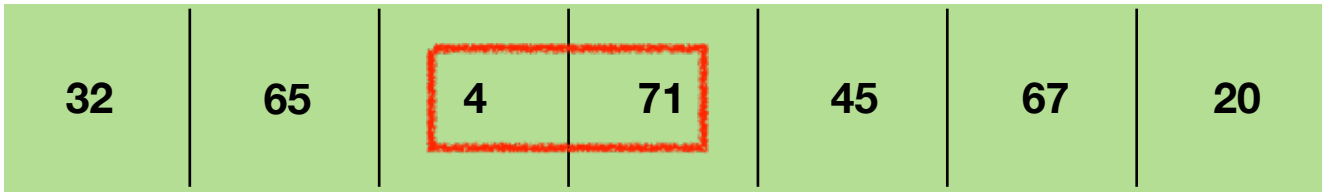


71>4

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

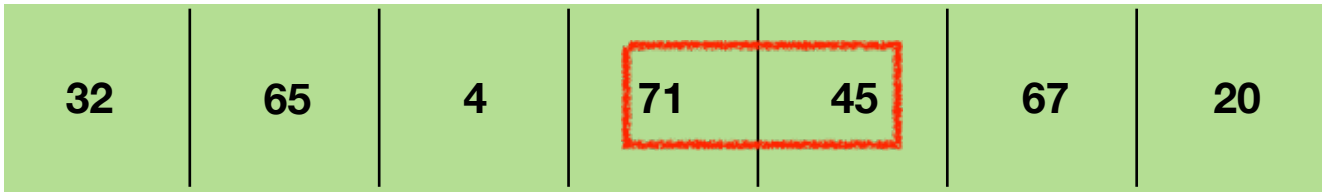


71>4

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

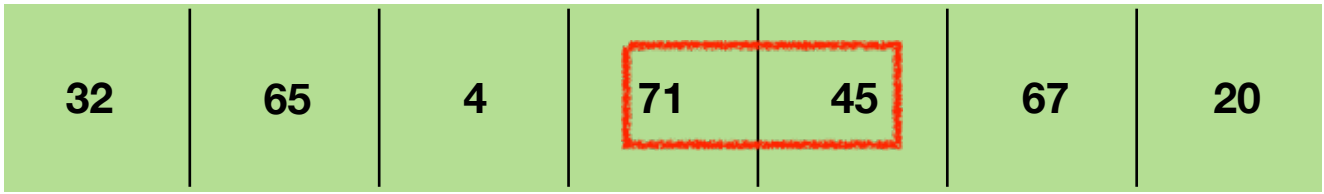
First Pass:



Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

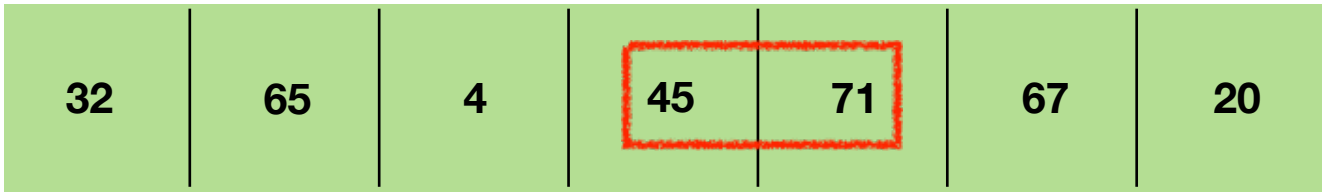


71>45

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

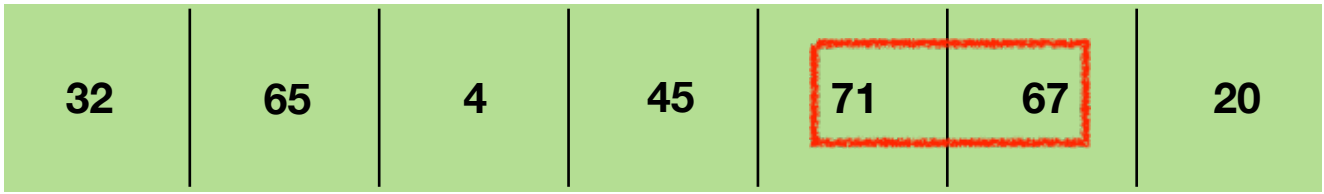


71>45

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

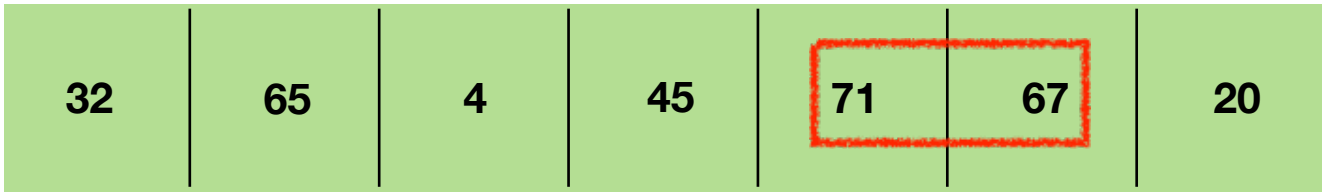
First Pass:



Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

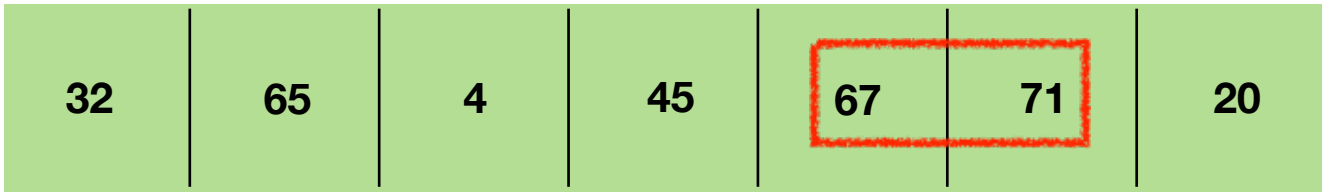


71>67

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

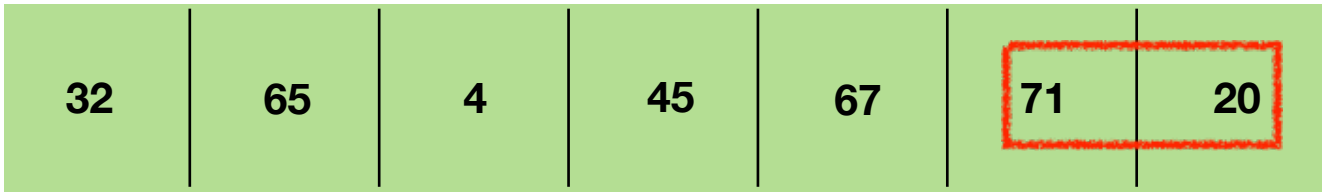


71>67

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

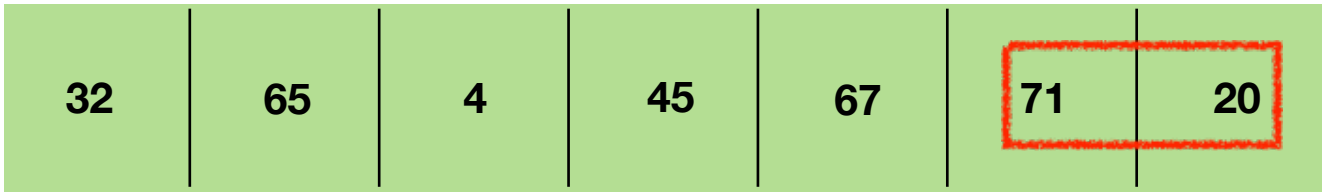
First Pass:



Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

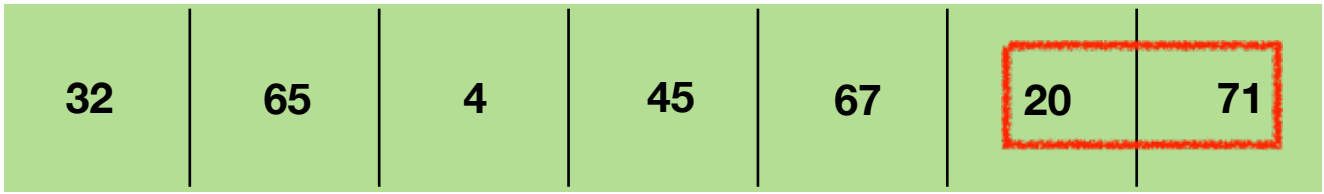


71>20

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:



71>20

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

32	65	4	45	67	20	71

Bubble Sort Algorithm

Sort the following array from lowest number to greatest number

First Pass:

