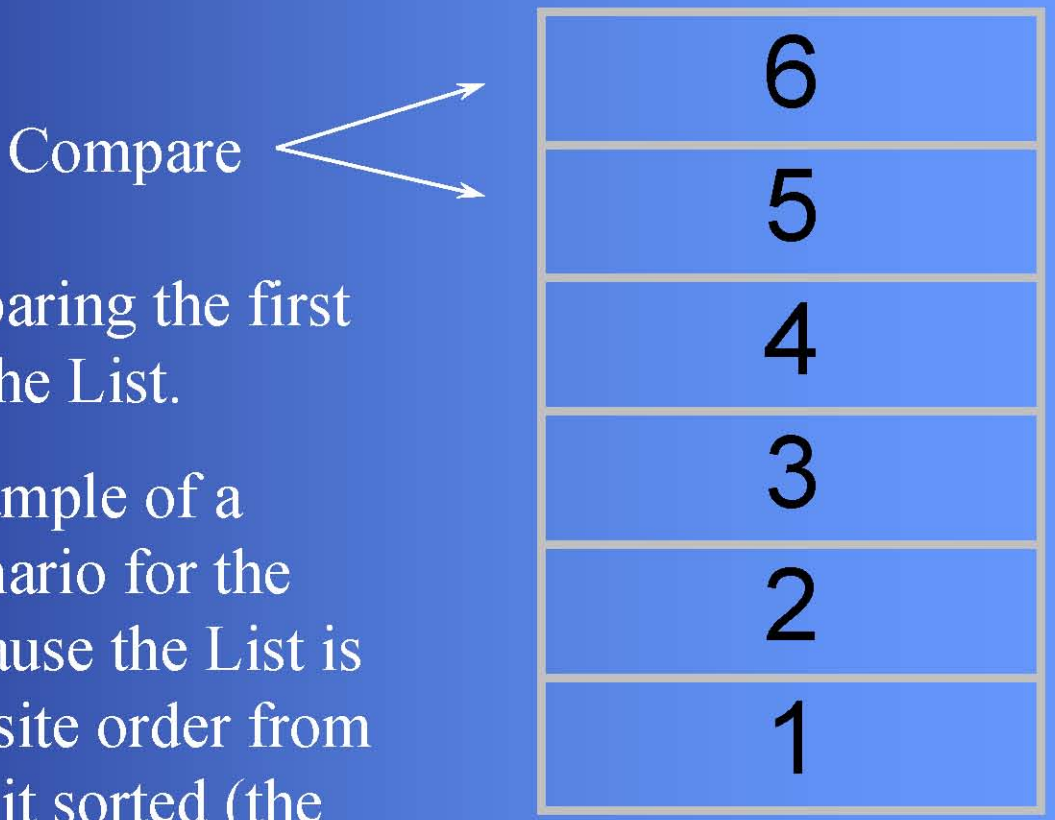
The Bubble Sort

Mr. Dave Clausen
La Cañada High School

The Bubble Sort Algorithm

The Bubble Sort compares adjacent elements in a list, and "swaps" them if they are not in order. Each pair of adjacent elements is compared and swapped until the smallest element "bubbles" to the top. Repeat this process each time stopping one indexed element less, until you compare only the first (or last) two elements in the list. The largest element will have "sunk" to the bottom.

A Bubble Sort Example

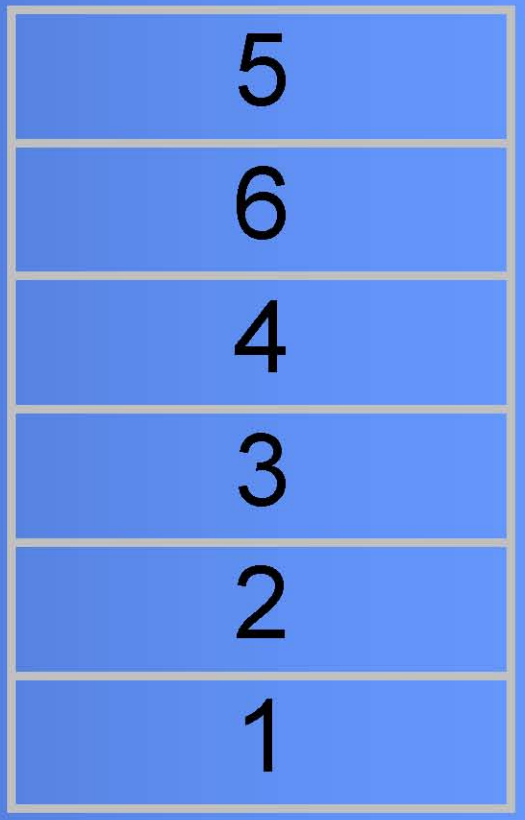


We start by comparing the first two elements in the List.

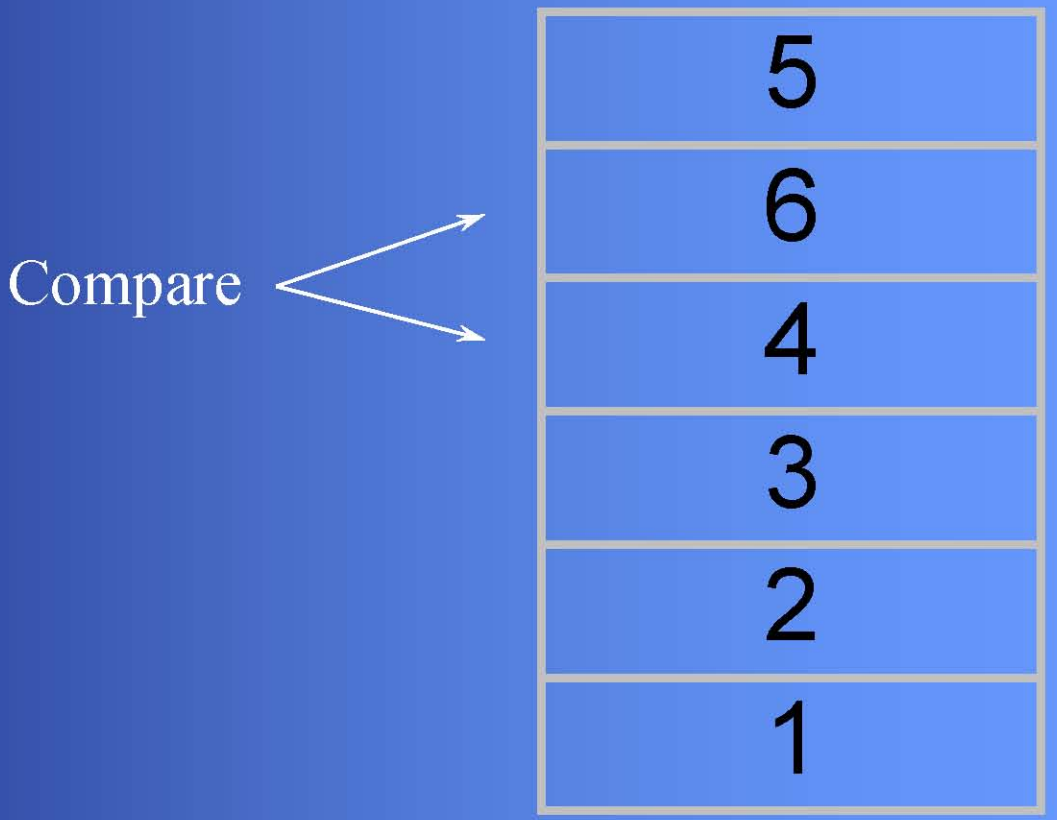
This list is an example of a "worst case" scenario for the Bubble Sort, because the List is in the exact opposite order from the way we want it sorted (the computer program does not know that it is sorted at all).