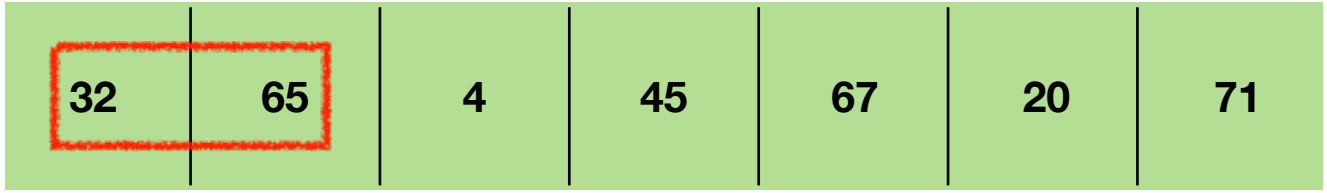
32	65	4	45	67	20	71

Swaps occurred in the first pass —> next pass

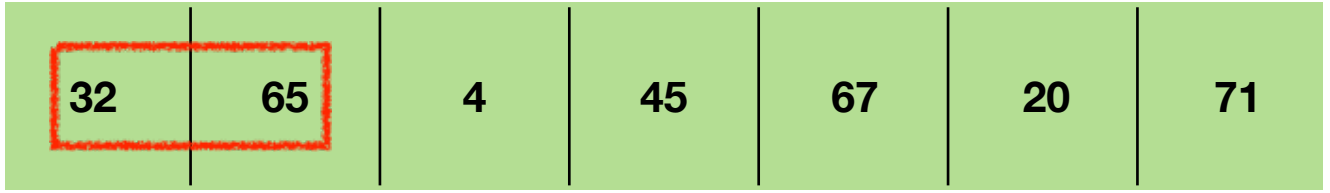
Bubble Sort - Second Pass

32	65	4	45	67	20	71

Bubble Sort - Second Pass

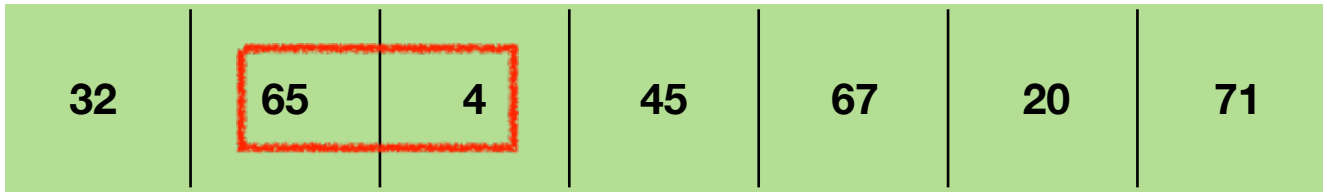


Bubble Sort - Second Pass

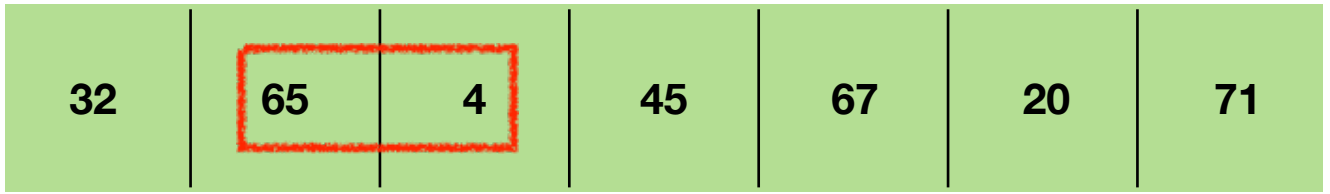


32<65

Bubble Sort - Second Pass

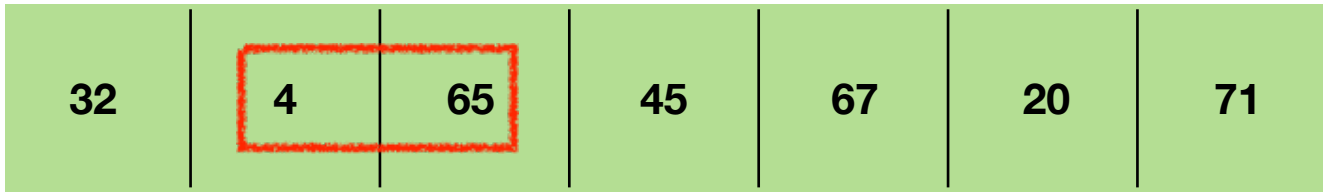


Bubble Sort - Second Pass



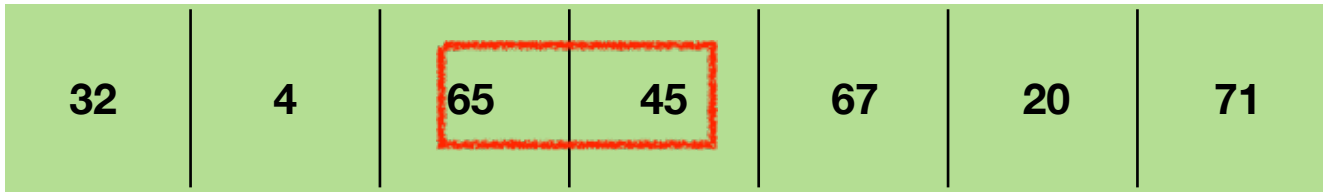
65>4

Bubble Sort - Second Pass

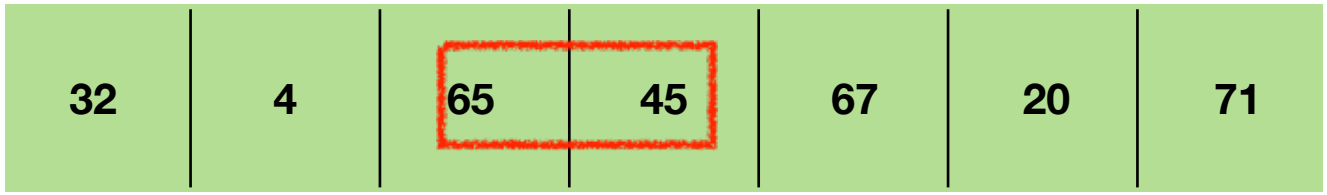


65>4

Bubble Sort - Second Pass

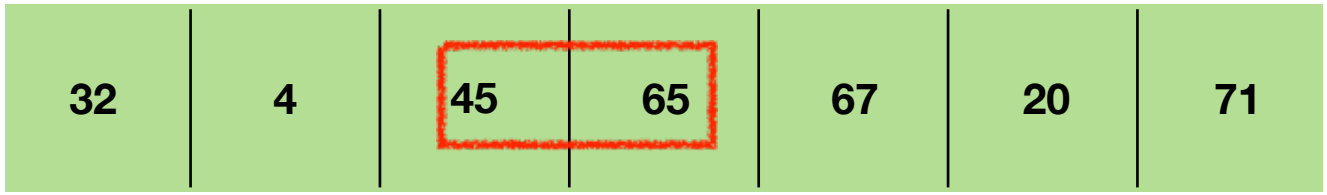


Bubble Sort - Second Pass



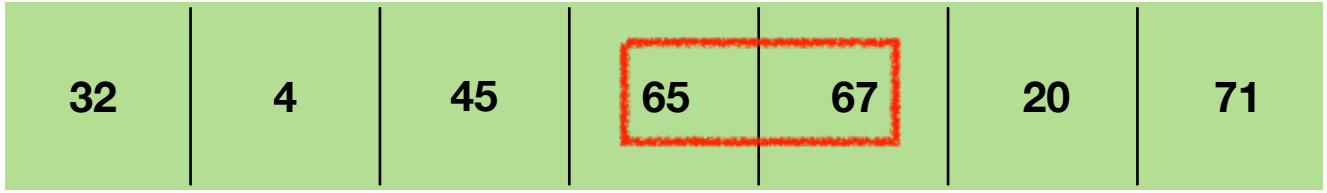
65>45

Bubble Sort - Second Pass

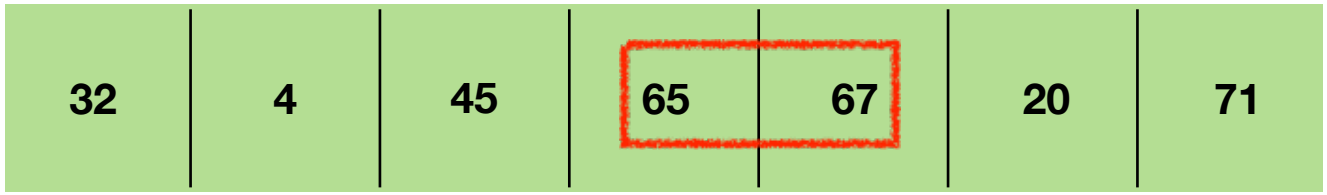


65>45

Bubble Sort - Second Pass

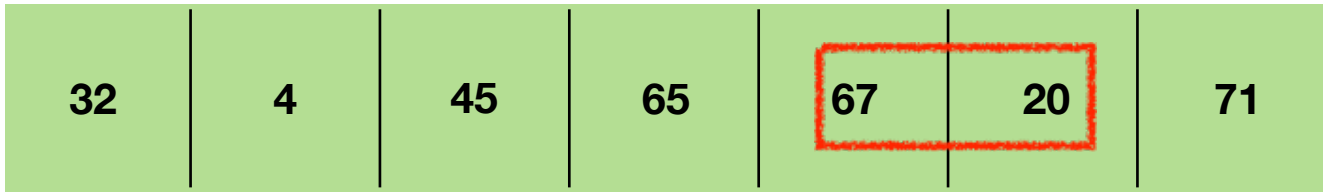


Bubble Sort - Second Pass

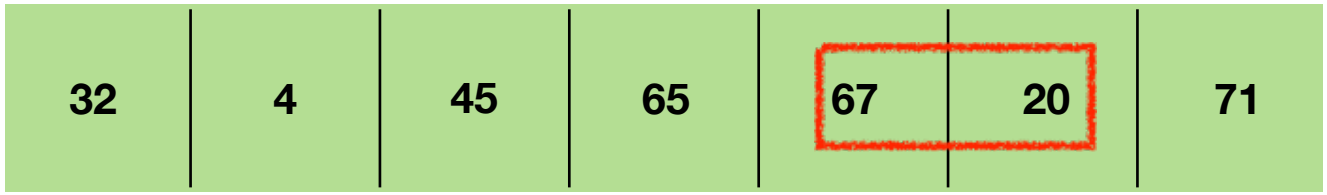


65<67

Bubble Sort - Second Pass

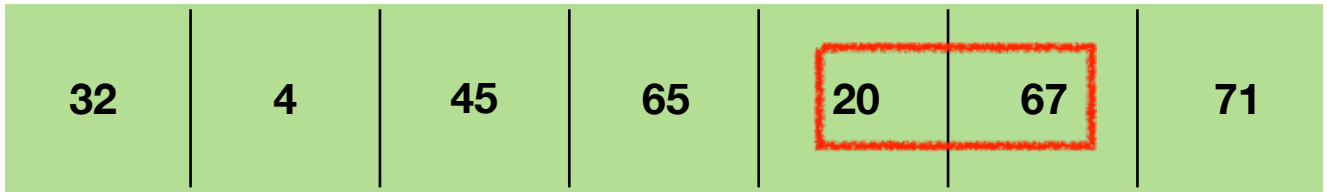


Bubble Sort - Second Pass



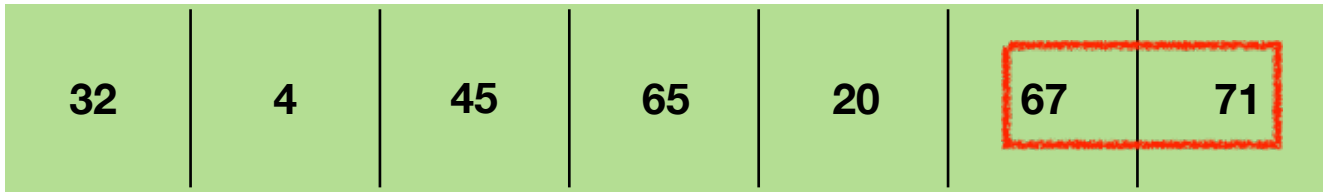
67>20

Bubble Sort - Second Pass



67>20

Bubble Sort - Second Pass



67<71

Bubble Sort - Second Pass

32	4	45	65	20	67	71

Bubble Sort - Second Pass

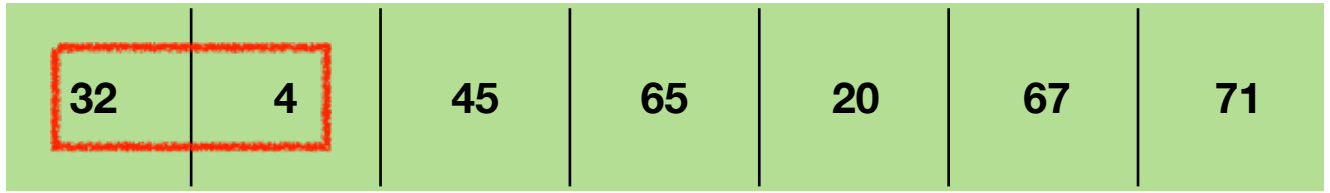
32	4	45	65	20	67	71

Swaps occurred in the second pass —> next pass

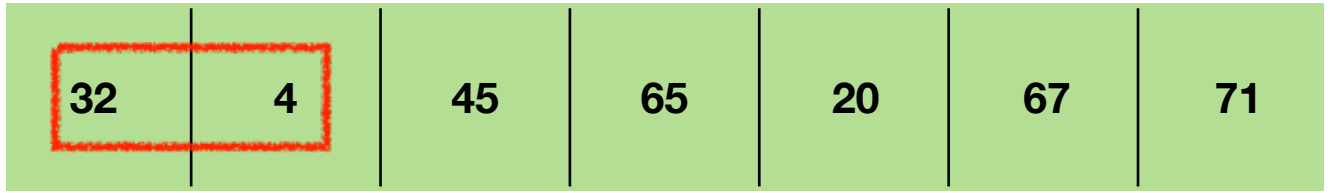
Bubble Sort - Third Pass

32	4	45	65	20	67	71

Bubble Sort - Third Pass

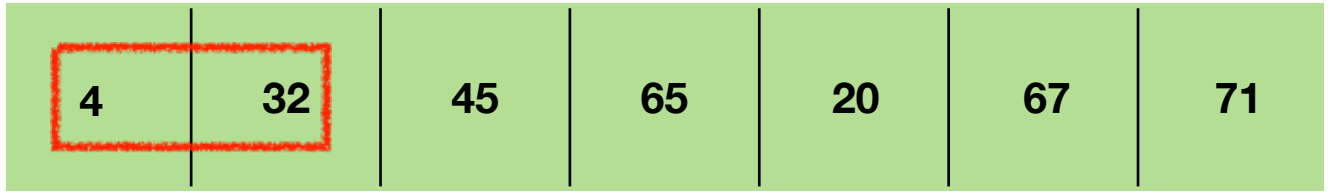


Bubble Sort - Third Pass



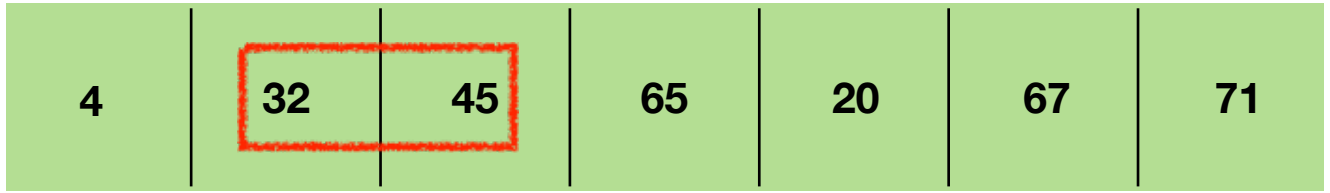
32>4

Bubble Sort - Third Pass

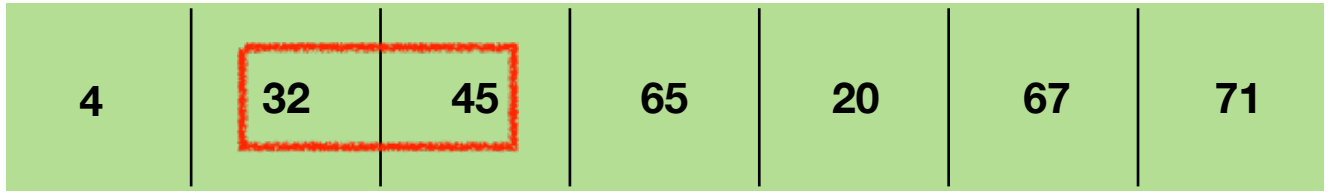


32>4

Bubble Sort - Third Pass

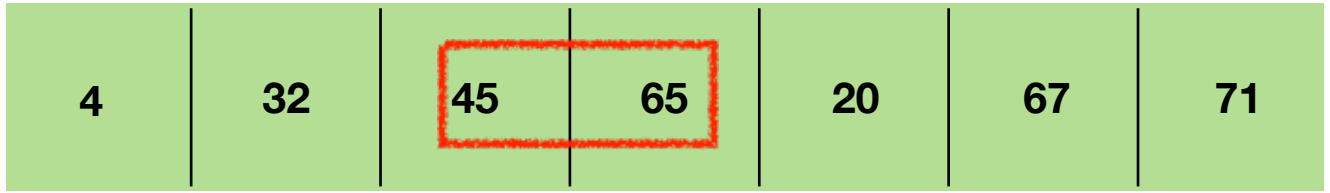


Bubble Sort - Third Pass

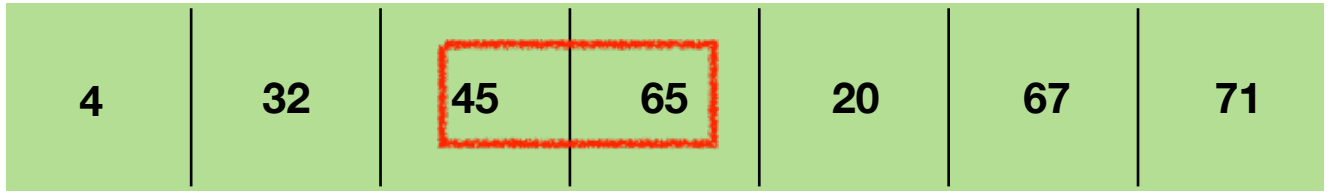


32<45

Bubble Sort - Third Pass

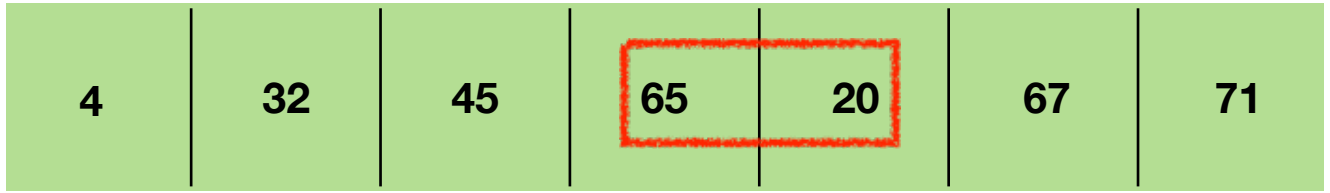


Bubble Sort - Third Pass

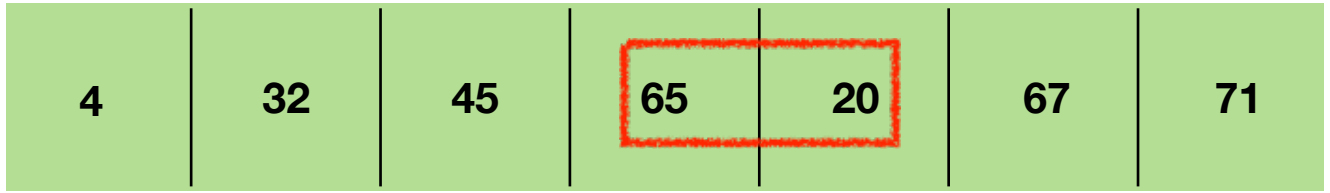


45<65

Bubble Sort - Third Pass

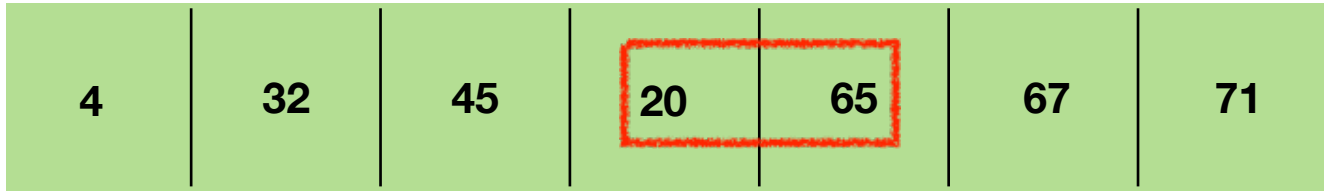


Bubble Sort - Third Pass



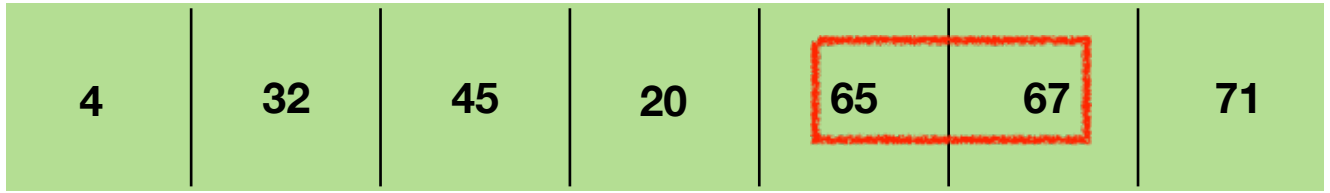
65>20

Bubble Sort - Third Pass

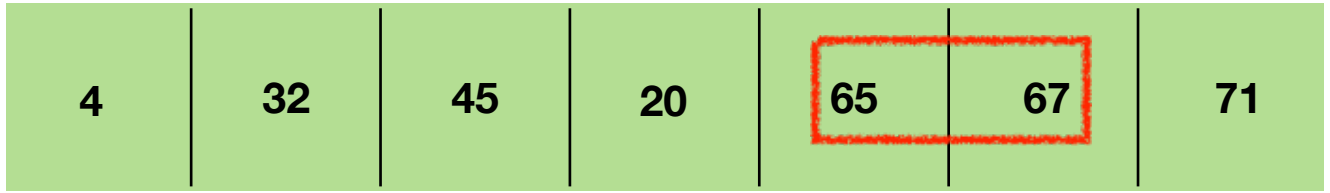


65>20

Bubble Sort - Third Pass

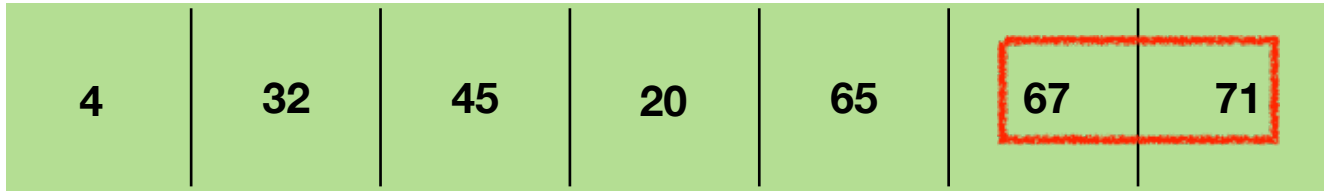


Bubble Sort - Third Pass

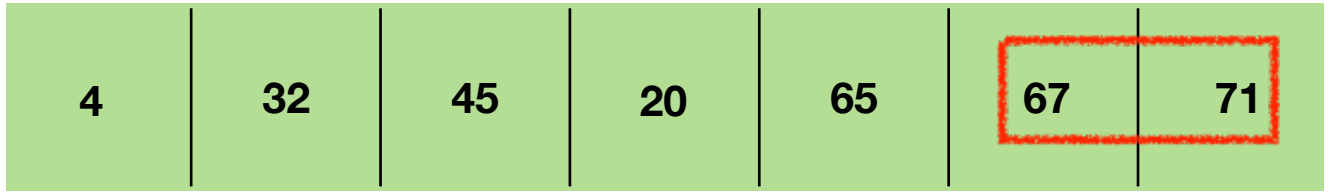


65<67

Bubble Sort - Third Pass



Bubble Sort - Third Pass



67<71

Bubble Sort - Third Pass

4	32	45	20	65	67	71

Bubble Sort - Third Pass

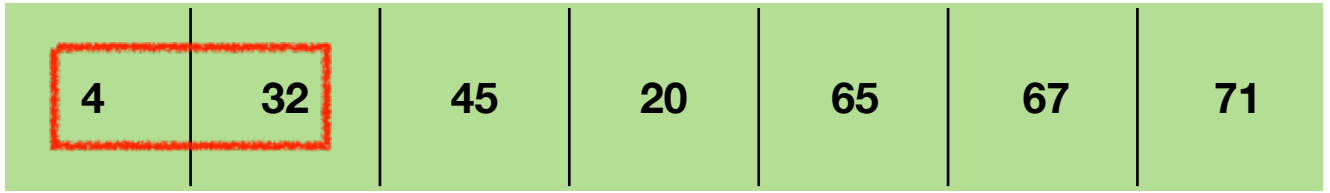
4	32	45	20	65	67	71

Swaps occurred in the third pass —> next pass

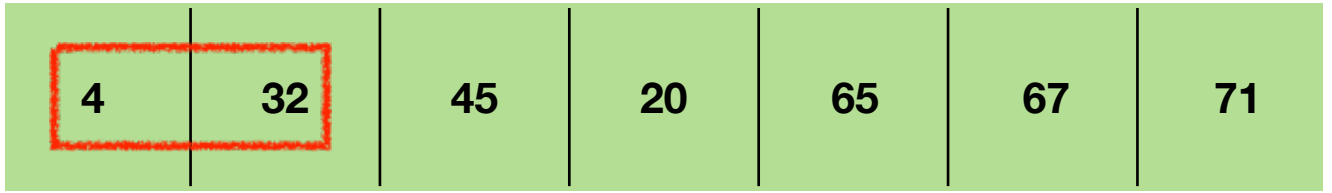
Bubble Sort - Fourth Pass

4	32	45	20	65	67	71

Bubble Sort - Fourth Pass

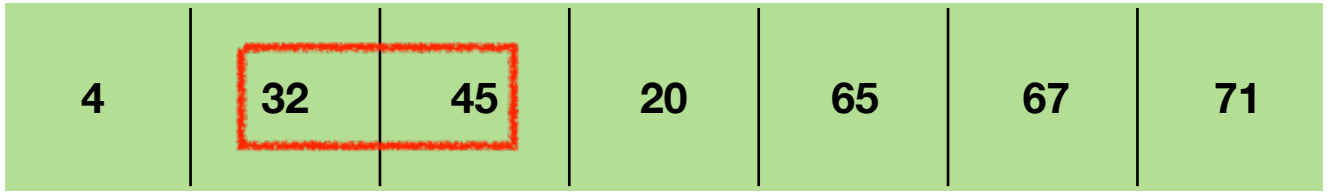


Bubble Sort - Fourth Pass

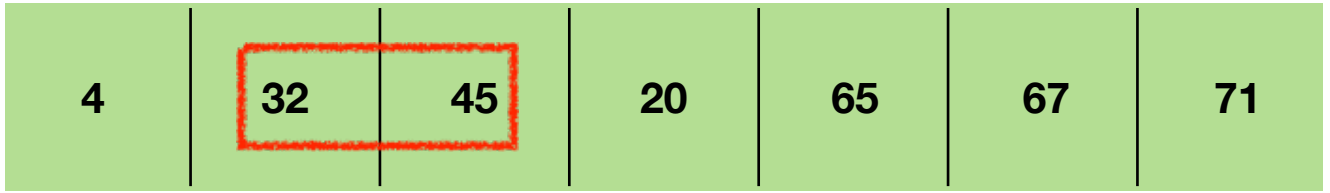


4<32

Bubble Sort - Fourth Pass

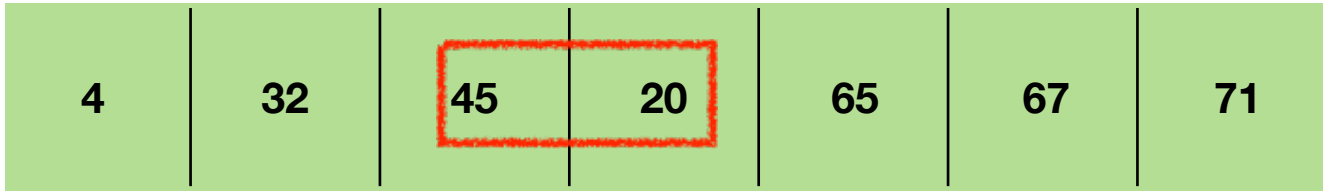


Bubble Sort - Fourth Pass

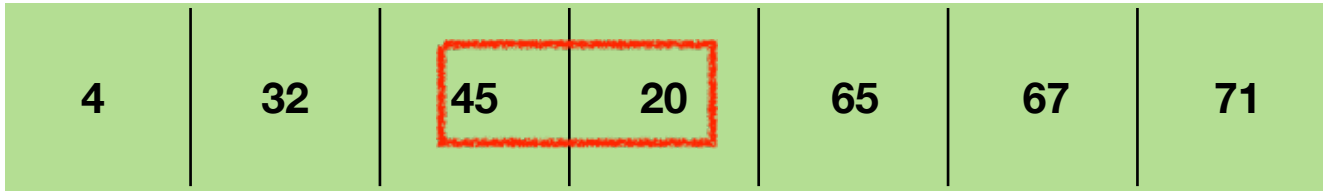


32<45

Bubble Sort - Fourth Pass

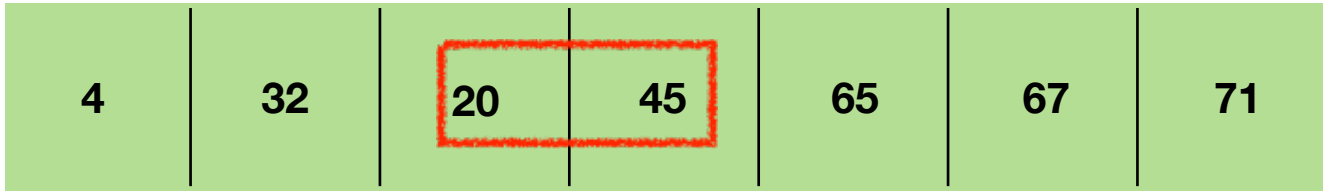


Bubble Sort - Fourth Pass



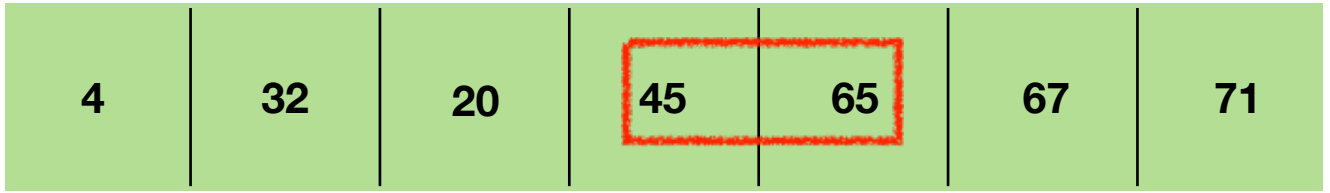
45>20

Bubble Sort - Fourth Pass

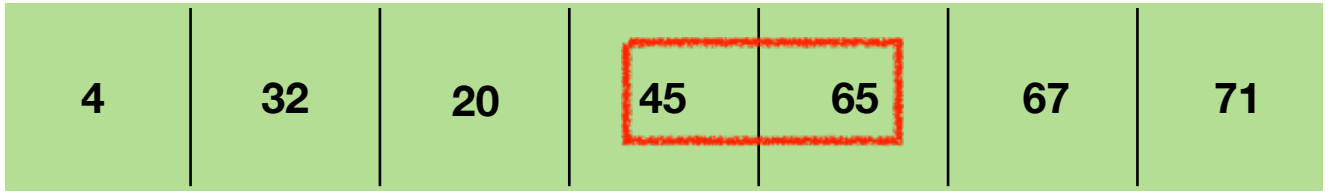


45>20

Bubble Sort - Fourth Pass

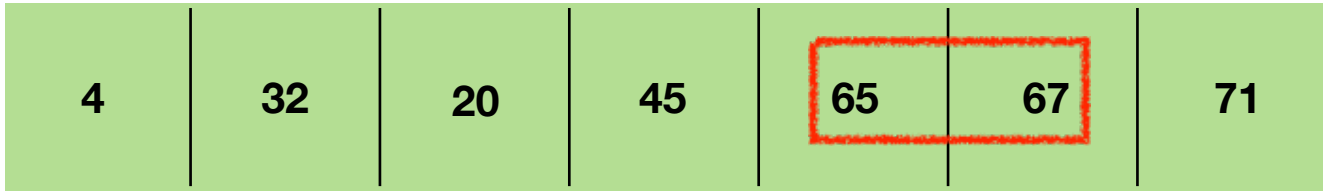


Bubble Sort - Fourth Pass

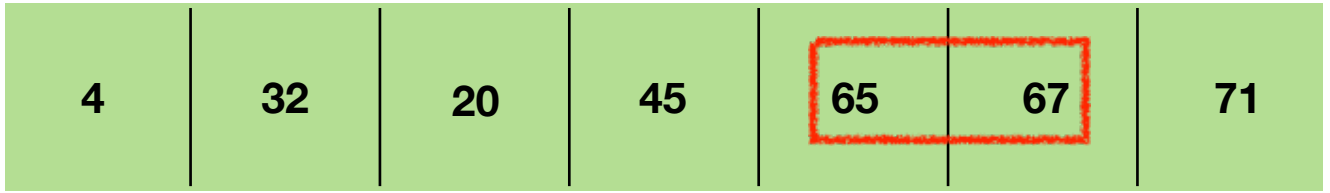


45<65

Bubble Sort - Fourth Pass

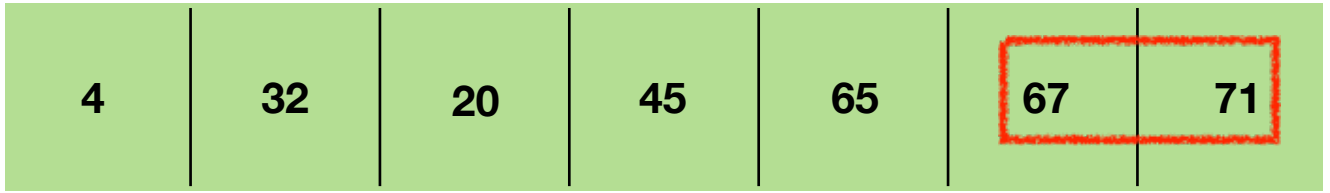


Bubble Sort - Fourth Pass

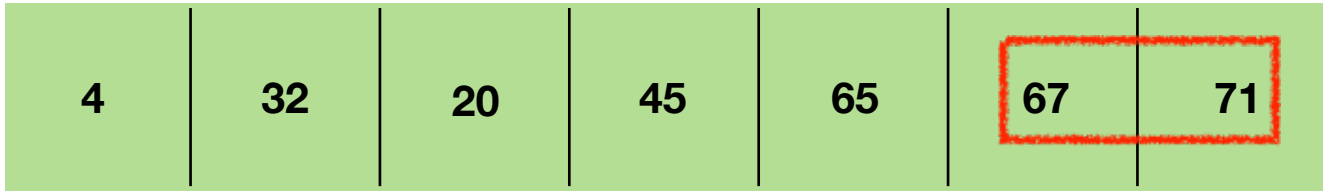


65<67

Bubble Sort - Fourth Pass



Bubble Sort - Fourth Pass

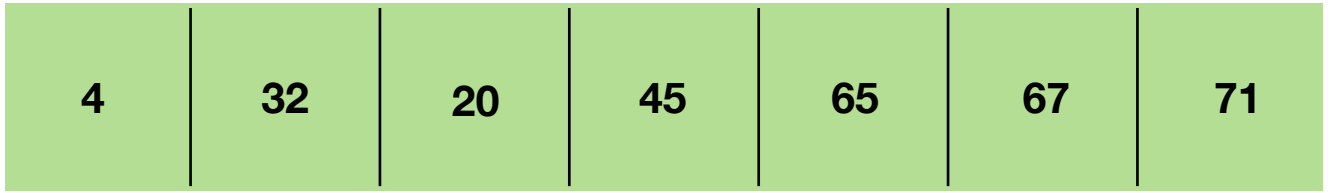


67<71

Bubble Sort - Fourth Pass

4	32	20	45	65	67	71

Bubble Sort - Fourth Pass

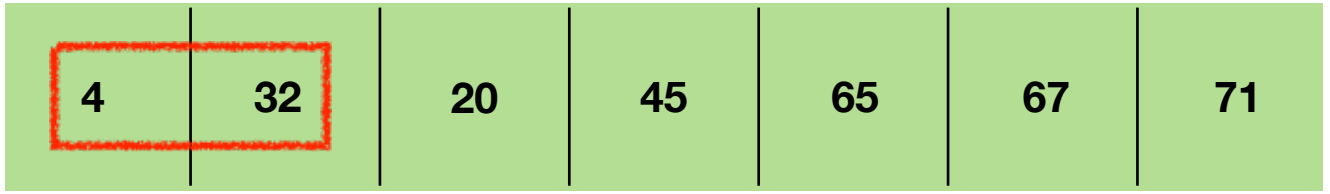


Swaps occurred in the fourth pass —> next pass

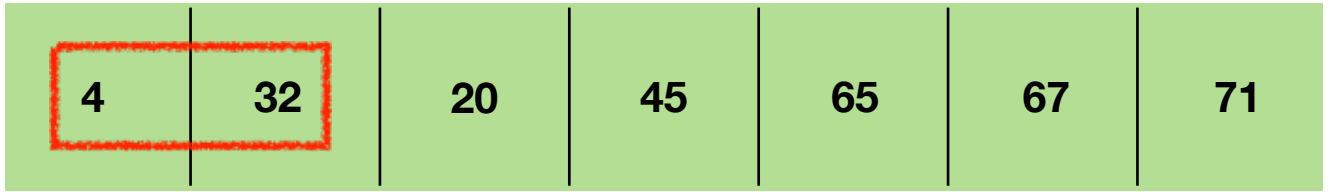
Bubble Sort - Fifth Pass

4	32	20	45	65	67	71

Bubble Sort - Fifth Pass

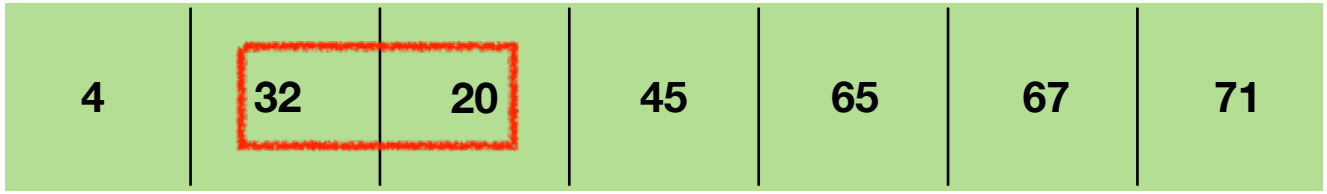


Bubble Sort - Fifth Pass

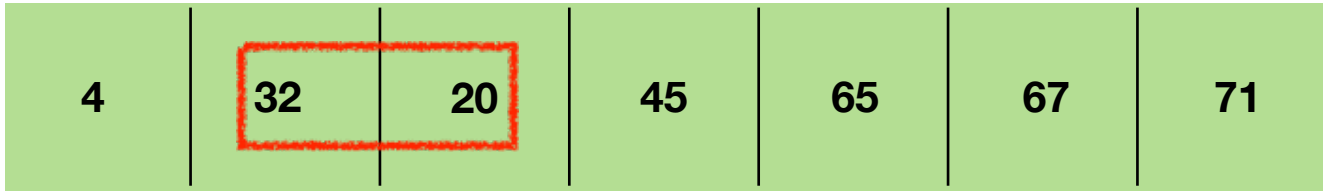


4<32

Bubble Sort - Fifth Pass

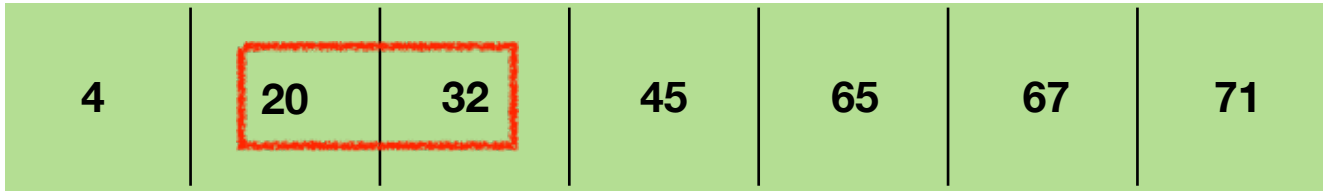


Bubble Sort - Fifth Pass



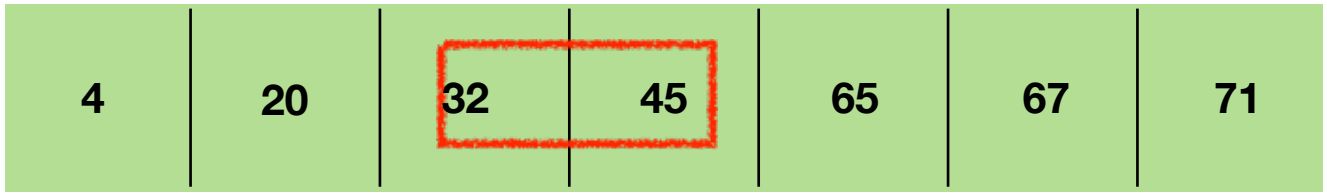
32>20

Bubble Sort - Fifth Pass

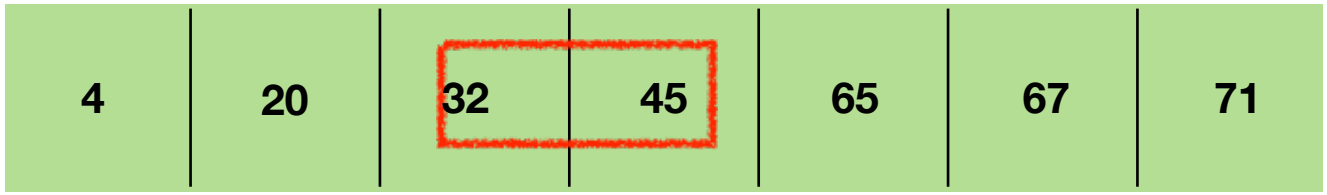


32>20

Bubble Sort - Fifth Pass

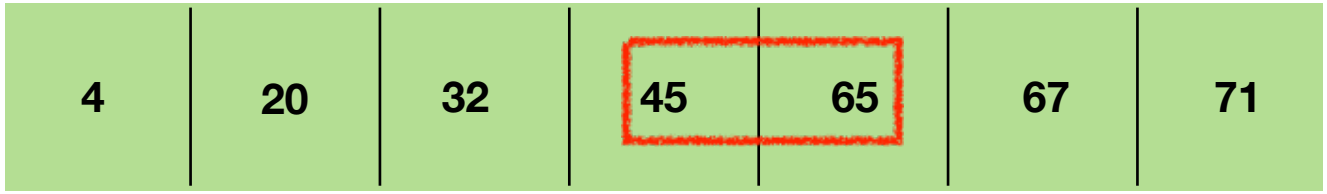


Bubble Sort - Fifth Pass

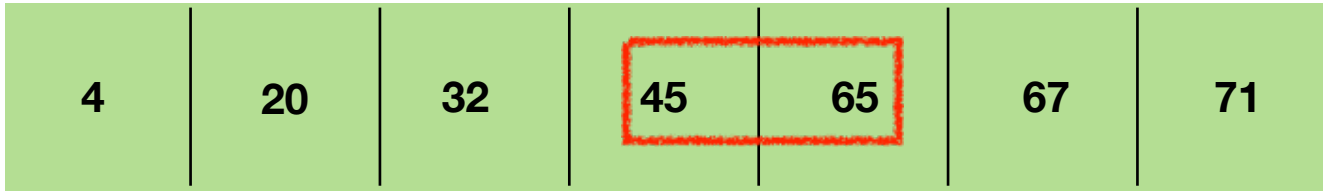


32<45

Bubble Sort - Fifth Pass

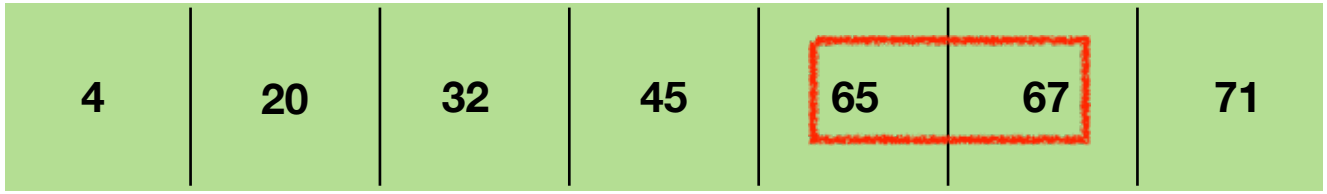


Bubble Sort - Fifth Pass

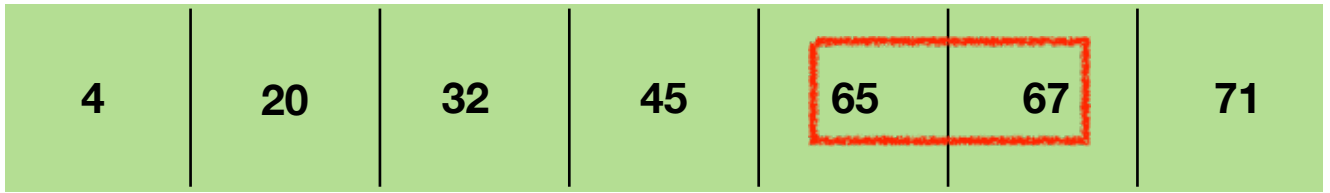


45<65

Bubble Sort - Fifth Pass

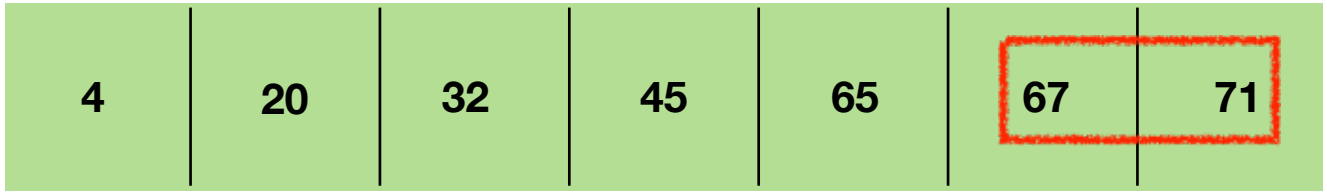


Bubble Sort - Fifth Pass

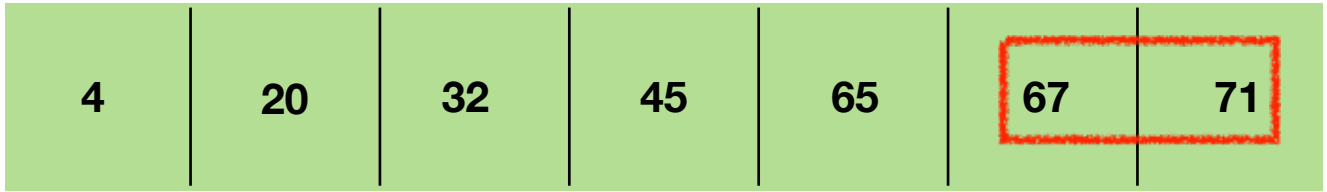


65<67

Bubble Sort - Fifth Pass



Bubble Sort - Fifth Pass



67<71

Bubble Sort - Fifth Pass

4	20	32	45	65	67	71

Bubble Sort - Fifth Pass

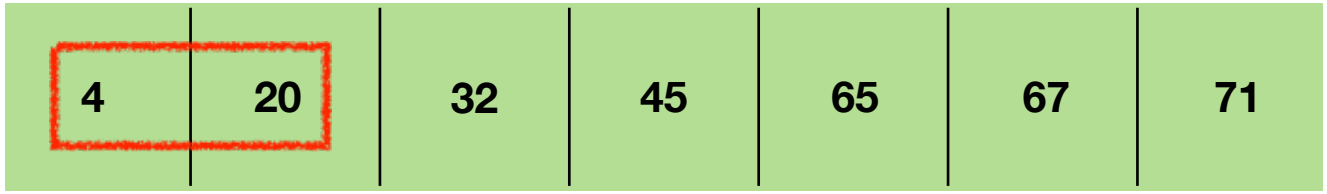
4	20	32	45	65	67	71

Swaps occurred in the fifth pass —> next pass

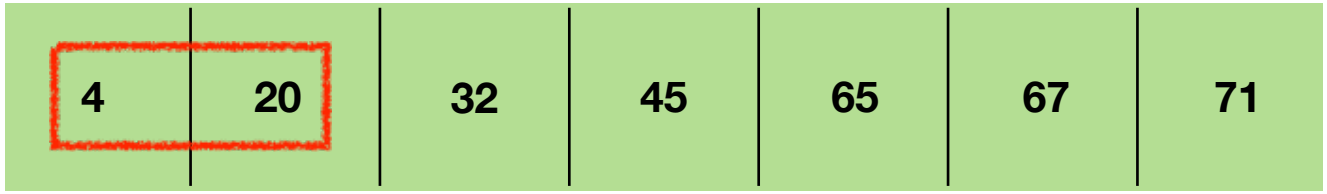
Bubble Sort - Sixth Pass

4	20	32	45	65	67	71

Bubble Sort - Sixth Pass

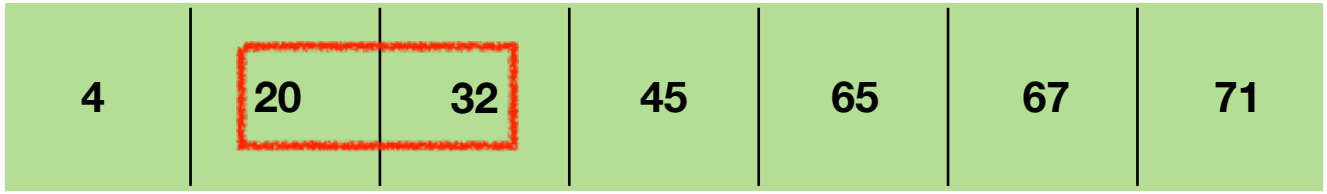


Bubble Sort - Sixth Pass

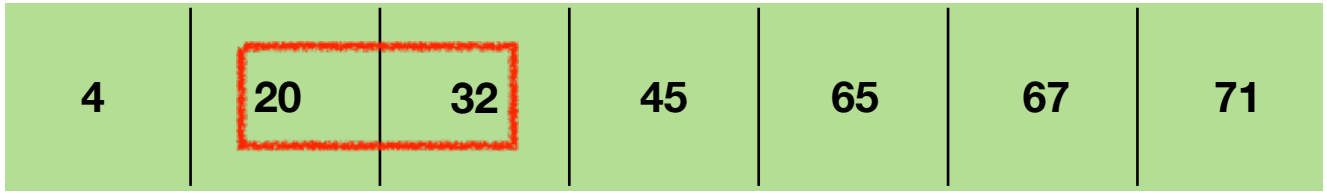


4<20

Bubble Sort - Sixth Pass

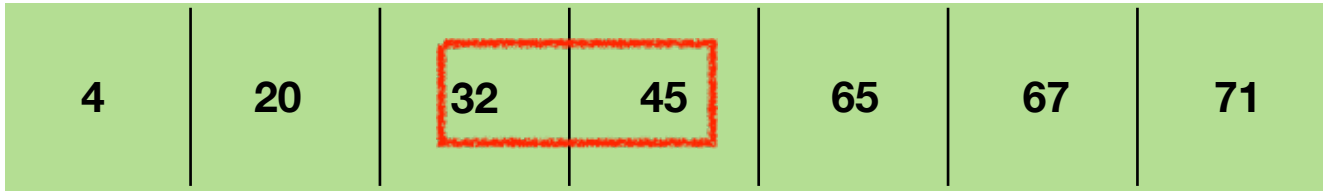


Bubble Sort - Sixth Pass

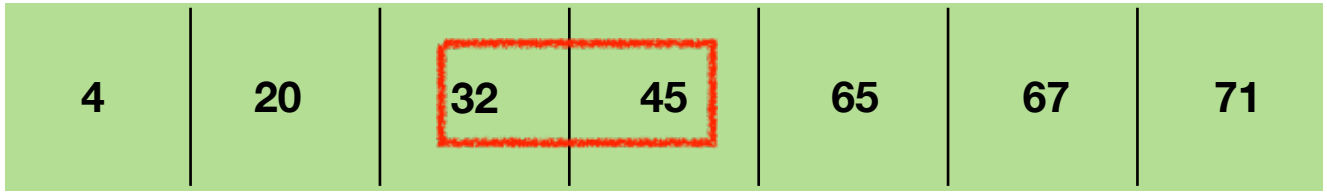


20<32

Bubble Sort - Sixth Pass

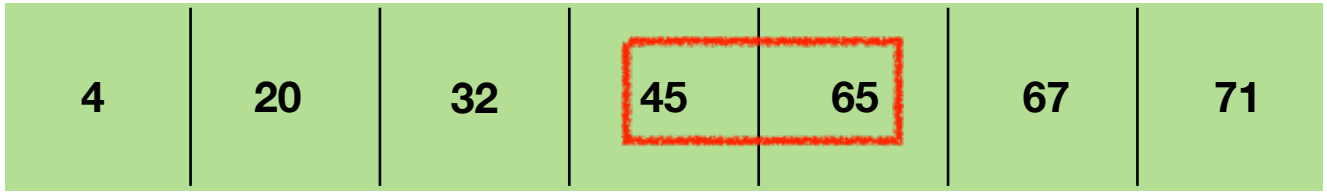


Bubble Sort - Sixth Pass

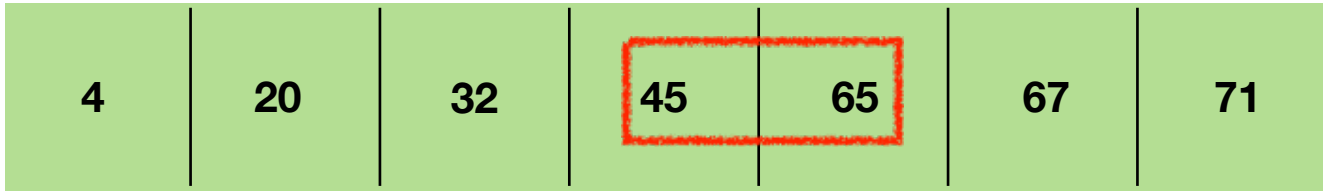


32<45

Bubble Sort - Sixth Pass

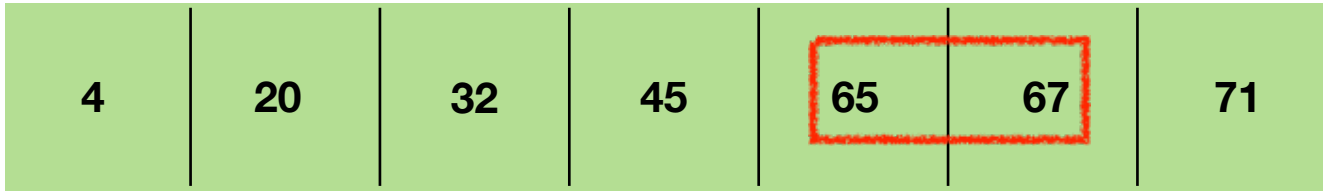


Bubble Sort - Sixth Pass

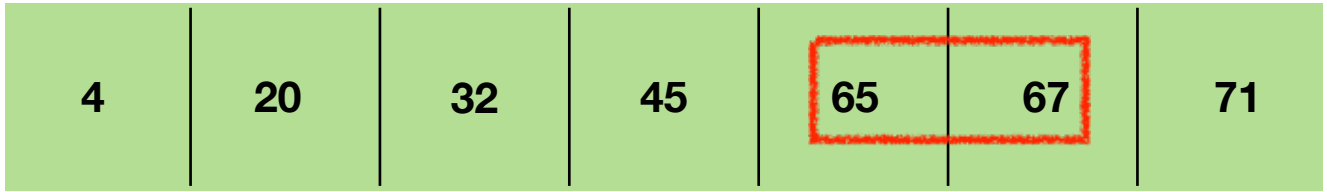


45<65

Bubble Sort - Sixth Pass

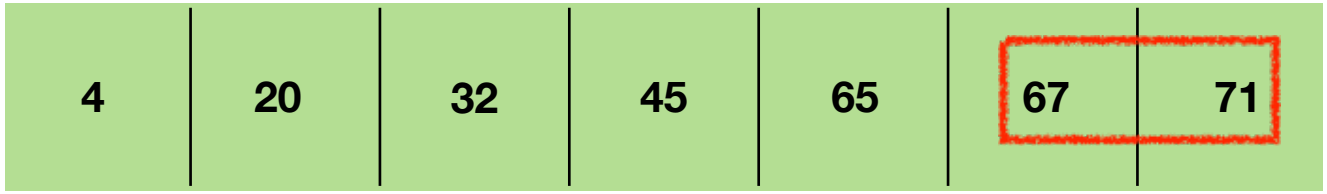


Bubble Sort - Sixth Pass

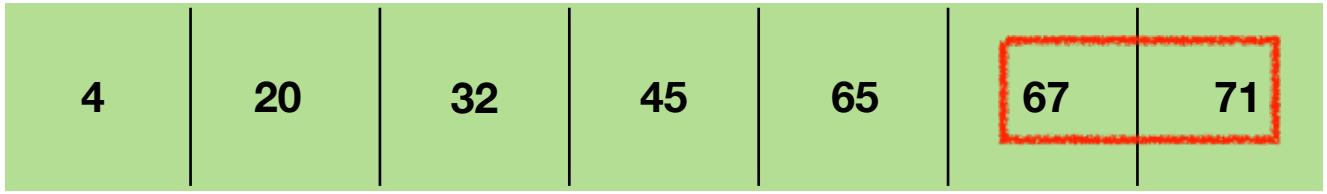


65<67

Bubble Sort - Sixth Pass



Bubble Sort - Sixth Pass



67<71

Bubble Sort - Sixth Pass

4	20	32	45	65	67	71

Bubble Sort - Sixth Pass

4	20	32	45	65	67	71

No swaps in the sixth pass: Done

Bubble Sort Pseudocode

```
bubbleSort(Array A : list of sortable items, n)
{
    n = A.size //1 unit time.
    flag = false
    do     //each pass take O(n) ///Question? How many passes?
        flag = false   //O(1)
        for i = 1 to n-1   // (n-1)* O(1) =O(n)
            if A[i] > A[i+1]          // 1 line
               swap(A[i], A[i+1]) // 3 lines
               flag = true.  // 1 line
    while(flag)
}
```

What is the running time?