A Bubble Sort Example

Swap

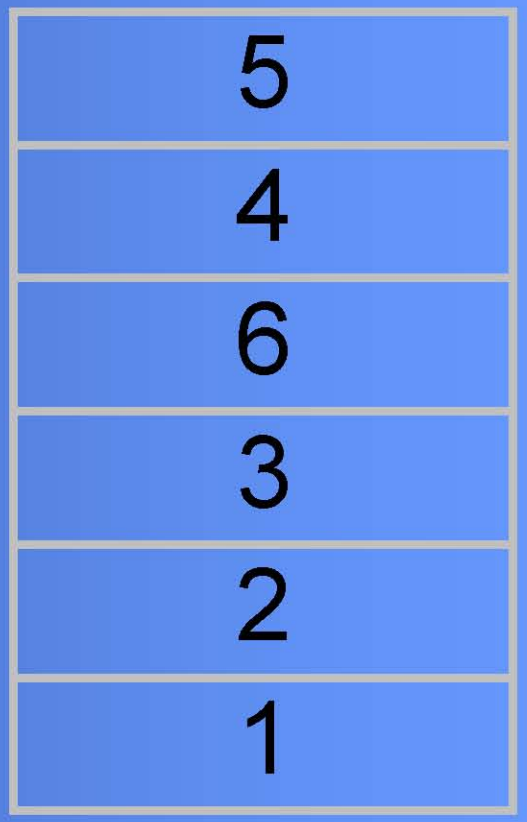


A Bubble Sort Example

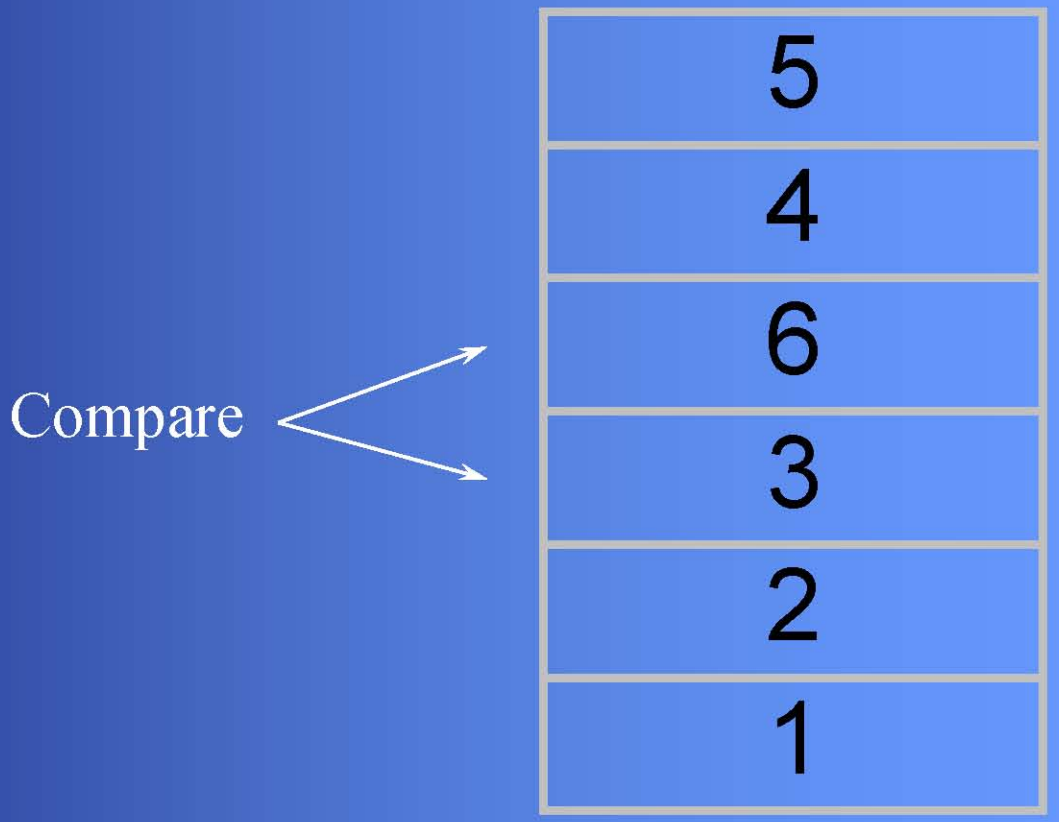


A Bubble Sort Example

Swap

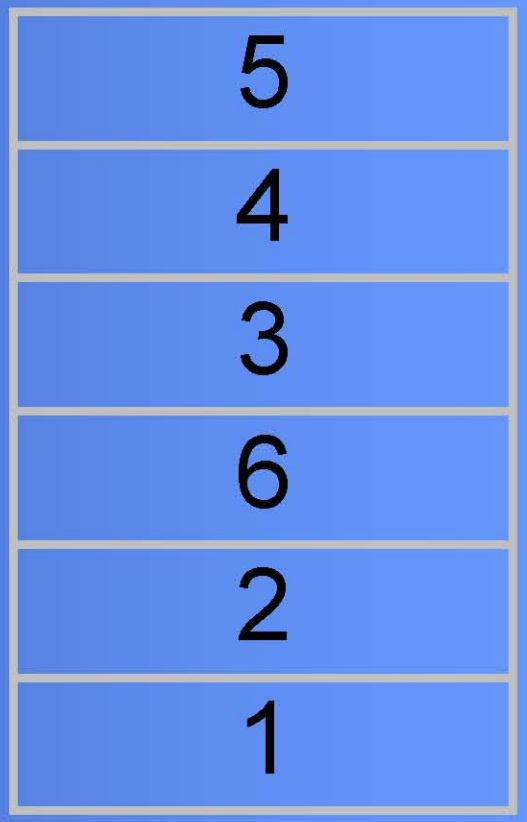


A Bubble Sort Example

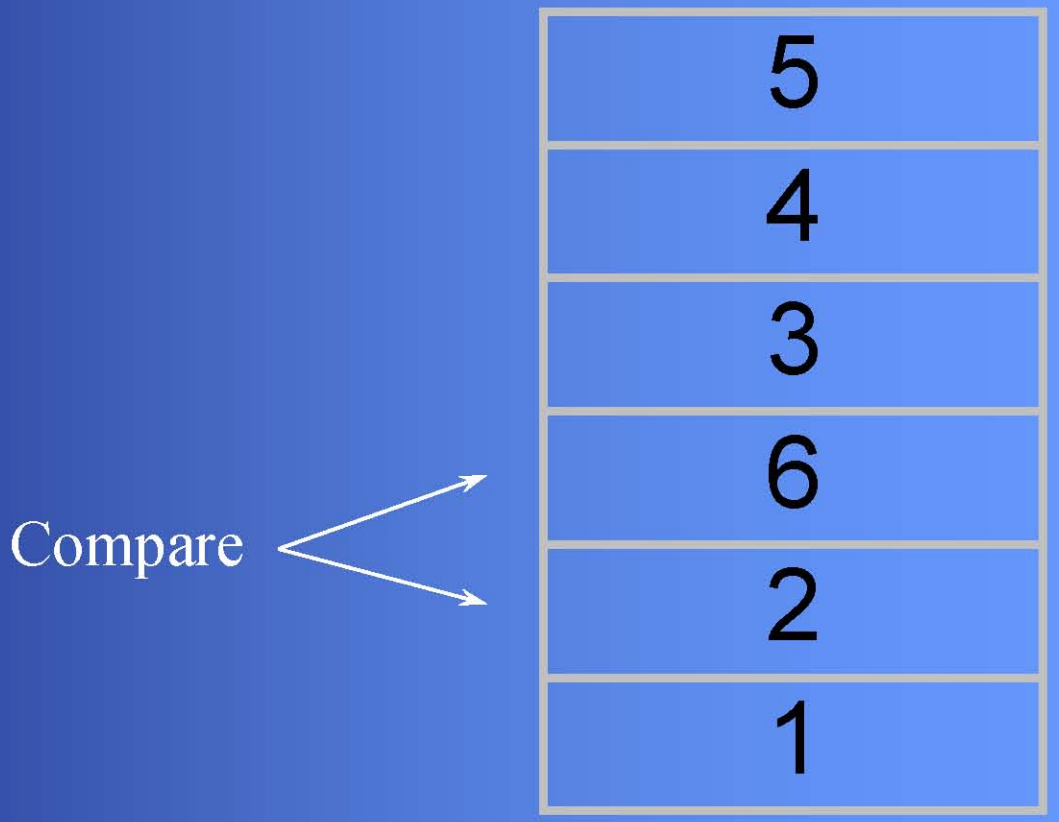


A Bubble Sort Example

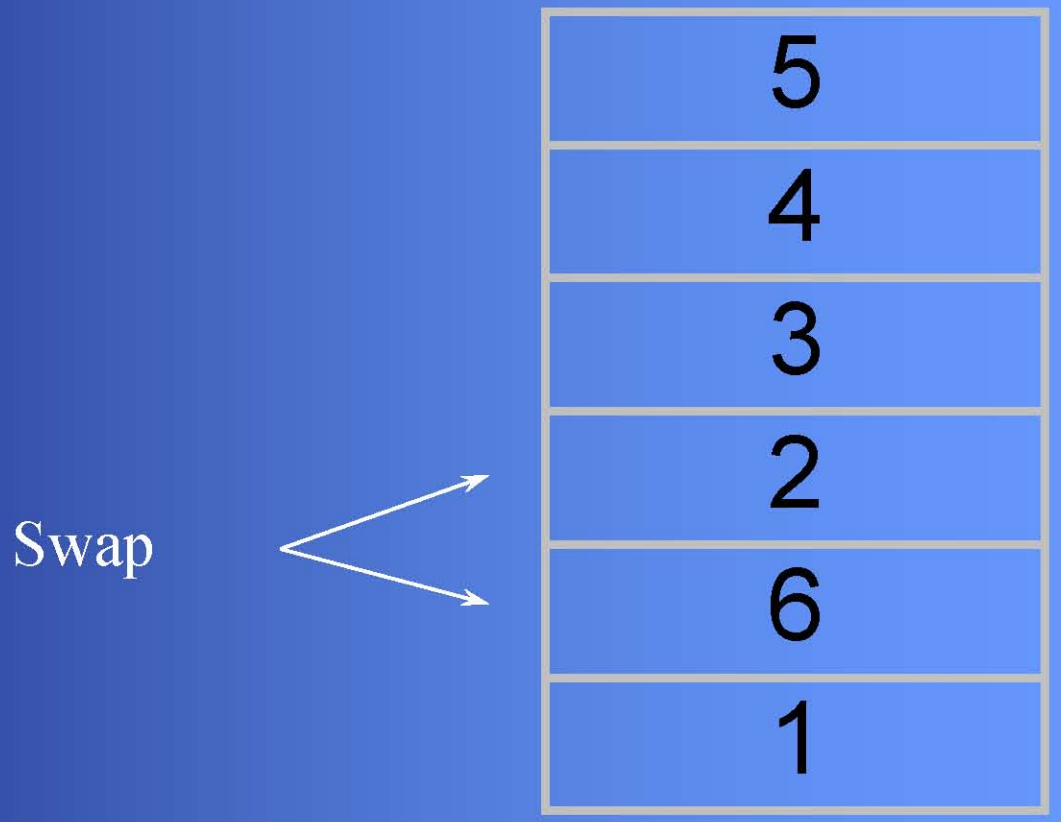
Swap



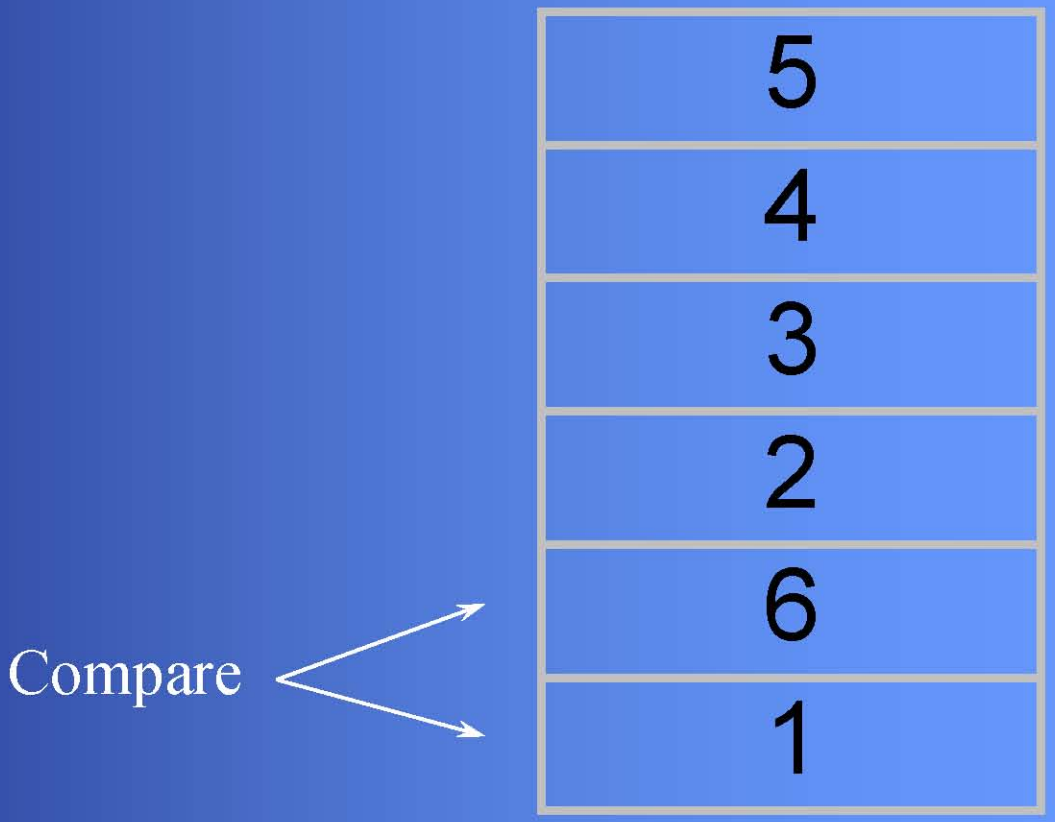
A Bubble Sort Example



A Bubble Sort Example



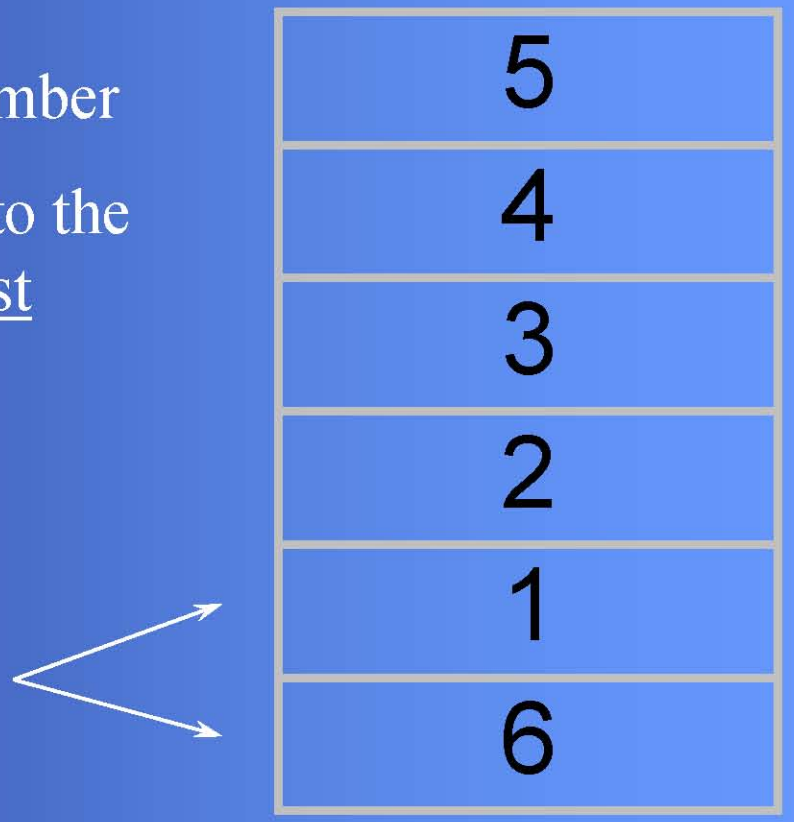