Worst-case Running Time T(n)

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

Worst-case Running Time T(n)

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

$$T(n) = 2 * \#ofPass + 4 * (n-1) * \#ofPass + 1$$

Worst-case Running Time T(n)

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

$$T(n) = 2 * \#ofPass + 4 * (n-1) * \#ofPass + 1$$

#ofPass = ?

Bubble Sort - Worst Case

A is the reverse order.

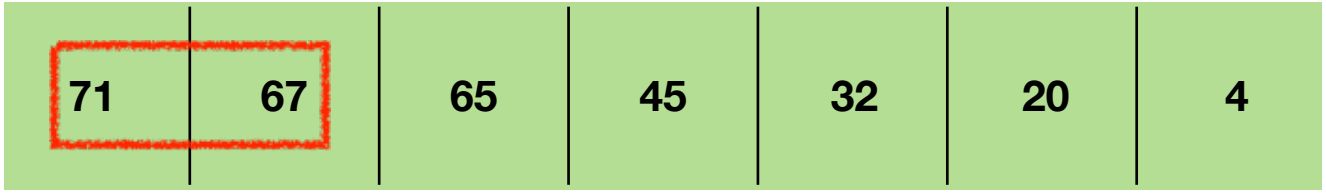
First Pass:

71	67	65	45	32	20	4

Bubble Sort - Worst Case

A is the reverse order.

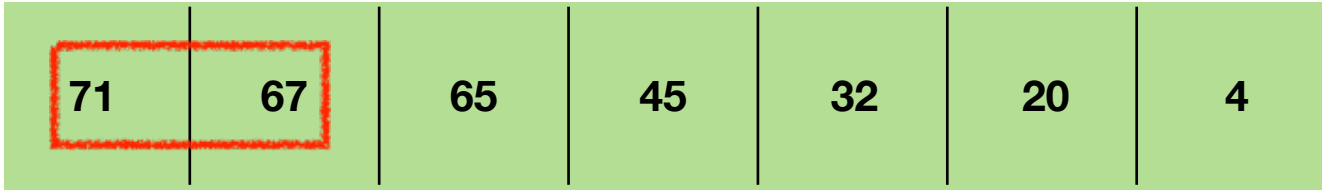
First Pass:



Bubble Sort - Worst Case

A is the reverse order.

First Pass:

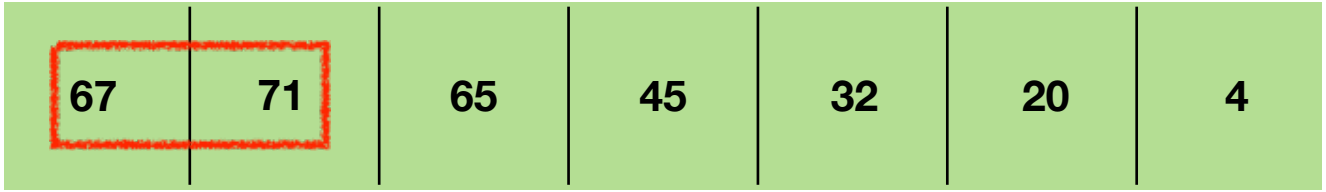


71>67

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

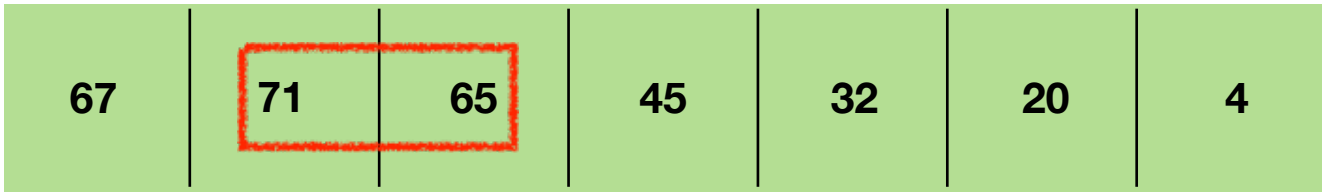


71>67

Bubble Sort - Worst Case

A is the reverse order.

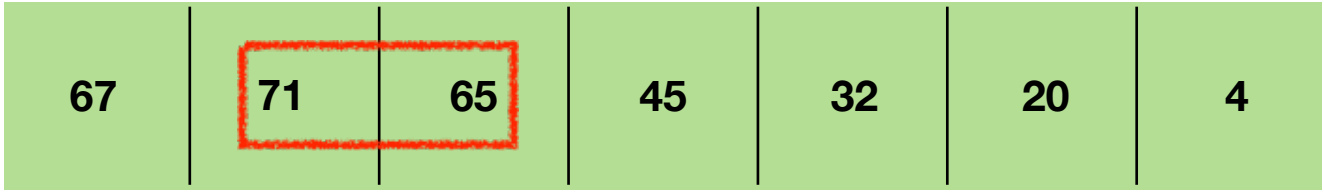
First Pass:



Bubble Sort - Worst Case

A is the reverse order.

First Pass:

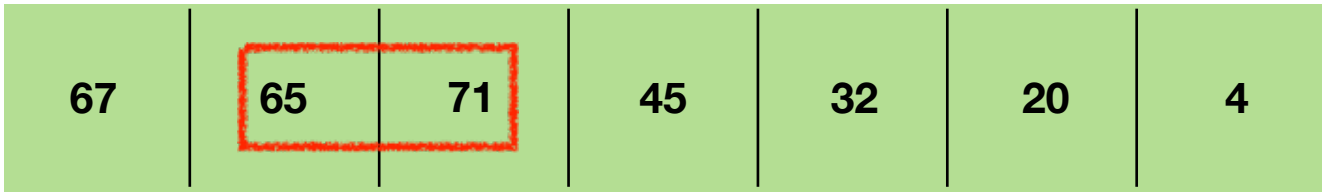


71>65

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

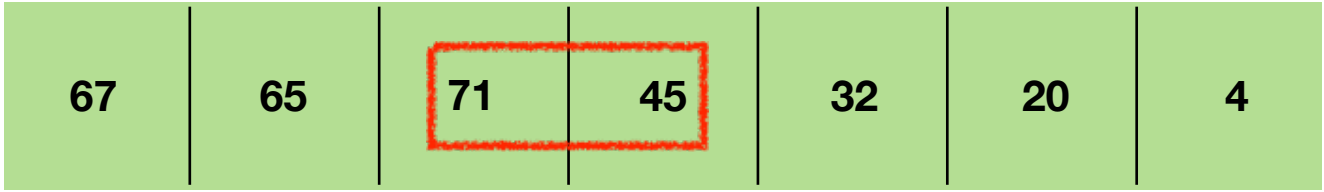


71>65

Bubble Sort - Worst Case

A is the reverse order.

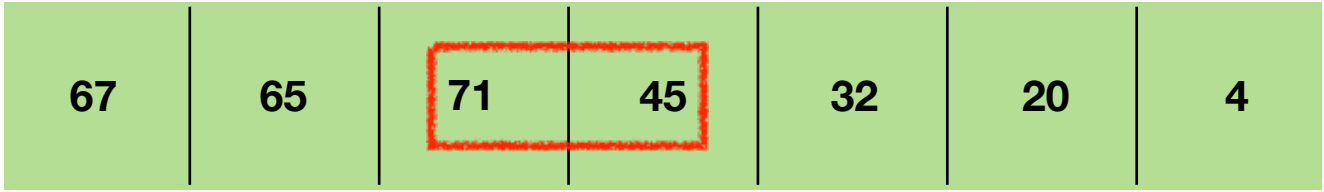
First Pass:



Bubble Sort - Worst Case

A is the reverse order.

First Pass:

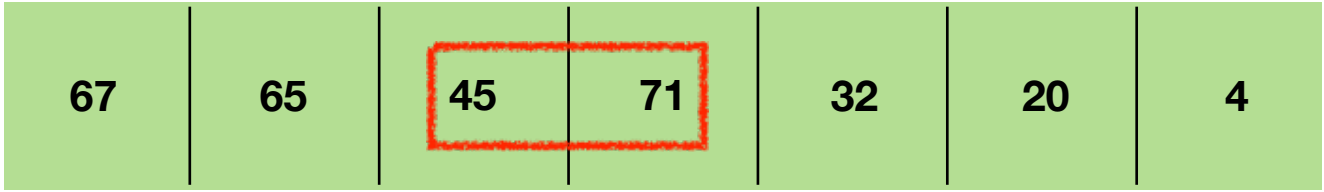


71>45

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

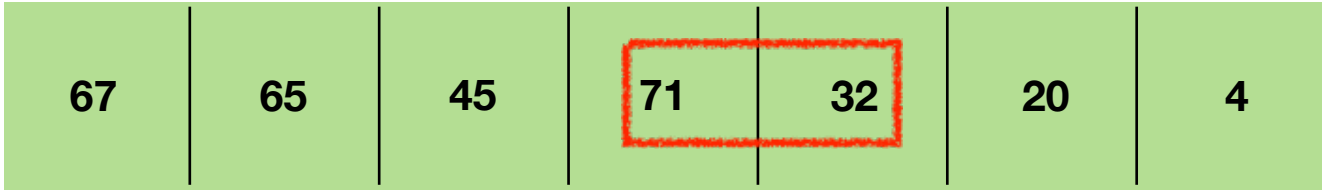


71>45

Bubble Sort - Worst Case

A is the reverse order.

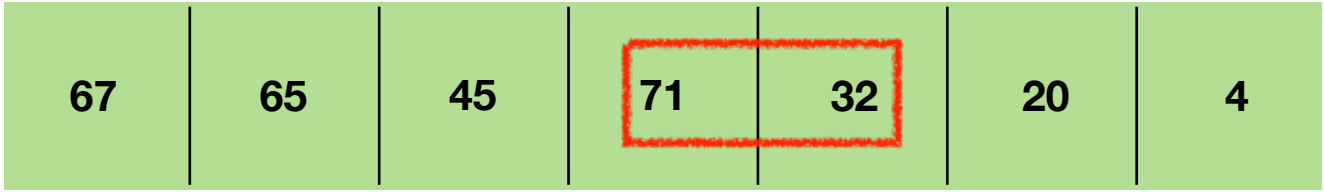
First Pass:



Bubble Sort - Worst Case

A is the reverse order.

First Pass:

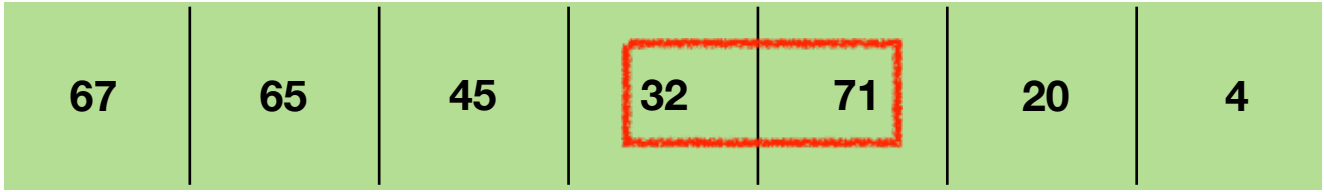


71>32

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

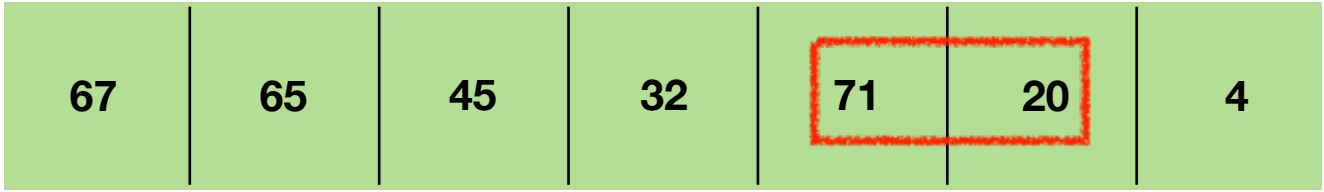


71>32

Bubble Sort - Worst Case

A is the reverse order.

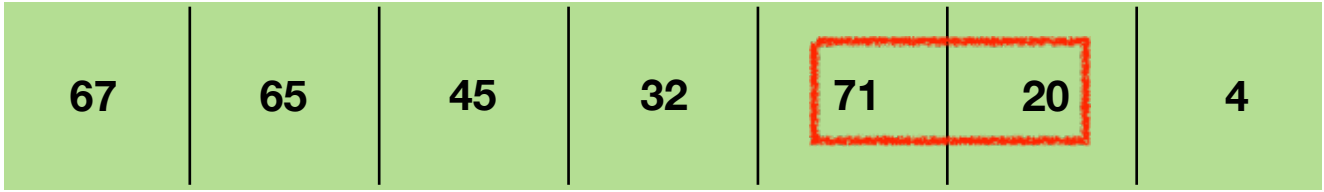
First Pass:



Bubble Sort - Worst Case

A is the reverse order.

First Pass:

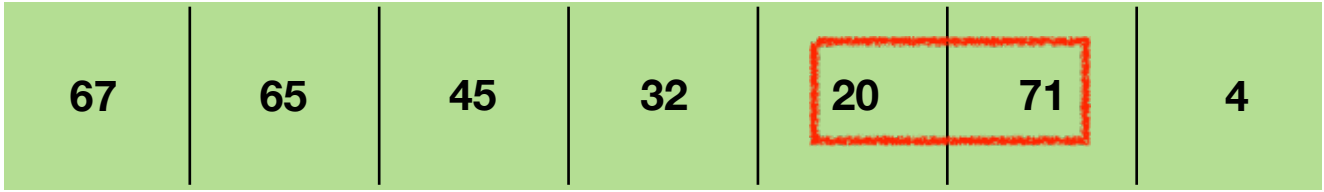


71>20

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

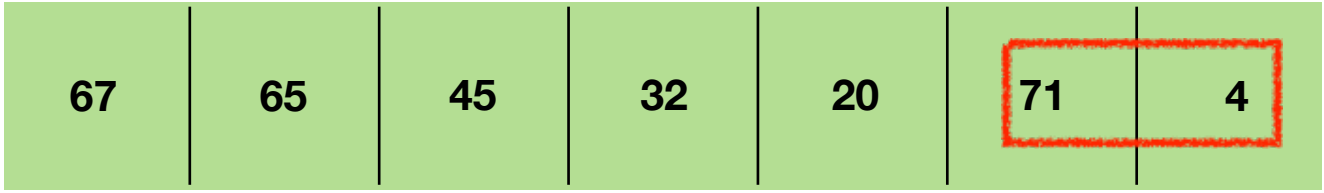


71>20

Bubble Sort - Worst Case

A is the reverse order.

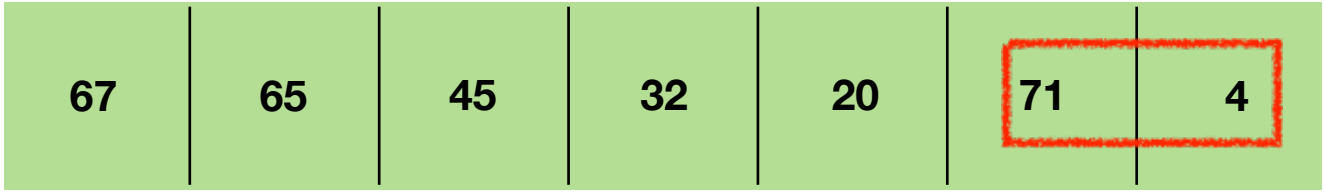
First Pass:



Bubble Sort - Worst Case

A is the reverse order.

First Pass:

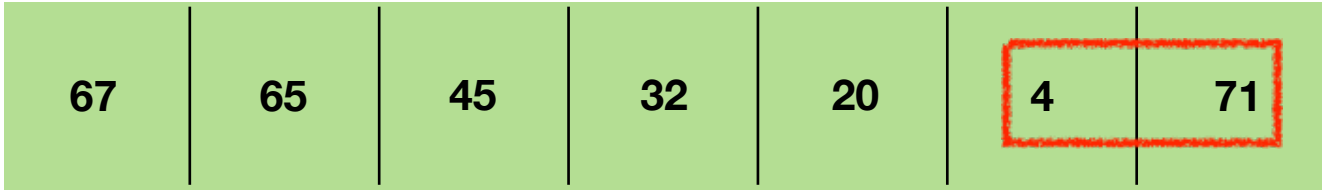


71>4

Bubble Sort - Worst Case

A is the reverse order.

First Pass:



71>4

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

67	65	45	32	20	4	71

Bubble Sort - Worst Case

A is the reverse order.

First Pass:

67	65	45	32	20	4	71

Swaps occurred in the first pass —> Next Pass

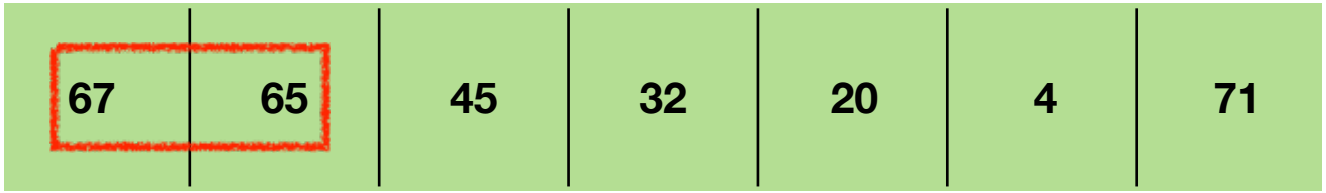
Bubble Sort - Worst Case (cont.)

Second Pass:

67	65	45	32	20	4	71

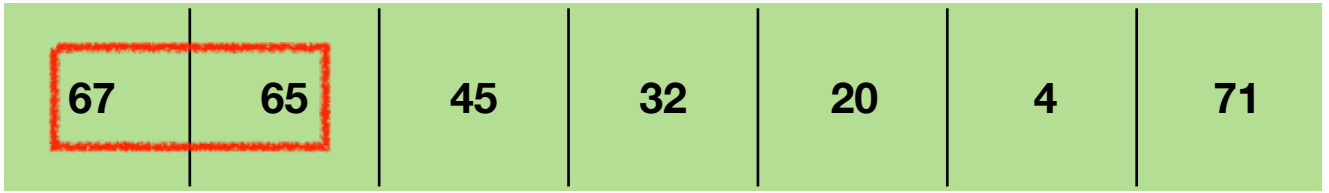
Bubble Sort - Worst Case (cont.)

Second Pass:



Bubble Sort - Worst Case (cont.)

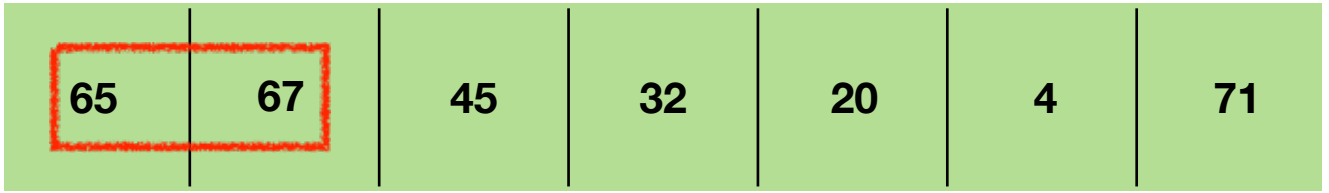
Second Pass:



67>65

Bubble Sort - Worst Case (cont.)

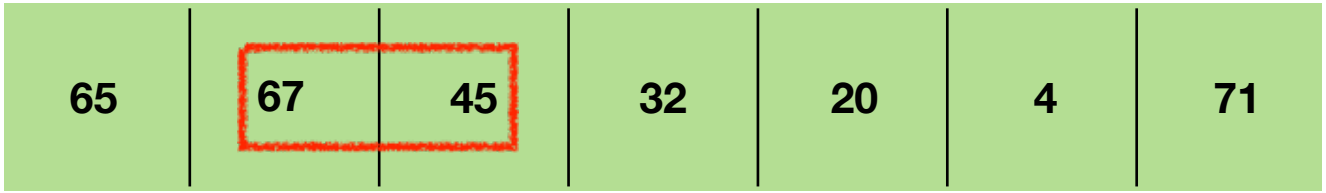
Second Pass:



67>65

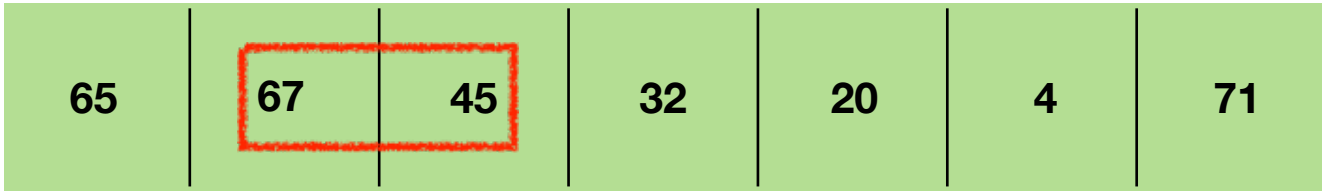
Bubble Sort - Worst Case (cont.)