A Bubble Sort Example



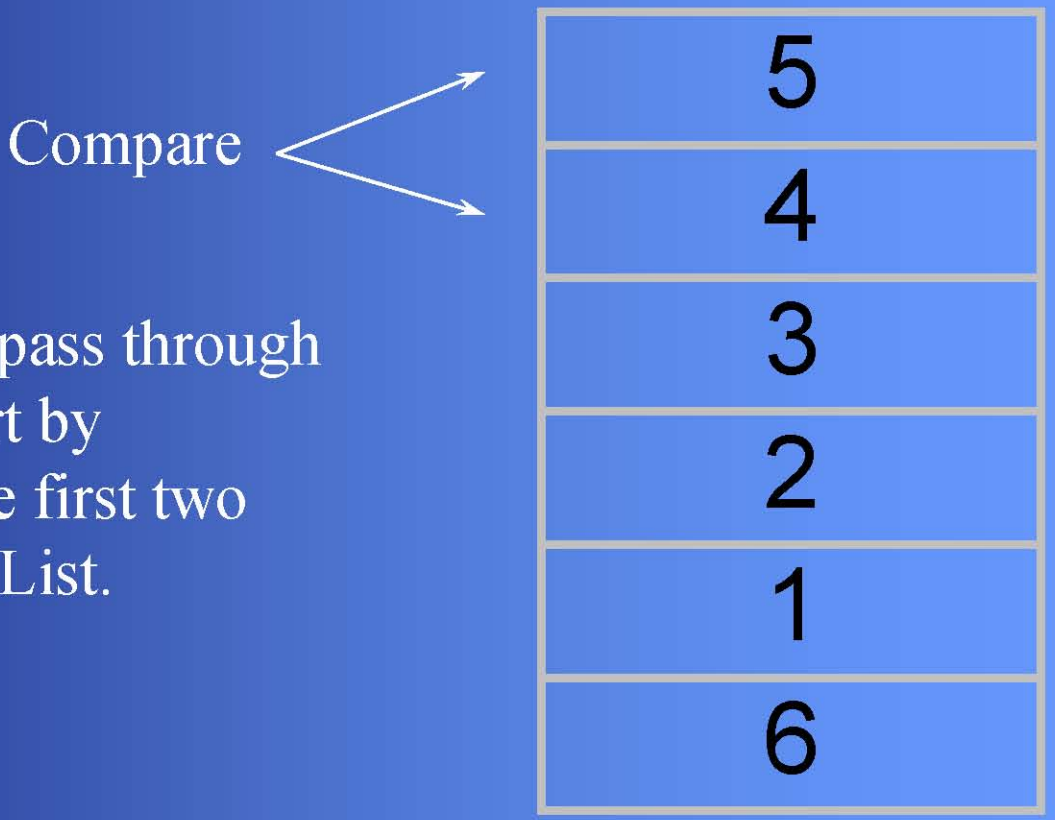
A Bubble Sort Example

As you can see, the largest number

has "bubbled" down, or sunk to the bottom of the List after the first pass through the List.



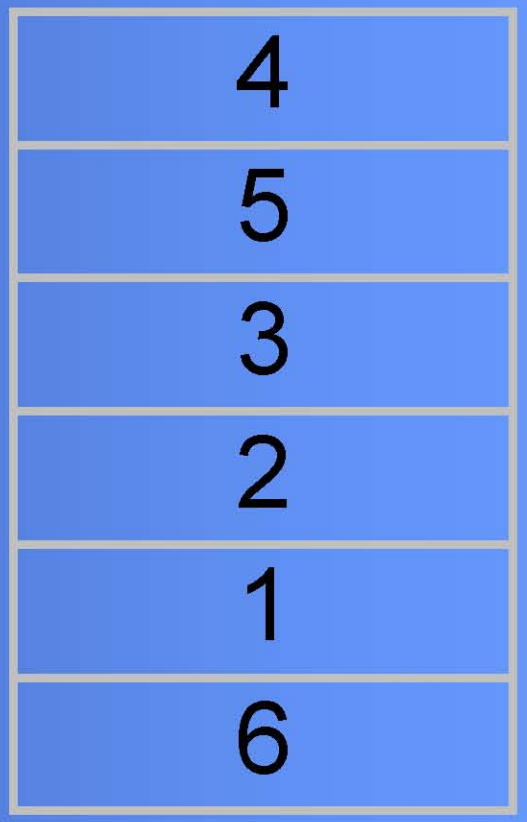
A Bubble Sort Example



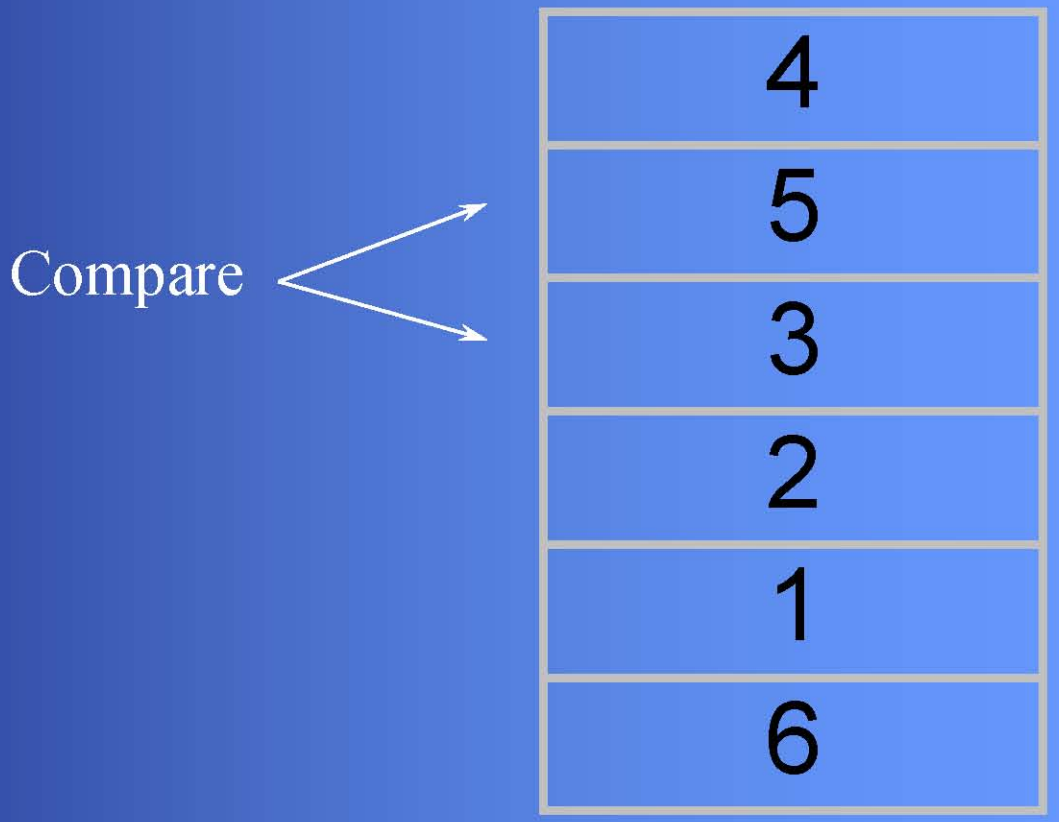
For our second pass through the List, we start by comparing these first two elements in the List.