Second Pass:



Bubble Sort - Worst Case (cont.)

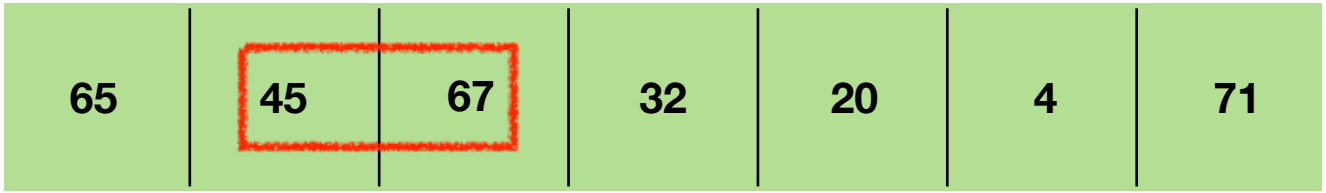
Second Pass:



67>45

Bubble Sort - Worst Case (cont.)

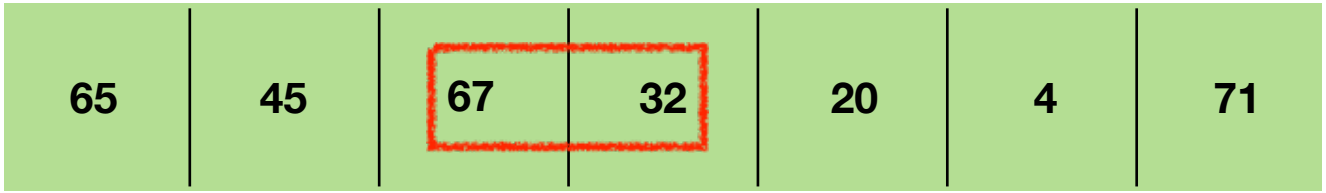
Second Pass:



67>45

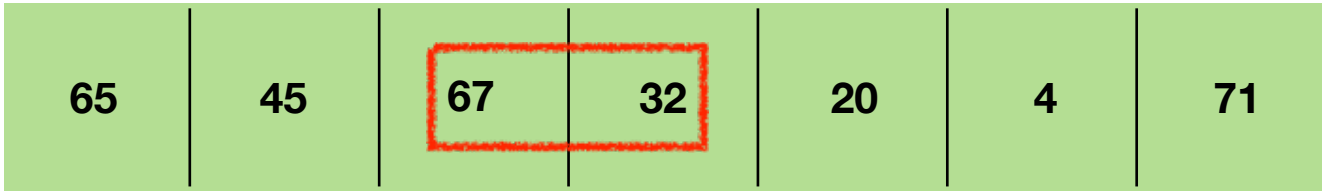
Bubble Sort - Worst Case (cont.)

Second Pass:



Bubble Sort - Worst Case (cont.)

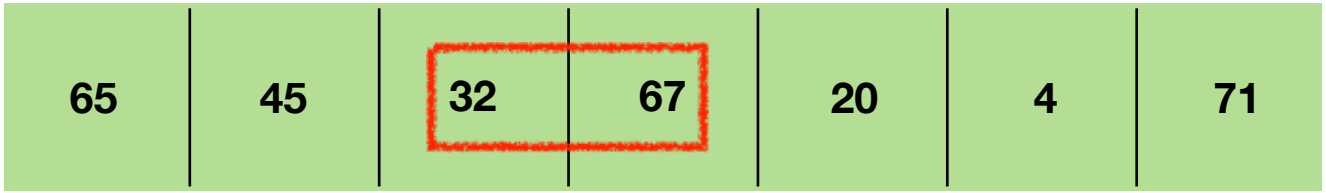
Second Pass:



67>32

Bubble Sort - Worst Case (cont.)

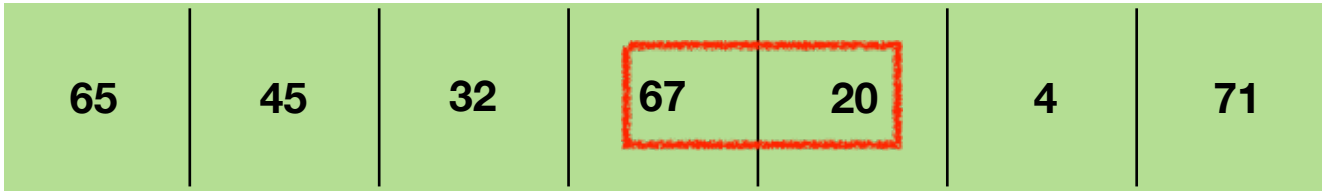
Second Pass:



67>32

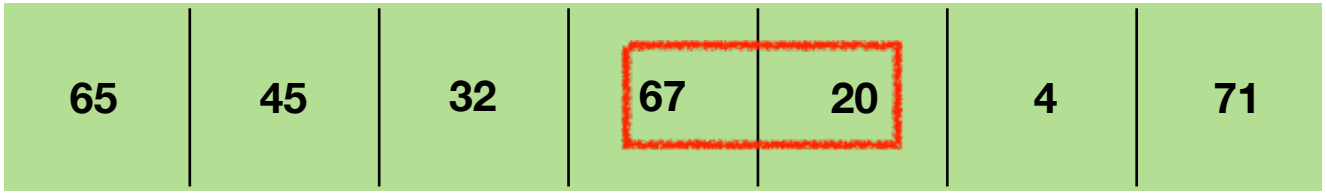
Bubble Sort - Worst Case (cont.)

Second Pass:



Bubble Sort - Worst Case (cont.)

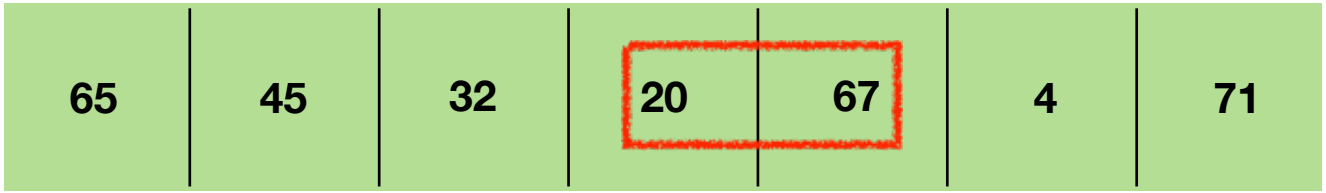
Second Pass:



67>20

Bubble Sort - Worst Case (cont.)

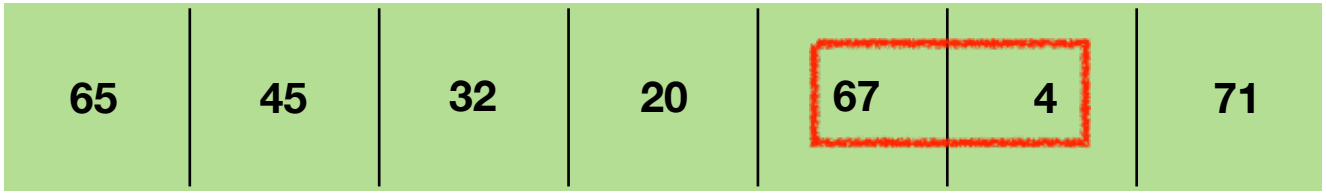
Second Pass:



67>20

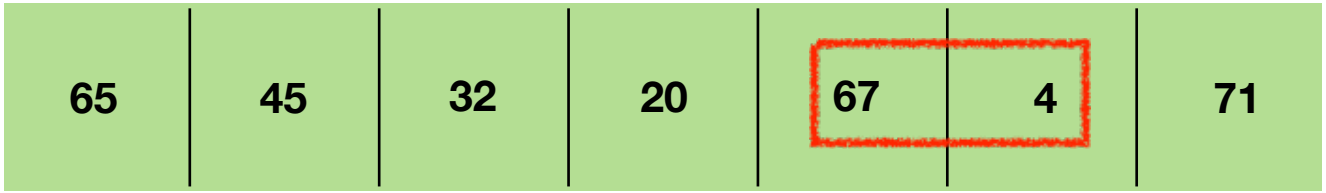
Bubble Sort - Worst Case (cont.)

Second Pass:



Bubble Sort - Worst Case (cont.)

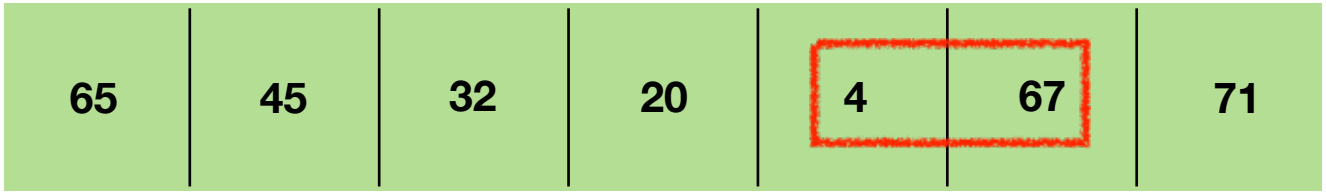
Second Pass:



67>4

Bubble Sort - Worst Case (cont.)

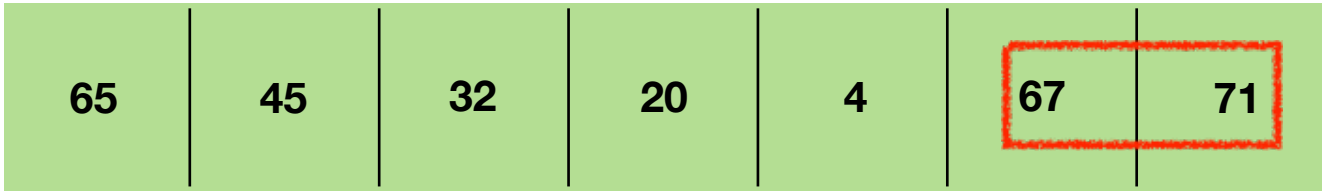
Second Pass:



67>4

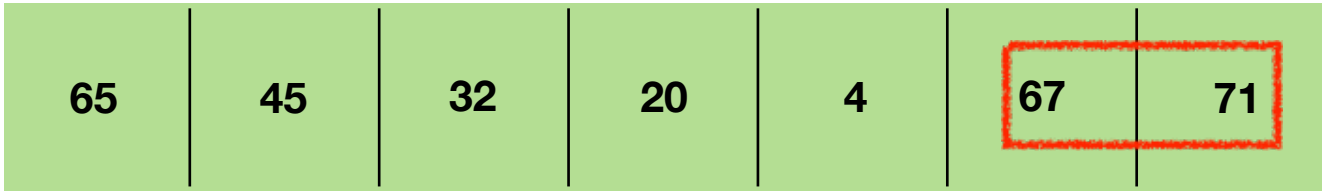
Bubble Sort - Worst Case (cont.)

Second Pass:



Bubble Sort - Worst Case (cont.)

Second Pass:



67<71

Bubble Sort - Worst Case (cont.)

Second Pass:

65	45	32	20	4	67	71

Bubble Sort - Worst Case (cont.)

Second Pass:

65	45	32	20	4	67	71

Swaps occurred in the second pass —> Next Pass

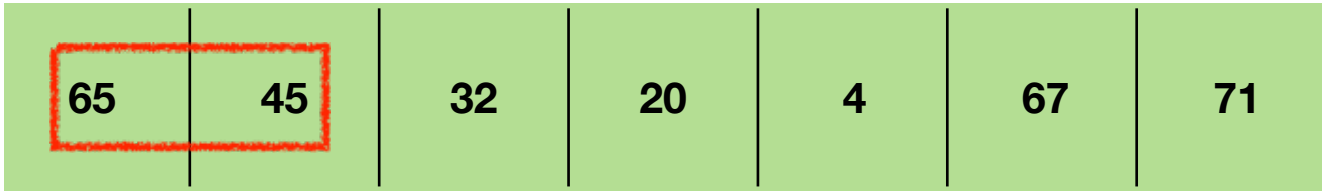
Bubble Sort - Worst Case (cont.)

Third Pass:

65	45	32	20	4	67	71

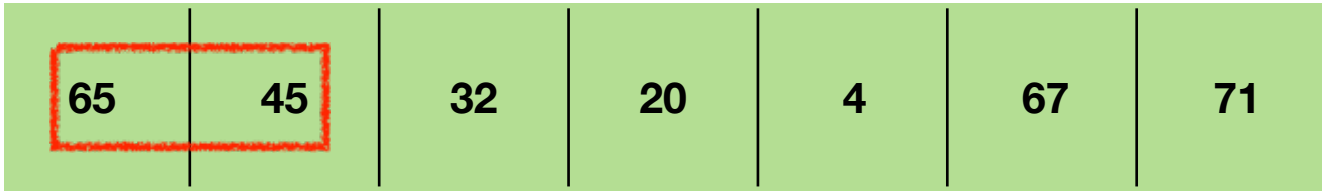
Bubble Sort - Worst Case (cont.)

Third Pass:



Bubble Sort - Worst Case (cont.)

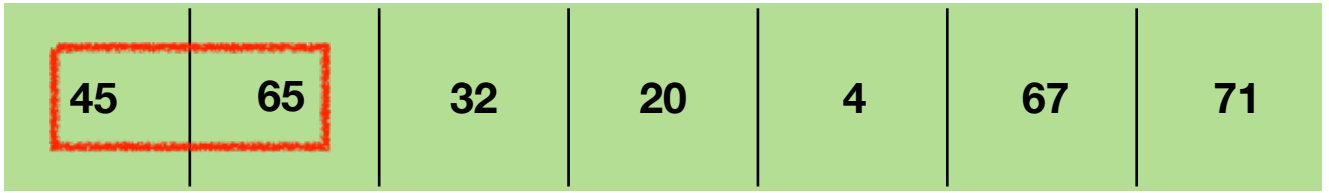
Third Pass:



65>45

Bubble Sort - Worst Case (cont.)

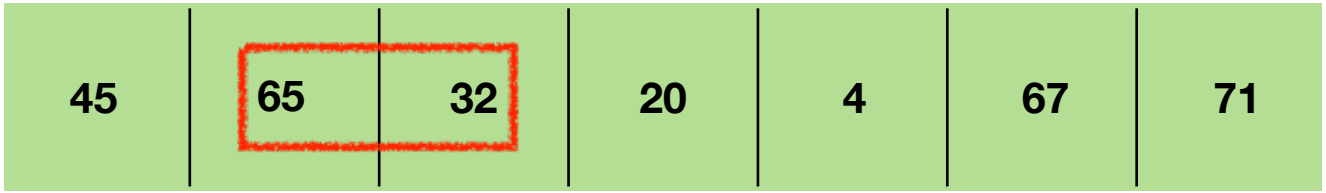
Third Pass:



65>45

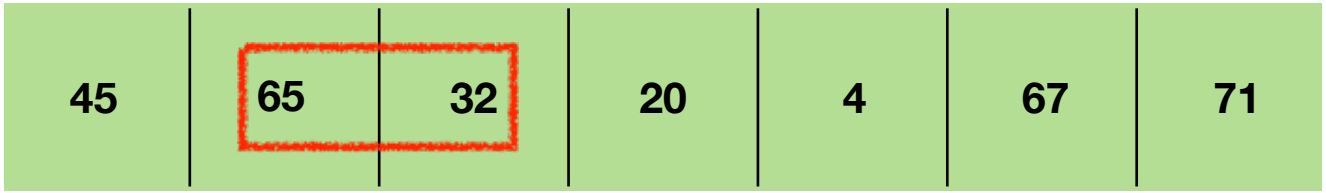
Bubble Sort - Worst Case (cont.)

Third Pass:



Bubble Sort - Worst Case (cont.)

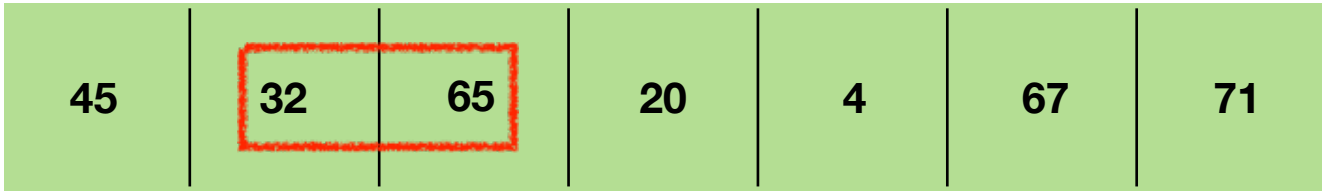
Third Pass:



65>32

Bubble Sort - Worst Case (cont.)

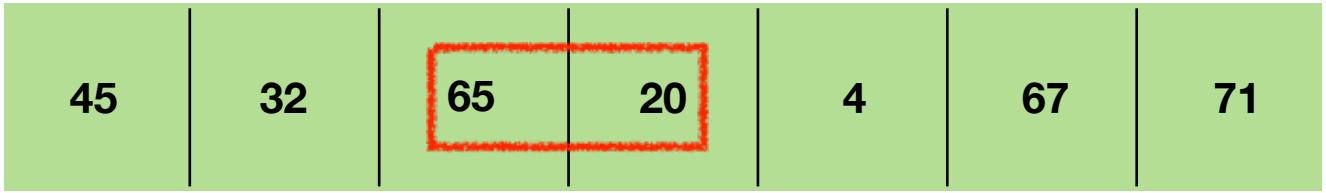
Third Pass:



65>32

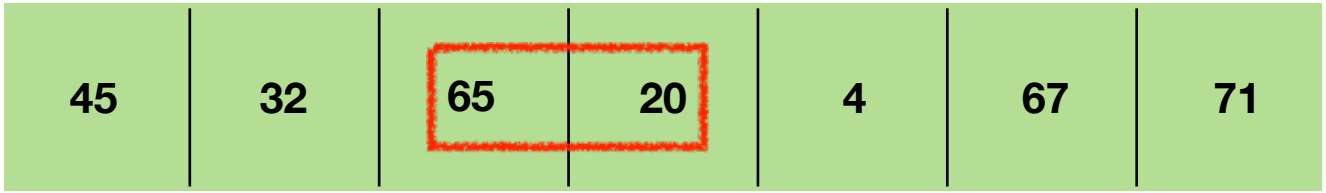
Bubble Sort - Worst Case (cont.)

Third Pass:



Bubble Sort - Worst Case (cont.)

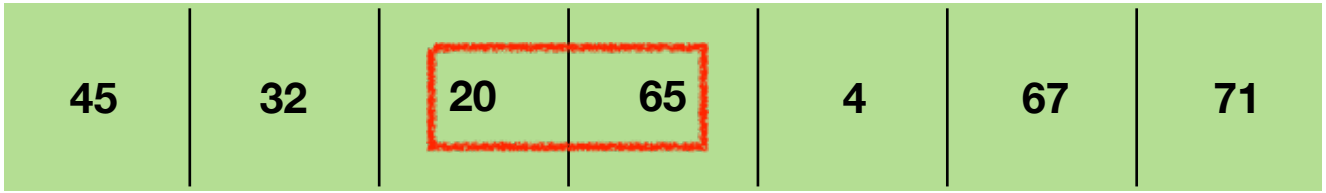
Third Pass:



65>20

Bubble Sort - Worst Case (cont.)

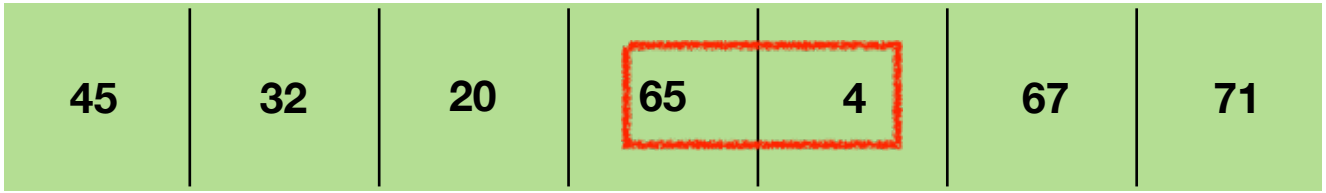
Third Pass:



65>20

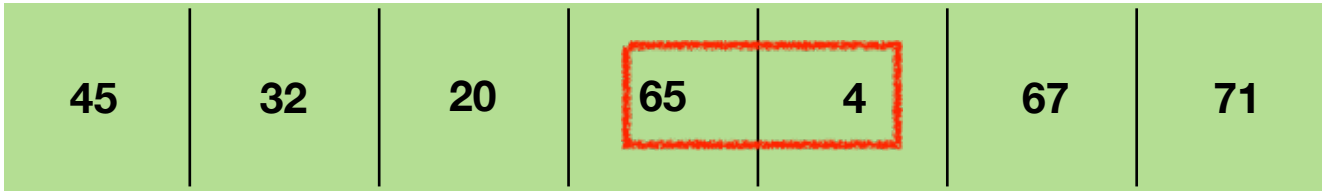
Bubble Sort - Worst Case (cont.)

Third Pass:



Bubble Sort - Worst Case (cont.)

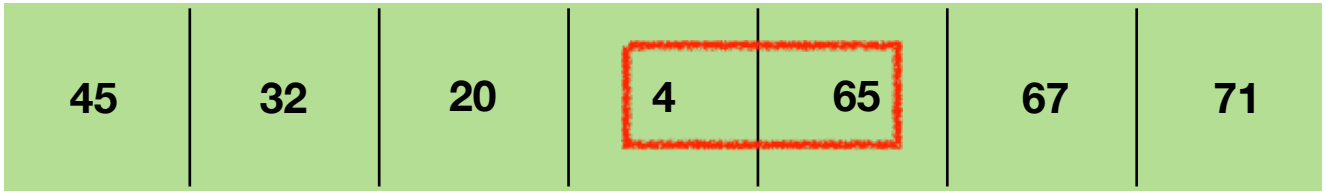
Third Pass:



65>4

Bubble Sort - Worst Case (cont.)

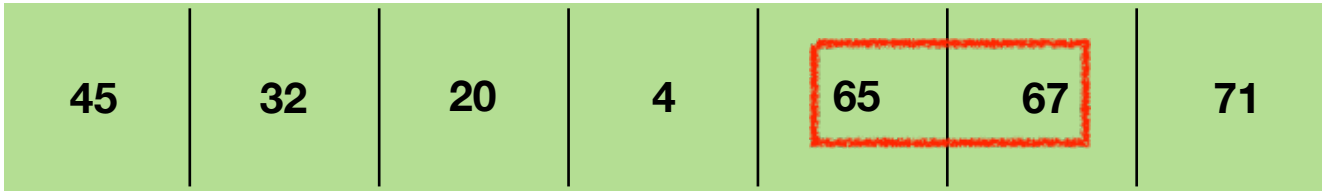
Third Pass:



65>4

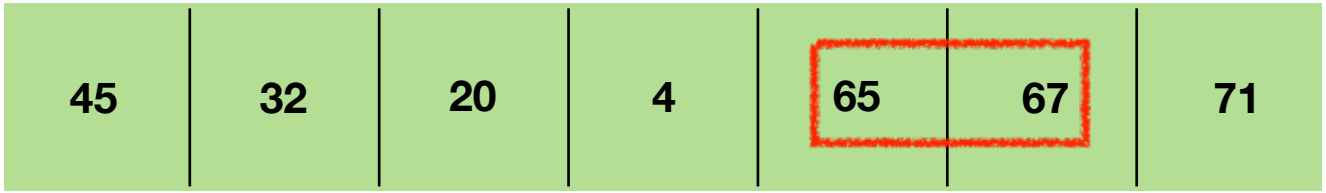
Bubble Sort - Worst Case (cont.)

Third Pass:



Bubble Sort - Worst Case (cont.)

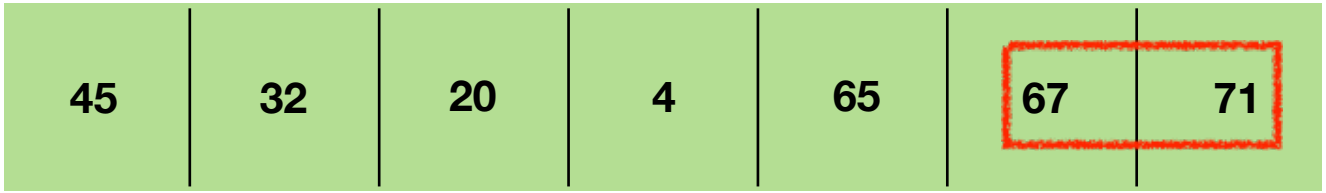
Third Pass:



65<67

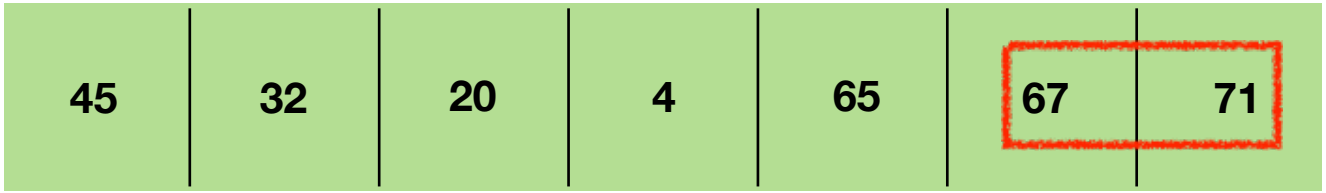
Bubble Sort - Worst Case (cont.)

Third Pass:



Bubble Sort - Worst Case (cont.)

Third Pass:



67<71

Bubble Sort - Worst Case (cont.)

Third Pass:

45	32	20	4	65	67	71

Bubble Sort - Worst Case (cont.)

Third Pass:

45	32	20	4	65	67	71

Swaps occurred in the third pass —> Next Pass

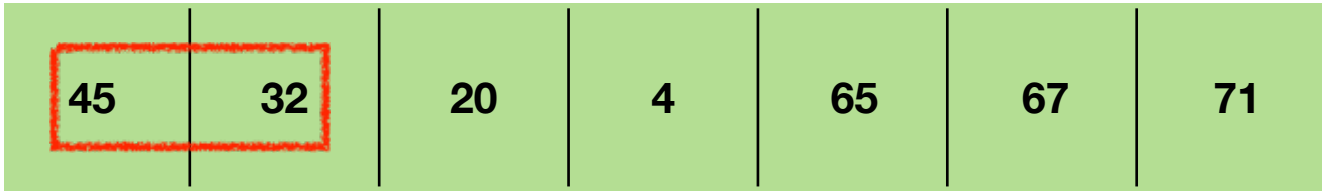
Bubble Sort - Worst Case (cont.)

Fourth Pass:

45	32	20	4	65	67	71

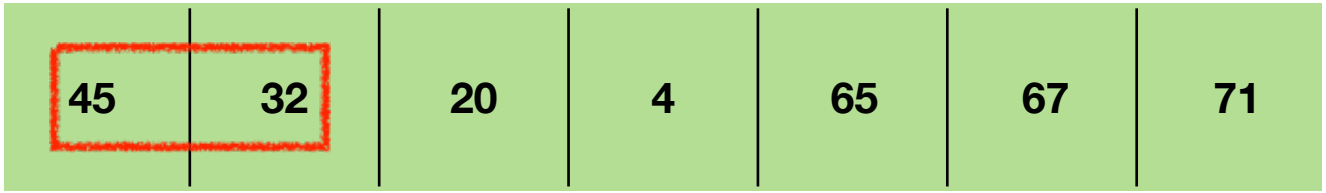
Bubble Sort - Worst Case (cont.)

Fourth Pass:



Bubble Sort - Worst Case (cont.)

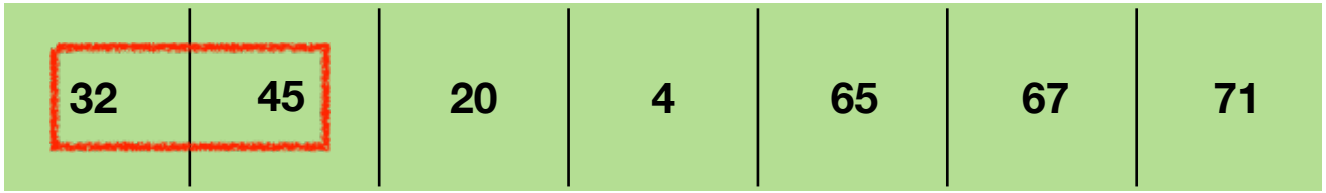
Fourth Pass:



45>32

Bubble Sort - Worst Case (cont.)

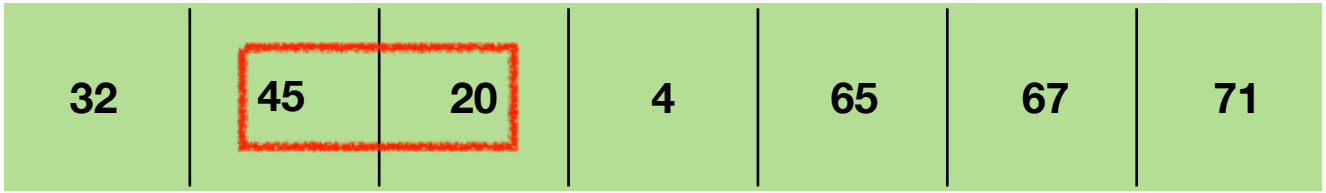
Fourth Pass:



45>32

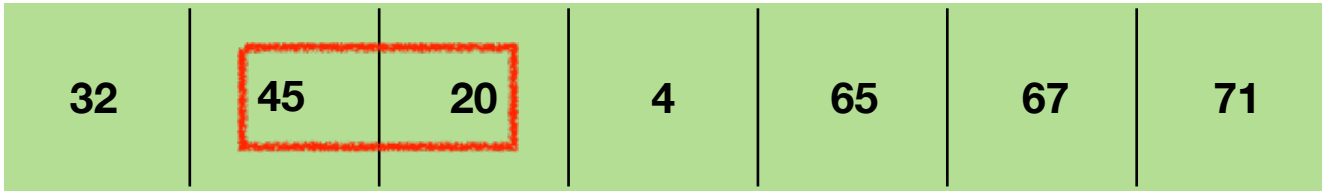
Bubble Sort - Worst Case (cont.)

Fourth Pass:



Bubble Sort - Worst Case (cont.)

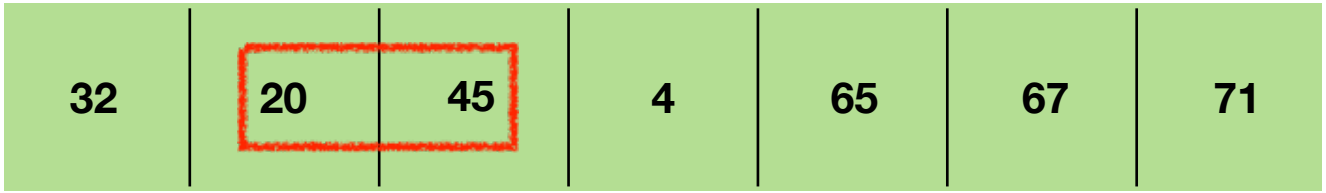
Fourth Pass:



45>20

Bubble Sort - Worst Case (cont.)

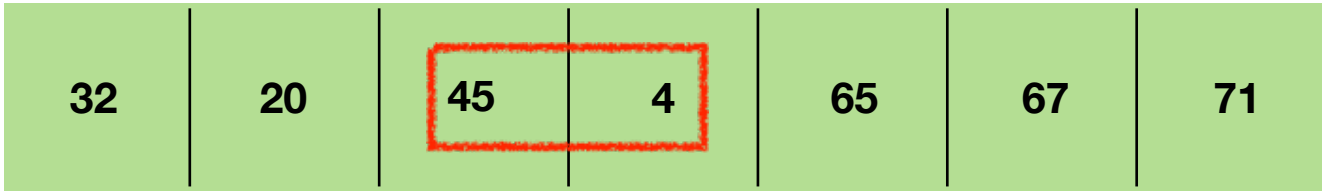
Fourth Pass:



45>20

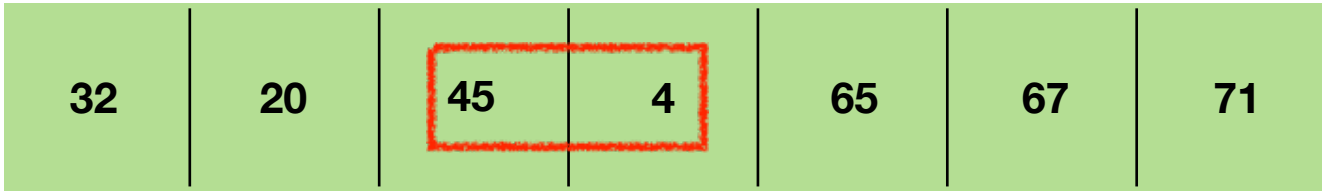
Bubble Sort - Worst Case (cont.)

Fourth Pass:



Bubble Sort - Worst Case (cont.)

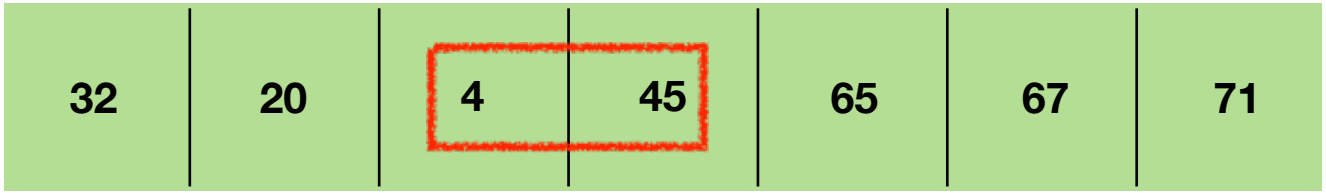
Fourth Pass:



45>4

Bubble Sort - Worst Case (cont.)

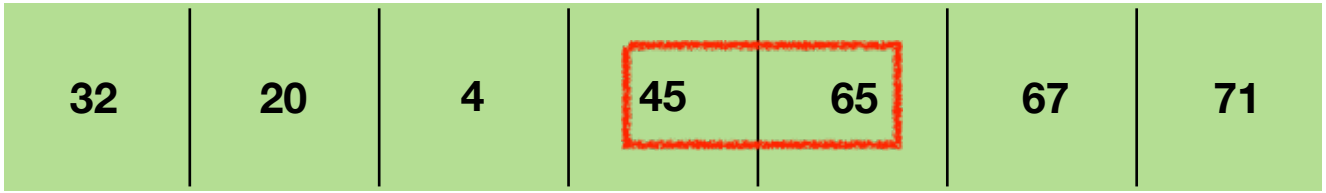
Fourth Pass:



45>4

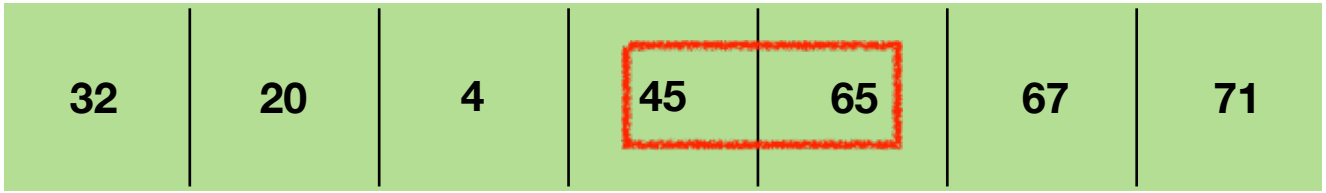
Bubble Sort - Worst Case (cont.)

Fourth Pass:



Bubble Sort - Worst Case (cont.)

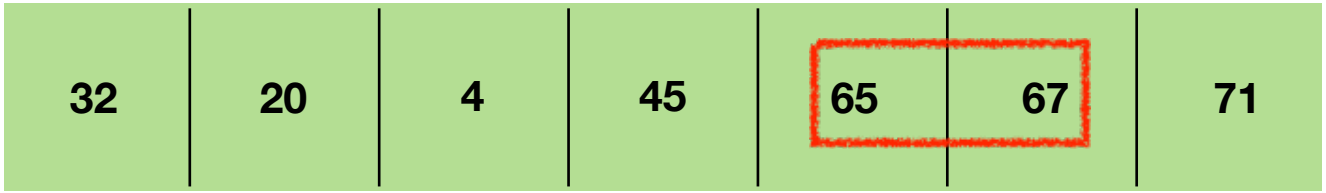
Fourth Pass:



45<65

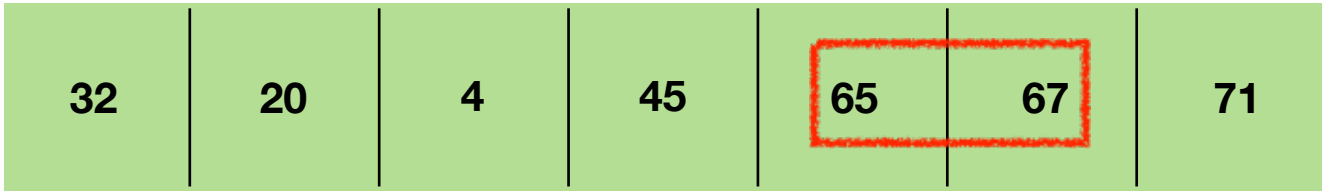
Bubble Sort - Worst Case (cont.)

Fourth Pass:



Bubble Sort - Worst Case (cont.)

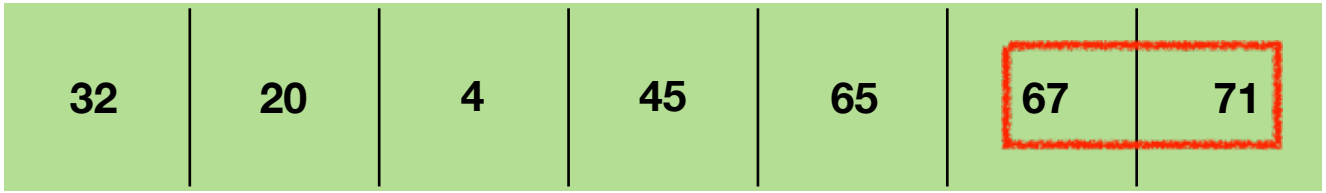
Fourth Pass:



65<67

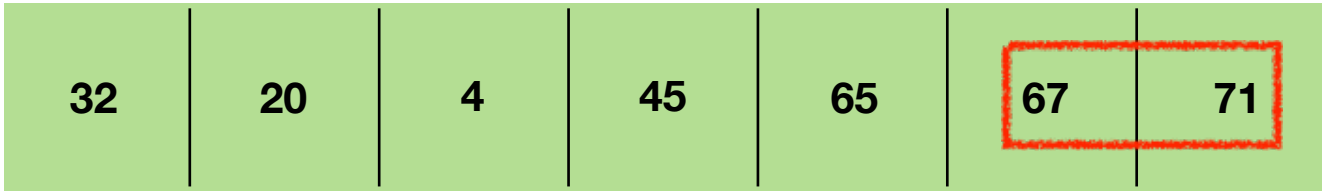
Bubble Sort - Worst Case (cont.)

Fourth Pass:



Bubble Sort - Worst Case (cont.)

Fourth Pass:



67<71

Bubble Sort - Worst Case (cont.)

Fourth Pass:

32	20	4	45	65	67	71

Bubble Sort - Worst Case (cont.)

Fourth Pass:

32	20	4	45	65	67	71

Swaps occurred in the fourth pass —> Next Pass

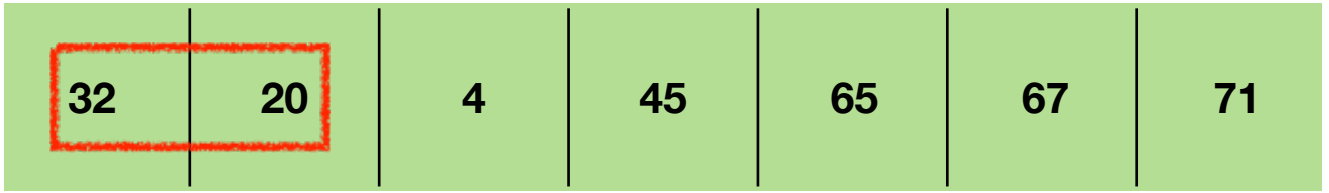
Bubble Sort - Worst Case (cont.)

Fifth Pass:

32	20	4	45	65	67	71

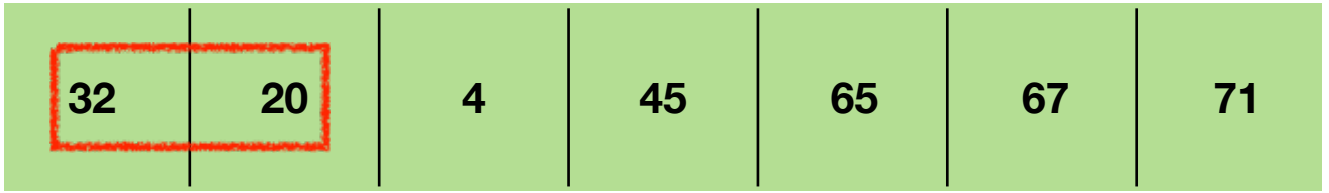
Bubble Sort - Worst Case (cont.)

Fifth Pass:



Bubble Sort - Worst Case (cont.)

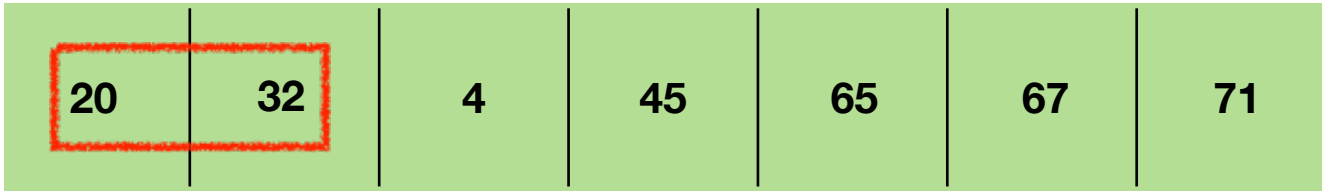
Fifth Pass:



32>20

Bubble Sort - Worst Case (cont.)

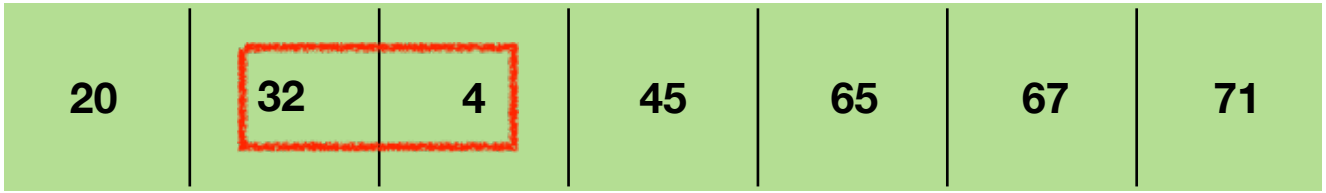
Fifth Pass:



32>20

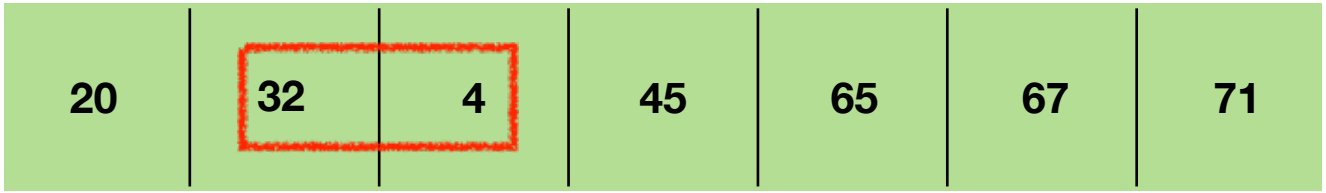
Bubble Sort - Worst Case (cont.)

Fifth Pass:



Bubble Sort - Worst Case (cont.)