A Bubble Sort Example

Swap

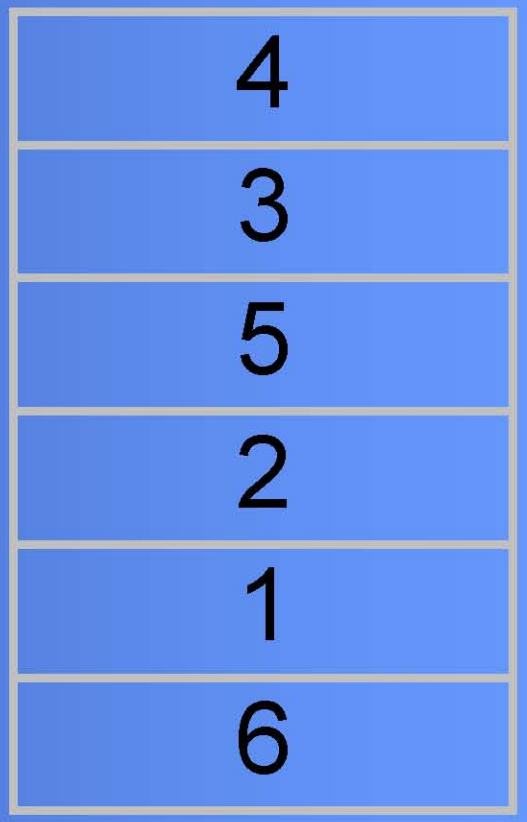


A Bubble Sort Example

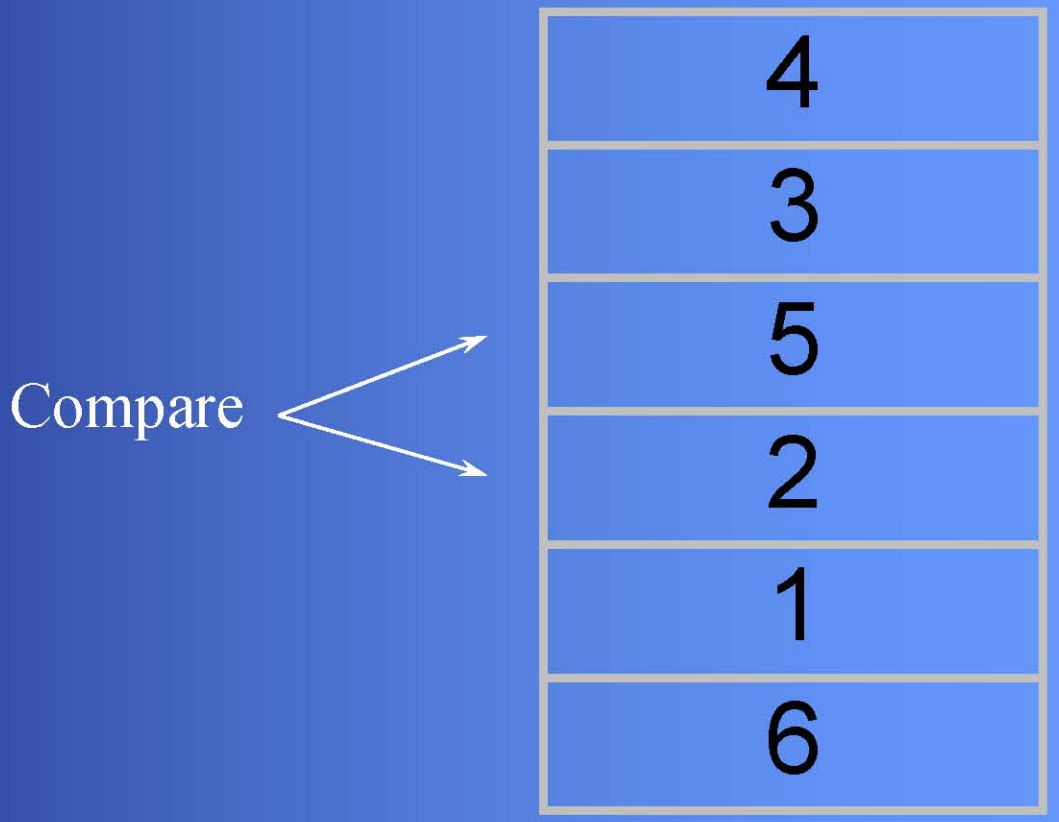


A Bubble Sort Example

Swap

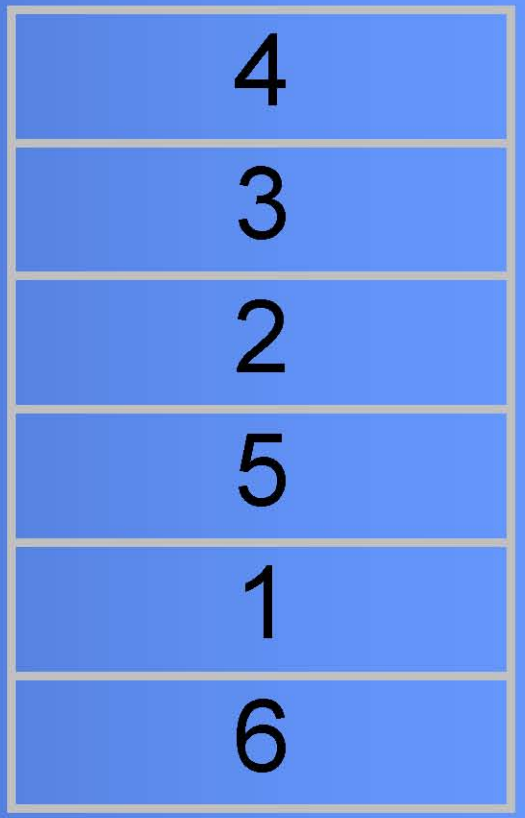


A Bubble Sort Example

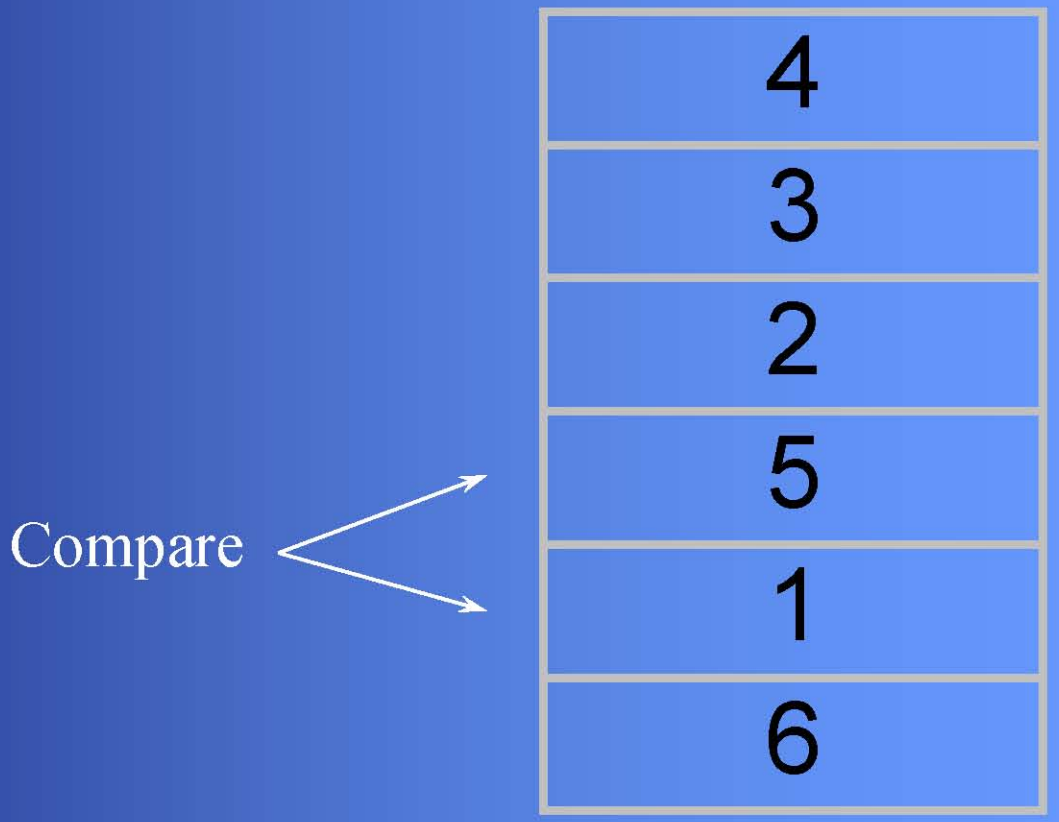


A Bubble Sort Example

Swap



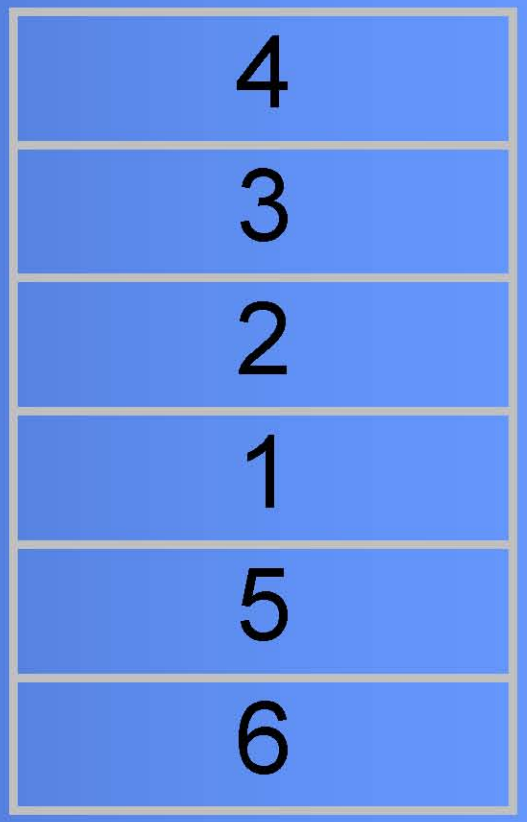
A Bubble Sort Example



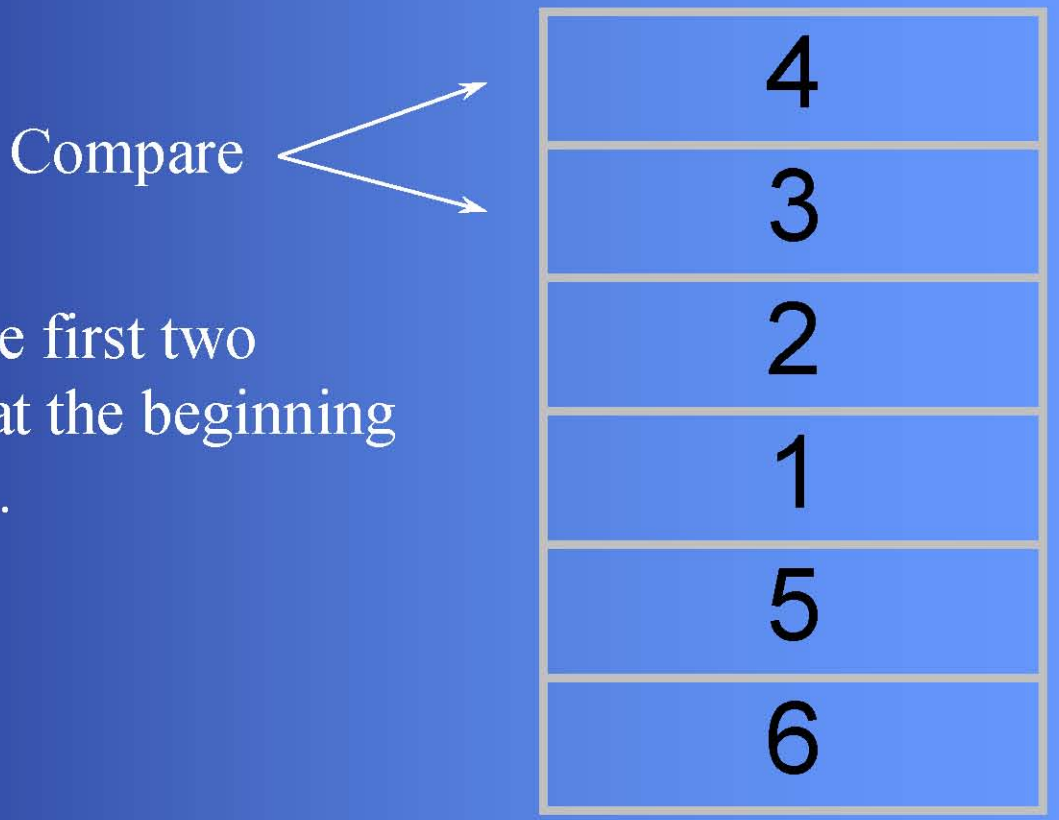
A Bubble Sort Example

At the end of the second pass, we stop at element number n - 1, because the largest element in the List is already in the last position. This places the second largest element in the second to last spot.

Swap



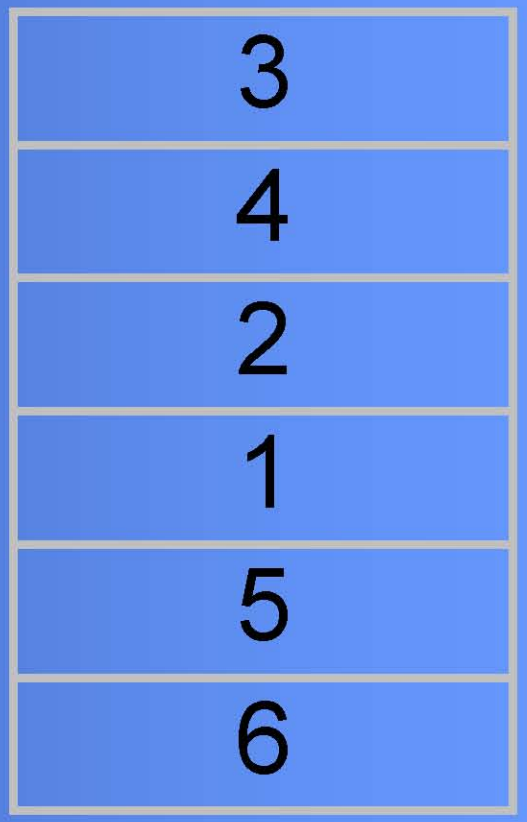
A Bubble Sort Example



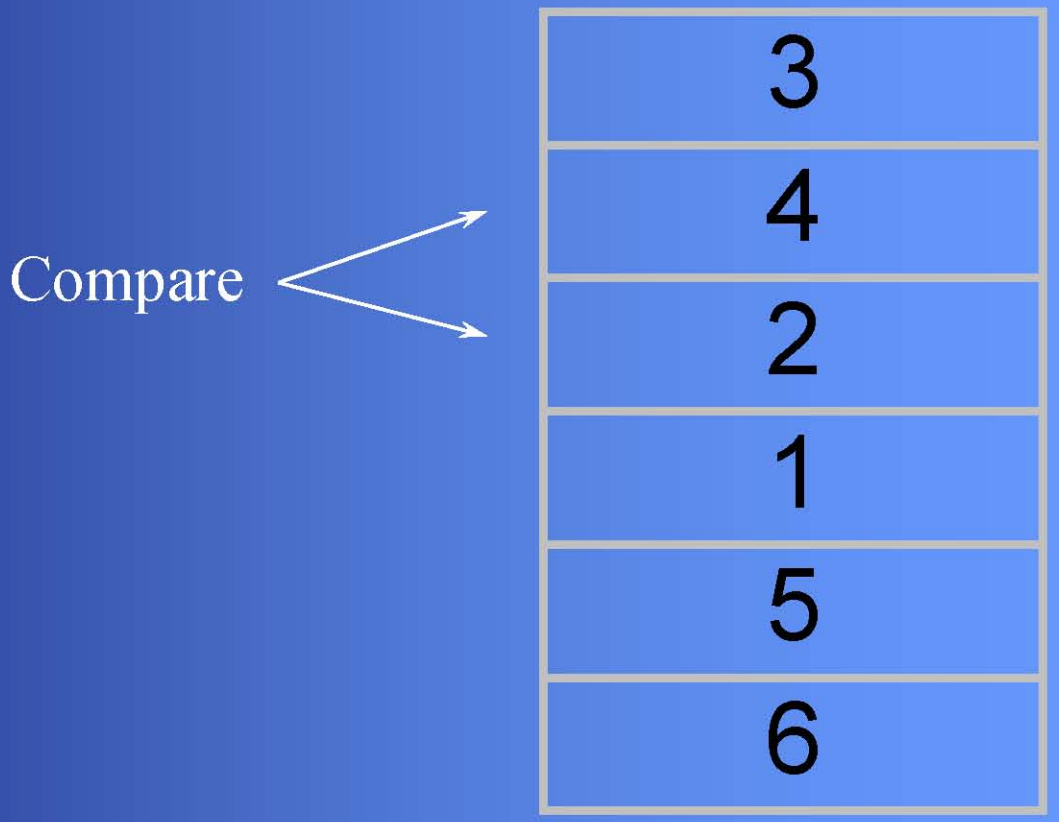
We start with the first two elements again at the beginning of the third pass.