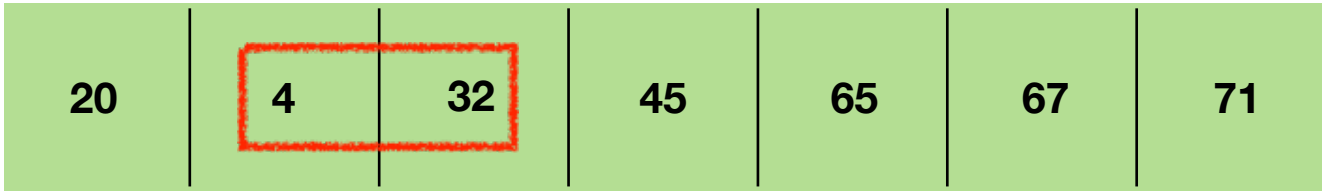
Fifth Pass:



32>4

Bubble Sort - Worst Case (cont.)

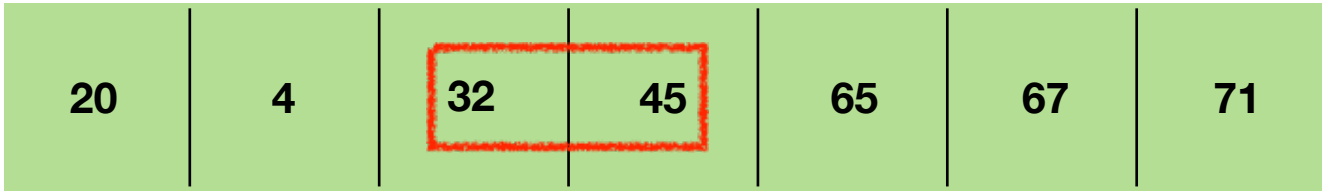
Fifth Pass:



32>4

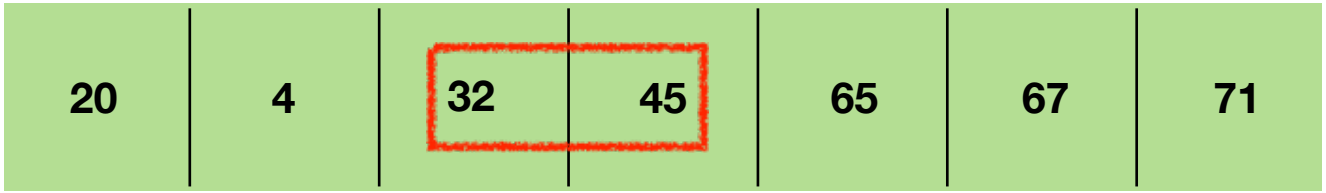
Bubble Sort - Worst Case (cont.)

Fifth Pass:



Bubble Sort - Worst Case (cont.)

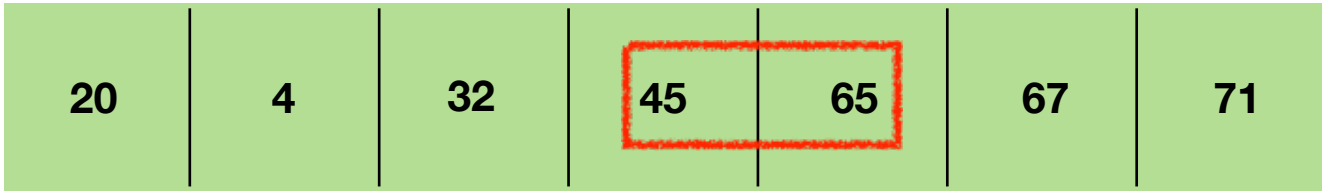
Fifth Pass:



32<45

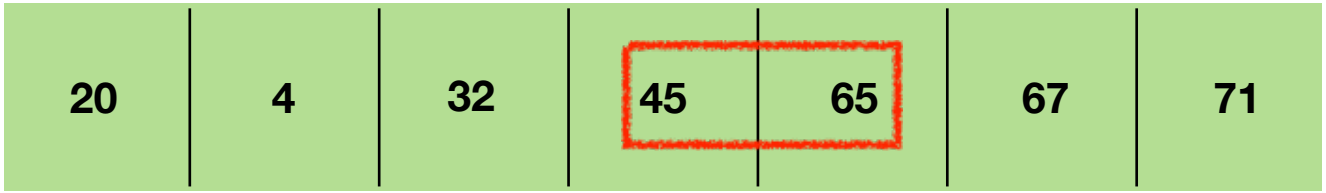
Bubble Sort - Worst Case (cont.)

Fifth Pass:



Bubble Sort - Worst Case (cont.)

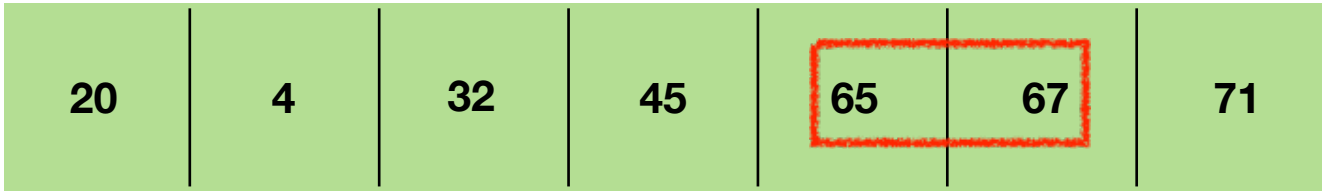
Fifth Pass:



45<65

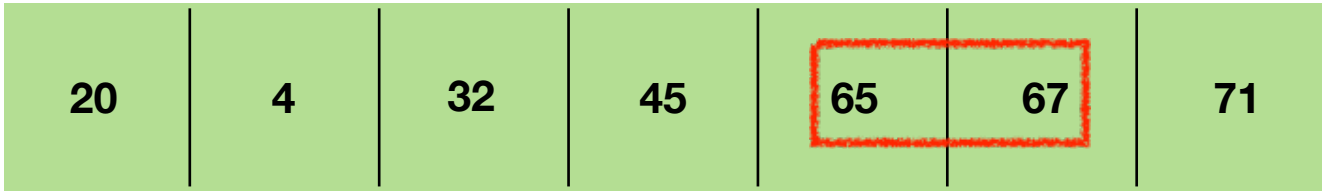
Bubble Sort - Worst Case (cont.)

Fifth Pass:



Bubble Sort - Worst Case (cont.)

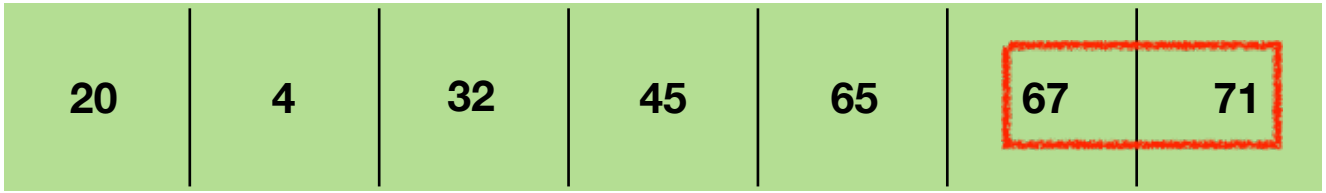
Fifth Pass:



65<67

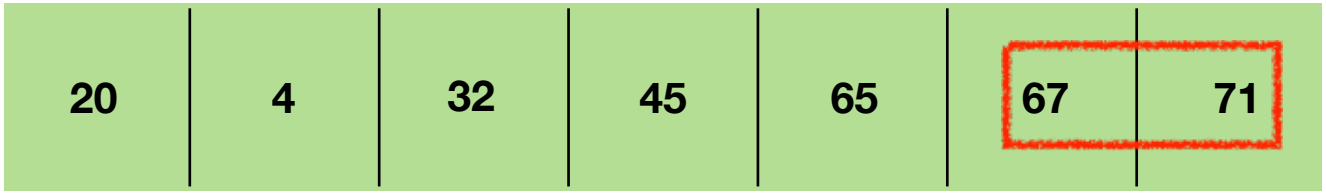
Bubble Sort - Worst Case (cont.)

Fifth Pass:



Bubble Sort - Worst Case (cont.)

Fifth Pass:



67<71

Bubble Sort - Worst Case (cont.)

Fifth Pass:

20	4	32	45	65	67	71

Bubble Sort - Worst Case (cont.)

Fifth Pass:

20	4	32	45	65	67	71

Swaps occurred in the fifth pass —> Next Pass

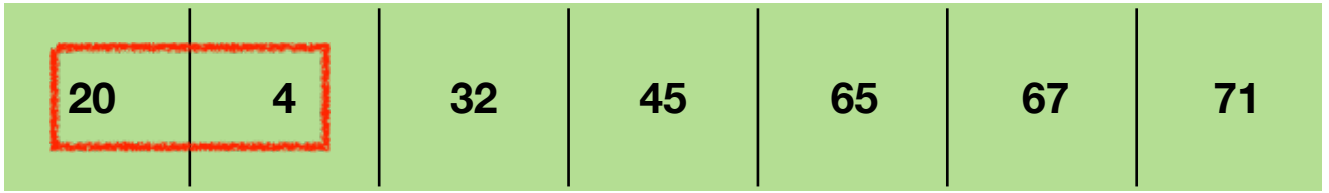
Bubble Sort - Worst Case (cont.)

Sixth Pass:

20	4	32	45	65	67	71

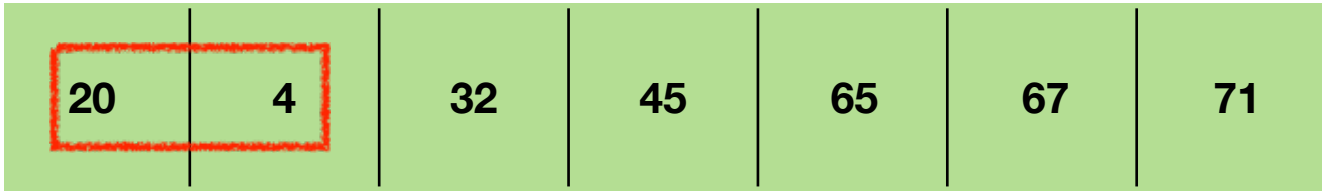
Bubble Sort - Worst Case (cont.)

Sixth Pass:



Bubble Sort - Worst Case (cont.)

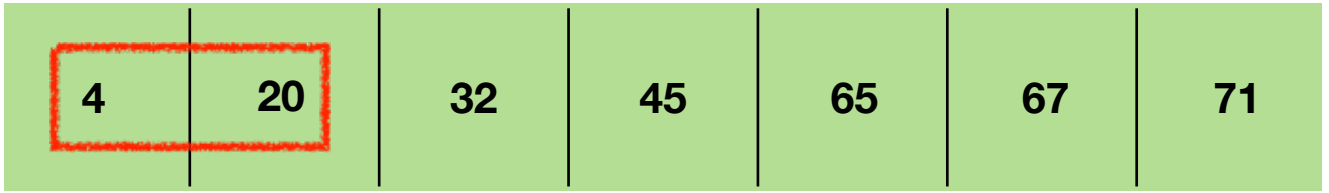
Sixth Pass:



20>4

Bubble Sort - Worst Case (cont.)

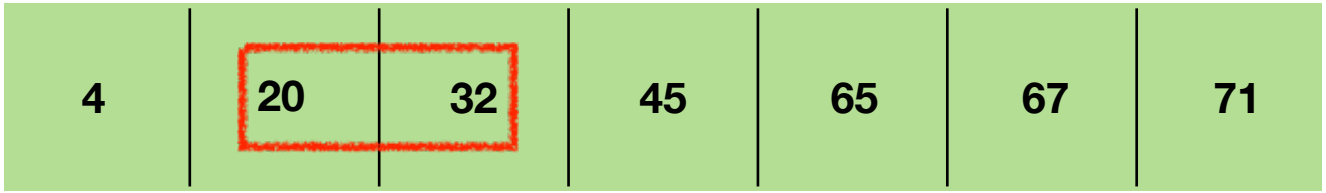
Sixth Pass:



20>4

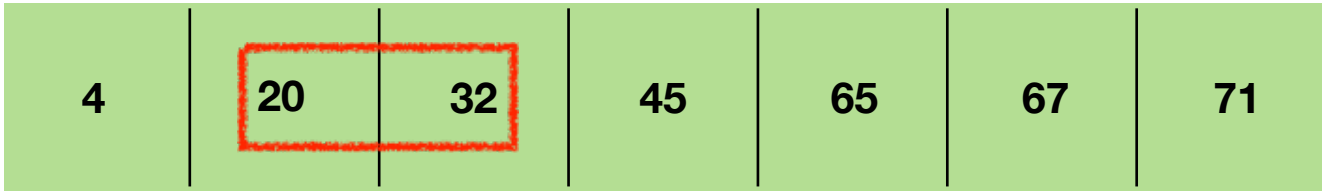
Bubble Sort - Worst Case (cont.)

Sixth Pass:



Bubble Sort - Worst Case (cont.)

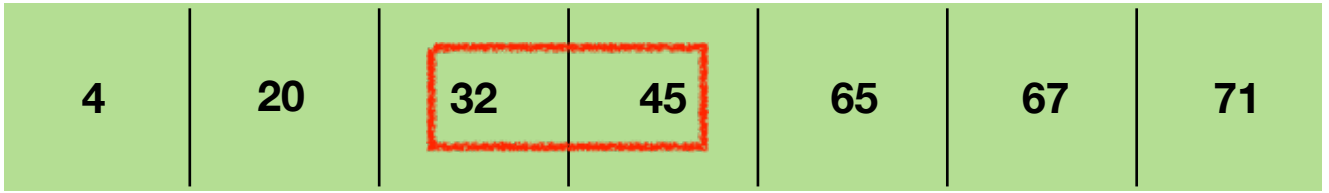
Sixth Pass:



20<32

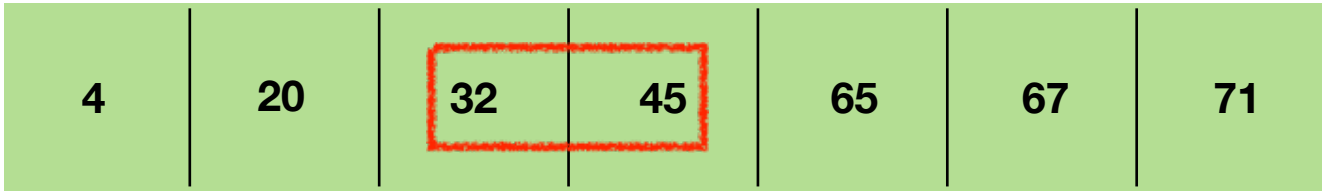
Bubble Sort - Worst Case (cont.)

Sixth Pass:



Bubble Sort - Worst Case (cont.)

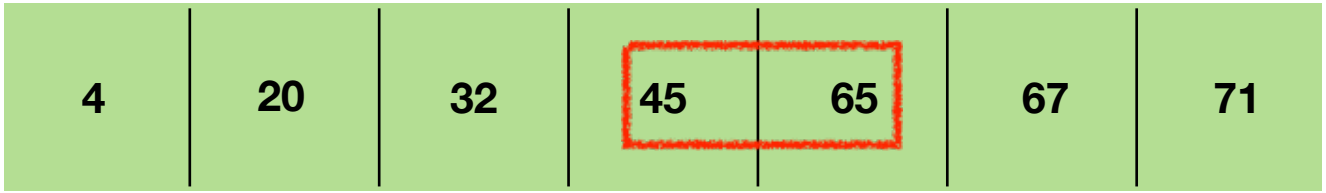
Sixth Pass:



32<45

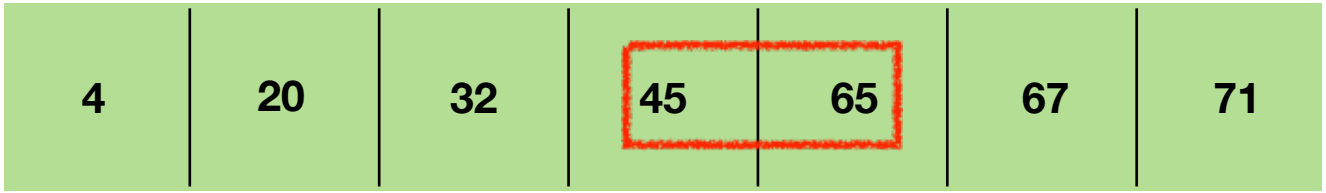
Bubble Sort - Worst Case (cont.)

Sixth Pass:



Bubble Sort - Worst Case (cont.)

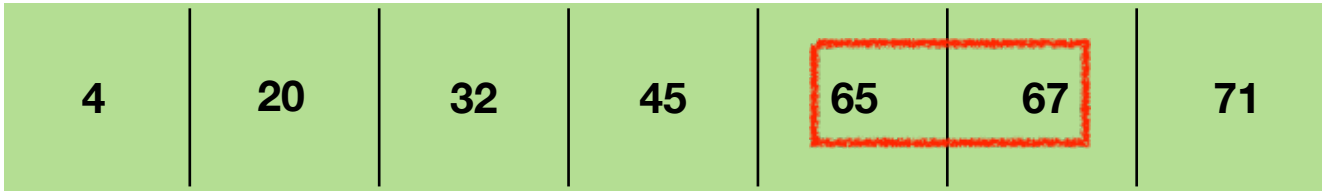
Sixth Pass:



45<65

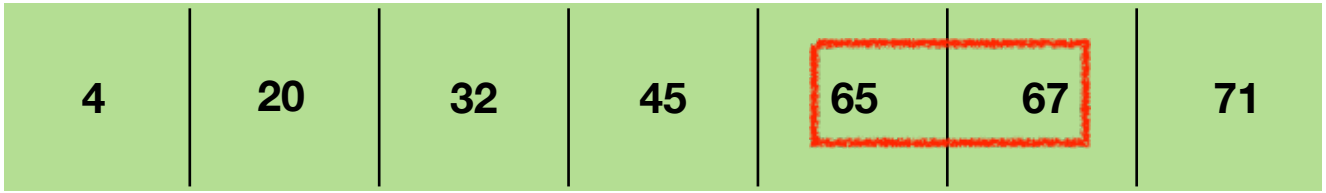
Bubble Sort - Worst Case (cont.)

Sixth Pass:



Bubble Sort - Worst Case (cont.)

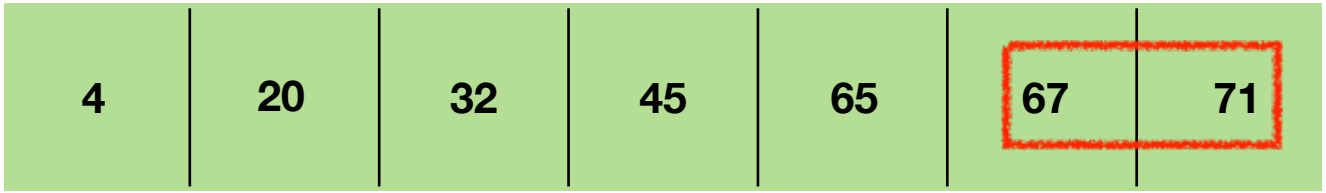
Sixth Pass:



65<67

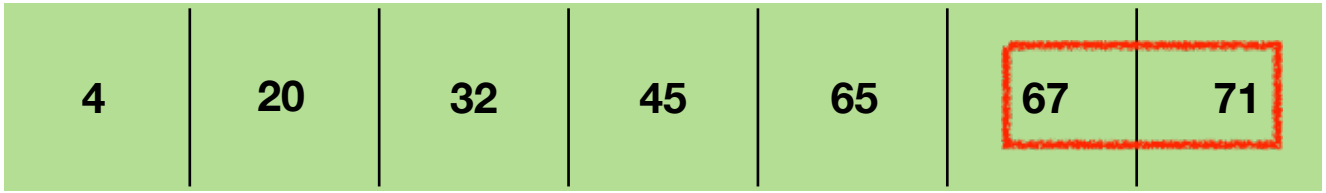
Bubble Sort - Worst Case (cont.)

Sixth Pass:



Bubble Sort - Worst Case (cont.)

Sixth Pass:



67<71

Bubble Sort - Worst Case (cont.)

Sixth Pass:

4	20	32	45	65	67	71

Bubble Sort - Worst Case (cont.)

Sixth Pass:

4	20	32	45	65	67	71

Swaps occurred in the sixth pass —> Next Pass

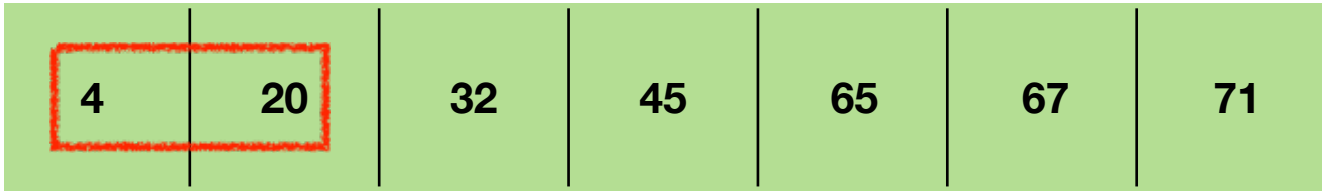
Bubble Sort - Worst Case (cont.)

Seventh Pass:

4	20	32	45	65	67	71

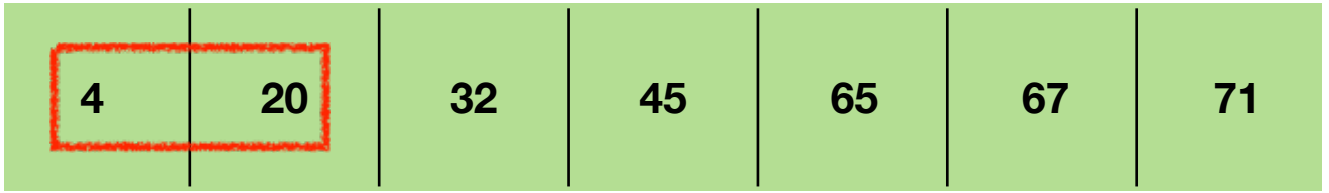
Bubble Sort - Worst Case (cont.)

Seventh Pass:



Bubble Sort - Worst Case (cont.)

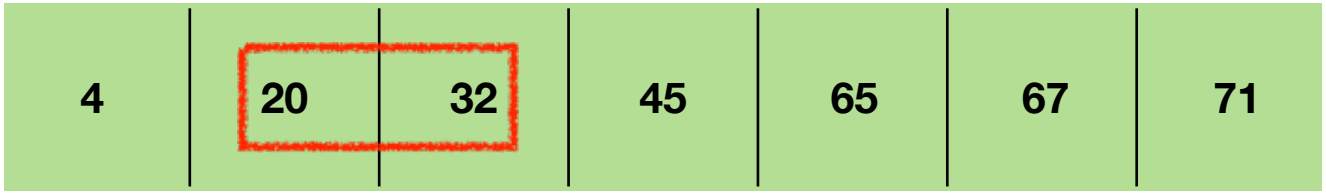
Seventh Pass:



4<20

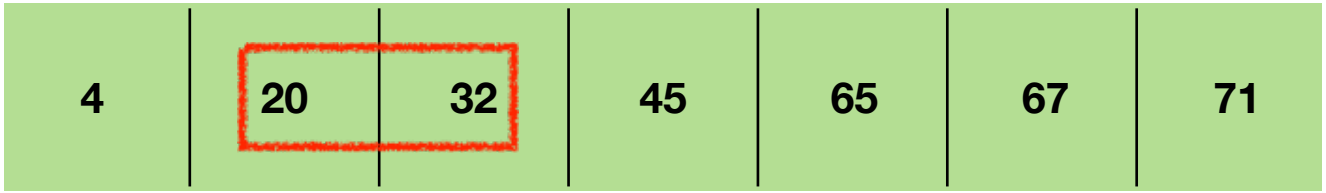
Bubble Sort - Worst Case (cont.)

Seventh Pass:



Bubble Sort - Worst Case (cont.)

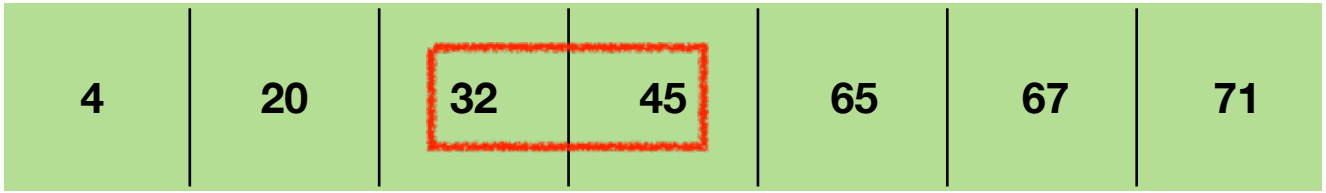
Seventh Pass:



20<32

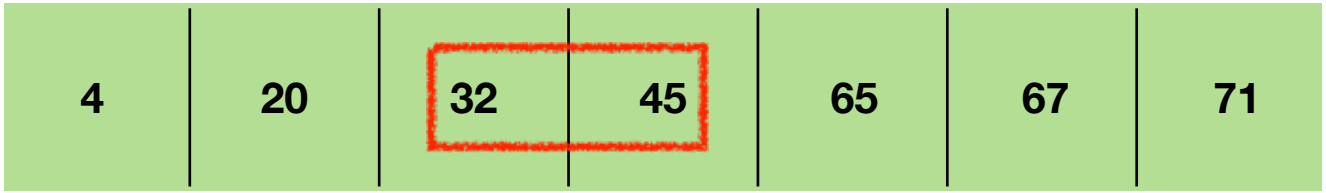
Bubble Sort - Worst Case (cont.)

Seventh Pass:



Bubble Sort - Worst Case (cont.)

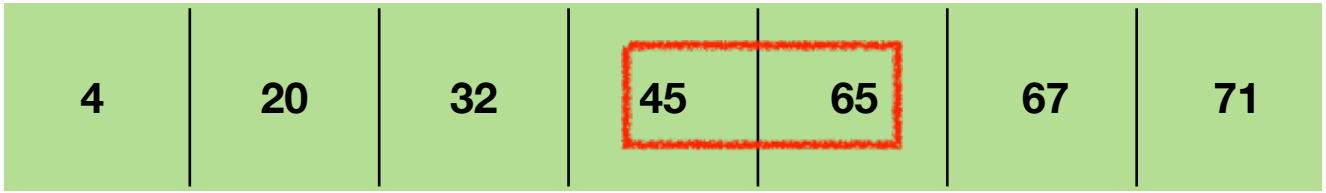
Seventh Pass:



32<45

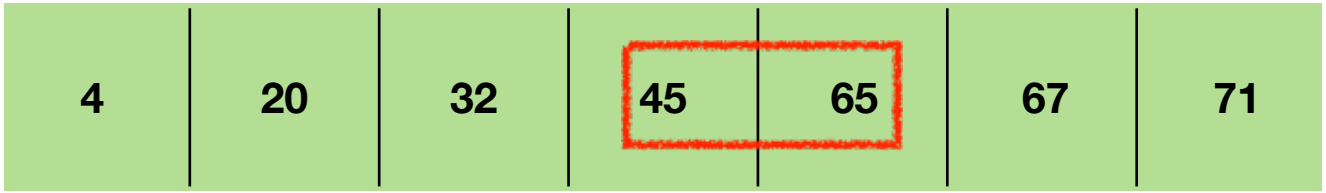
Bubble Sort - Worst Case (cont.)

Seventh Pass:



Bubble Sort - Worst Case (cont.)

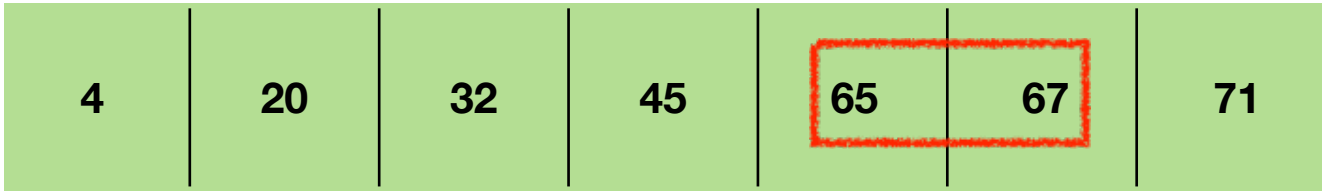
Seventh Pass:



45<65

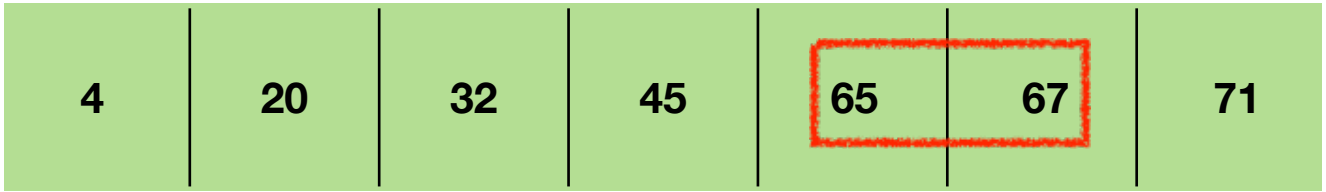
Bubble Sort - Worst Case (cont.)

Seventh Pass:



Bubble Sort - Worst Case (cont.)

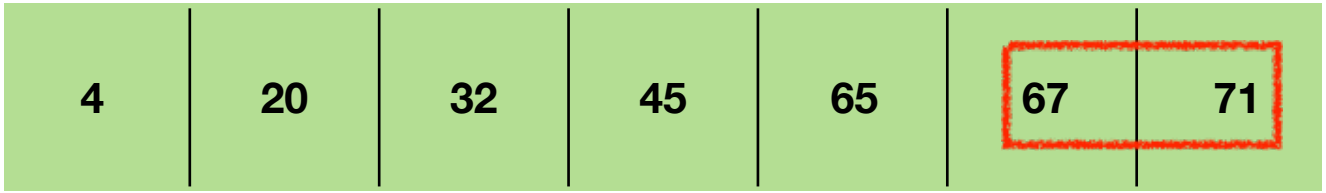
Seventh Pass:



65<67

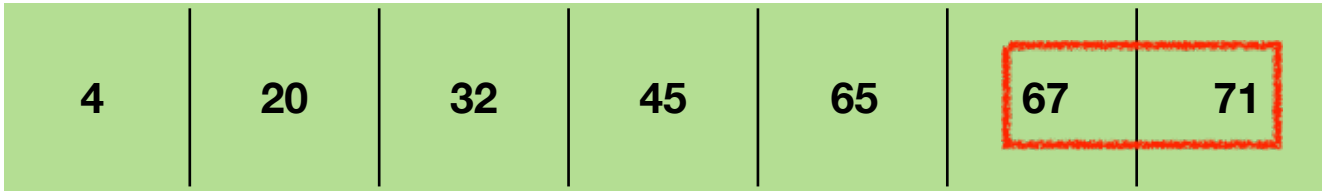
Bubble Sort - Worst Case (cont.)

Seventh Pass:



Bubble Sort - Worst Case (cont.)

Seventh Pass:



67<71

Bubble Sort - Worst Case (cont.)

Seventh Pass:

4	20	32	45	65	67	71

Bubble Sort - Worst Case (cont.)

Seventh Pass:

4	20	32	45	65	67	71

No swaps occurred in the seventh pass —> Done

Running Time T(n) in Worst Case

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

Running Time T(n) in Worst Case

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

$T(n) = 2* \#ofPass + 4(n-1)* \#ofPass + 1 = 4n^2 - 2n + 1$

Running Time T(n) in Worst Case

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

#ofPass = n

$T(n) = 2* \#ofPass + 4(n-1)* \#ofPass + 1 = 4n^2 - 2n + 1$

Worst-Case Time Complexity T(n)

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1     <=(n-1) * #ofPass
               flag = true                            1     <=(n-1) * #ofPass
    while(flag)                                       1        #ofPass
}
```

$$T(n) = 4n^2 - 2n + 1 = O(n^2)$$

Worst-Case Time Complexity T(n)

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1       #ofPass
        for i = 1 to n-1                              1    (n-1) * #ofPass
            if A[i] > A[i+1]                          1    (n-1) * #ofPass
               swap(A[i], A[i+1])                     1    <=(n-1) * #ofPass
               flag = true                            1    <=(n-1) * #ofPass
    while(flag)                                       1       #ofPass
}
```

Two rules:

- Ignore the low-order terms
- Drop the leading constant

$T(n) = 4n^2 - 2n + 1 = O(n^2)$

Bubble Sort — Best Case

A is given in the sorted order.

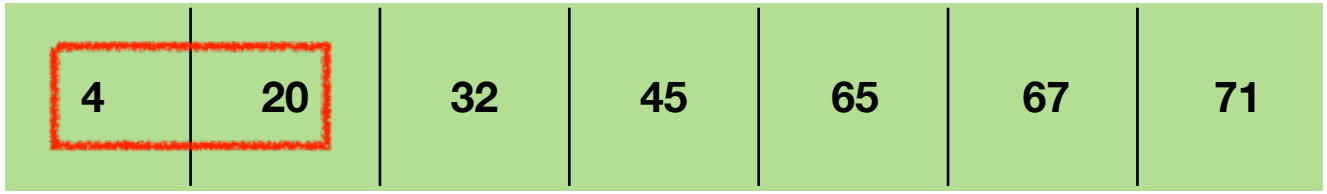
First Pass:

4	20	32	45	65	67	71

Bubble Sort — Best Case

A is given in the sorted order.

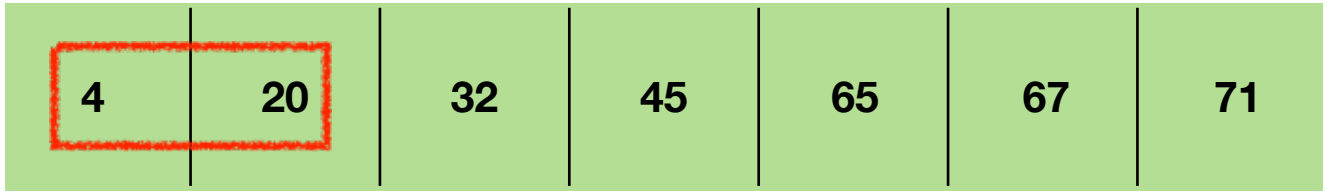
First Pass:



Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

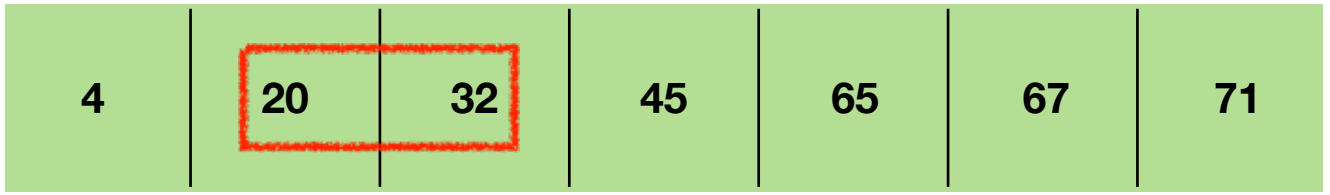


4<20

Bubble Sort — Best Case

A is given in the sorted order.

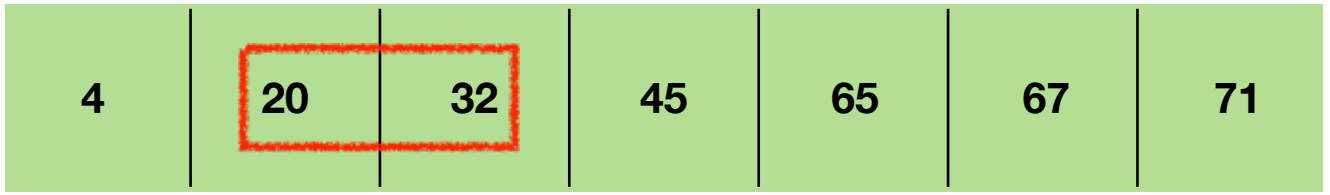
First Pass:



Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

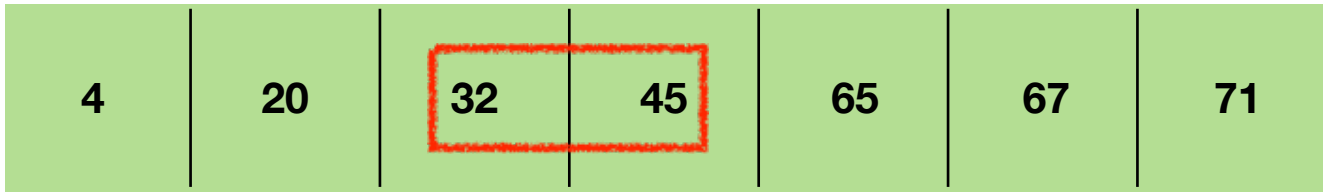


20<32

Bubble Sort — Best Case

A is given in the sorted order.

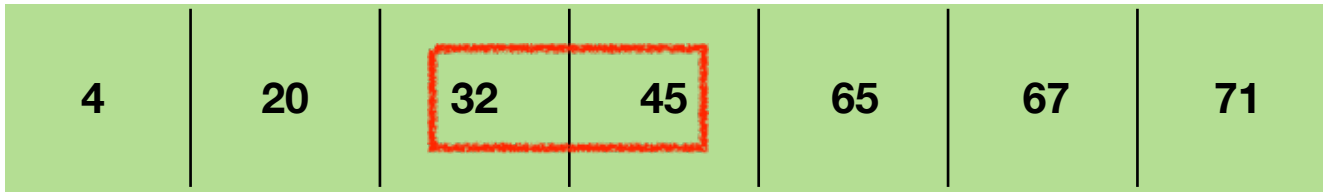
First Pass:



Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

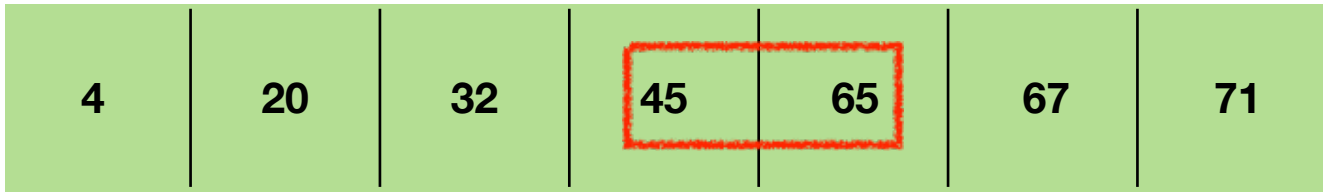


32<45

Bubble Sort — Best Case

A is given in the sorted order.

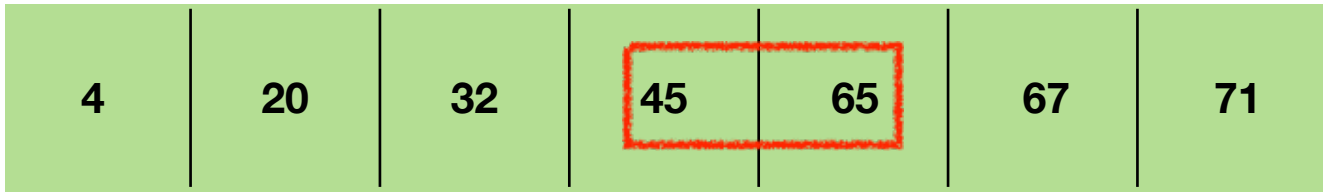
First Pass:



Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

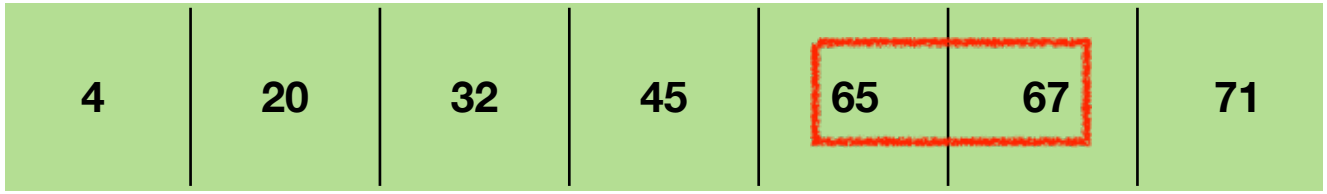


45<65

Bubble Sort — Best Case

A is given in the sorted order.

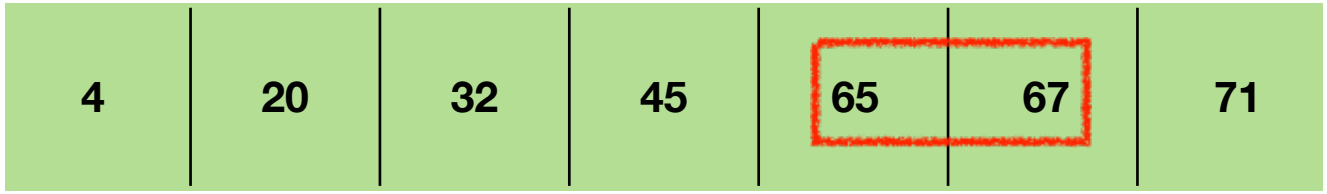
First Pass:



Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

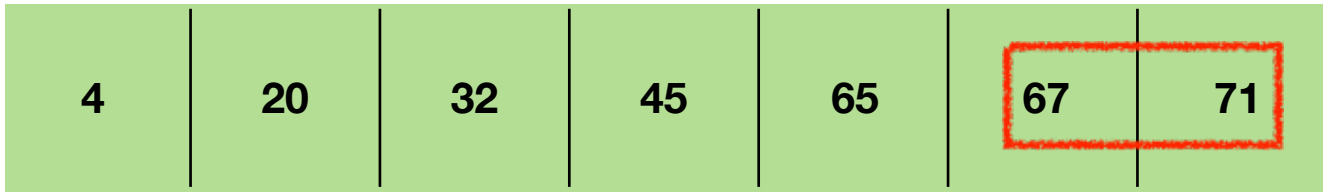


65<67

Bubble Sort — Best Case

A is given in the sorted order.

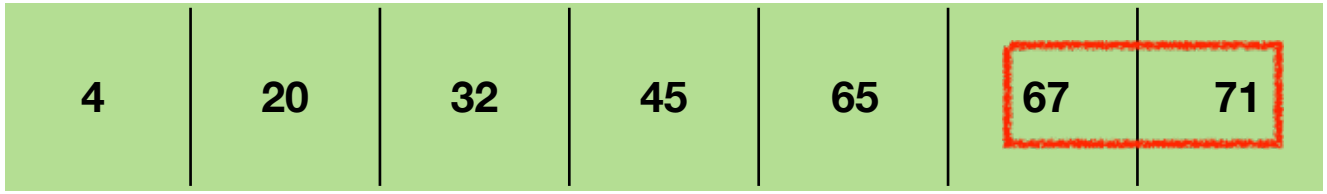
First Pass:



Bubble Sort — Best Case

A is given in the sorted order.

First Pass:



67<71

Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

4	20	32	45	65	67	71

Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

4	20	32	45	65	67	71

No swaps occurred in the first pass —> Done

Bubble Sort — Best Case

A is given in the sorted order.

First Pass:

4	20	32	45	65	67	71

No swaps occurred in the first pass —> Done

#ofPass = 1

Running Time T(n) in Best Case

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1          0
               flag = true                            1          0
    while(flag)                                       1        #ofPass
}
```

Running Time T(n) in Best Case

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1          0
               flag = true                            1          0
    while(flag)                                       1        #ofPass
}
```

#ofPass = 1

Running Time T(n) in Best Case

```
bubbleSort(Array A : list of sortable items, n)
{                                                    cost      times
    flag = false                                      1          1
    do
        flag = false                                  1        #ofPass
        for i = 1 to n-1                              1     (n-1) * #ofPass
            if A[i] > A[i+1]                          1     (n-1) * #ofPass
               swap(A[i], A[i+1])                     1          0
               flag = true                            1          0
    while(flag)                                       1        #ofPass
}
```

#ofPass = 1

$$T(n) = 2* \#ofPass + 2(n-1)* \#ofPass + 1 = 2n + 1 = O(n)$$

For every pass:
Do {
  Flag = false;
for i=1:n-1
  If A[i] >A[i+1]
  {
    swap(A[i], A[i+1])
      {
       temp= A[i+1];
       A[i+1] = A[i];
       A[i] = temp
     }
    Flag = true: //this pass has swap
  }
while(flag)

How many passes?
If every two adjacent element
Are in the increasing order,
NO SWAPS OCCURED IN ONE PASS
BUBBLE SORT ENDS.

For an algorithm to solve a problem:
The correctness.
Every pass check and correct the order of every two adjacent number.
The whole list is in in===== the correctness of
The time complexity of an algorithm