A Bubble Sort Example

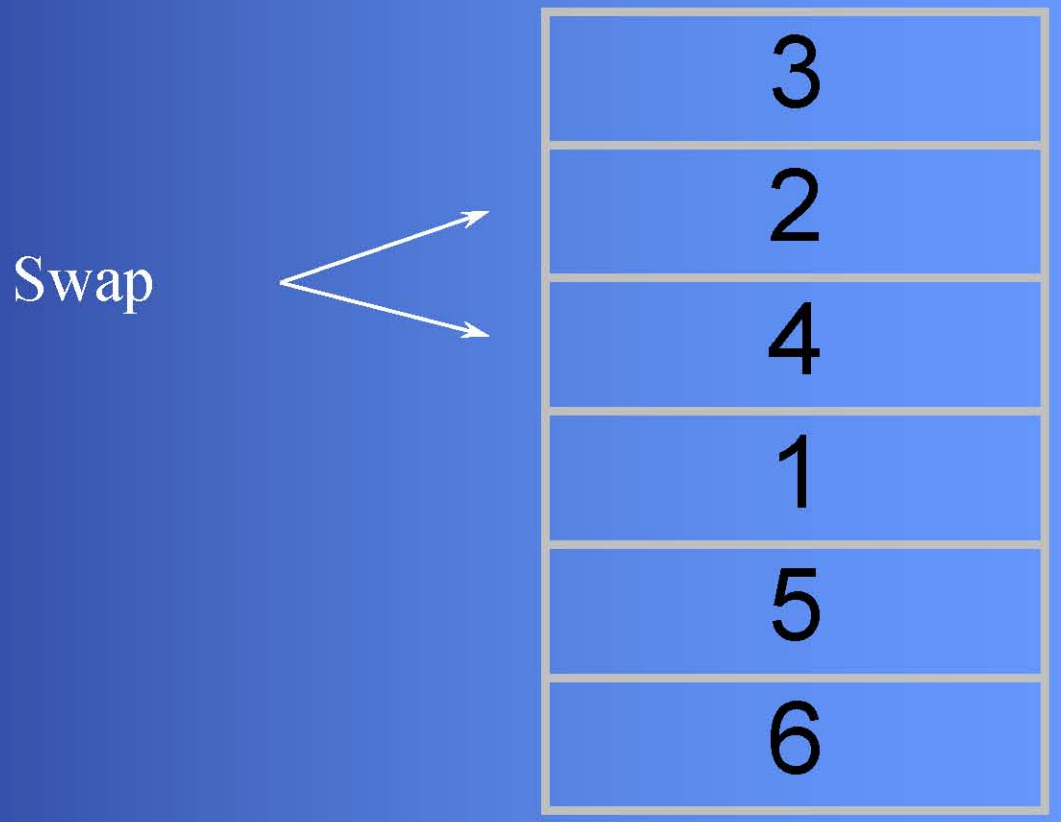
Swap



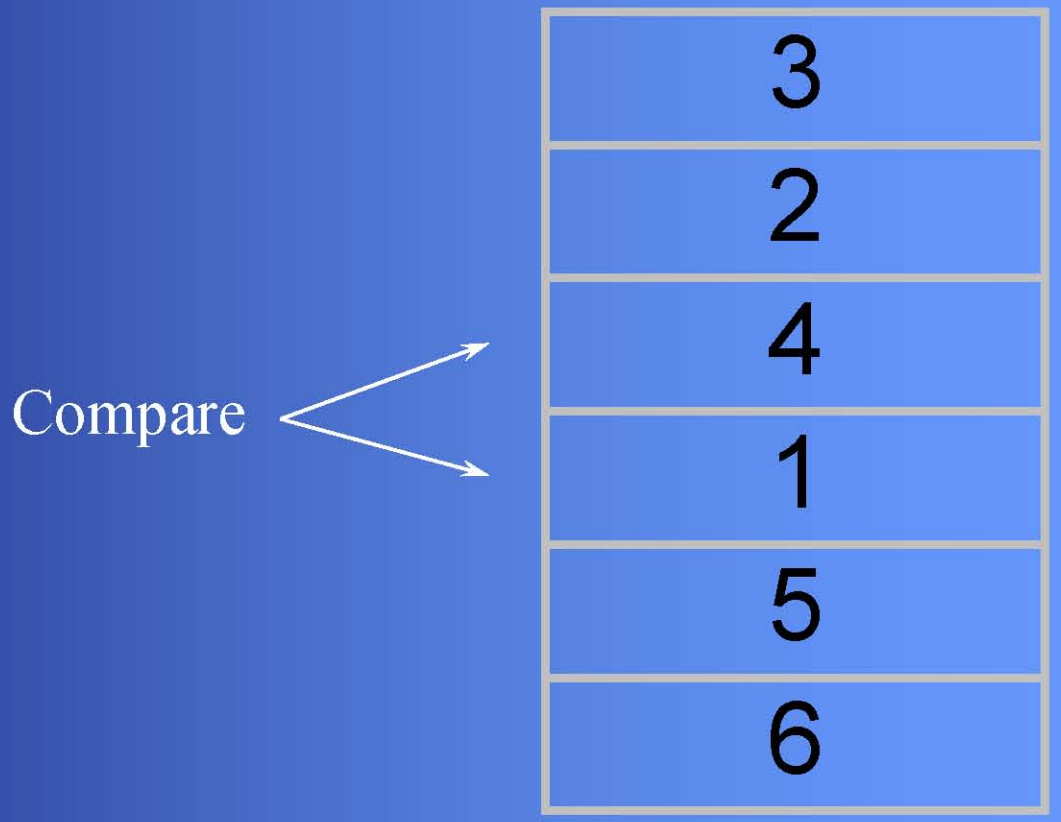
A Bubble Sort Example



A Bubble Sort Example

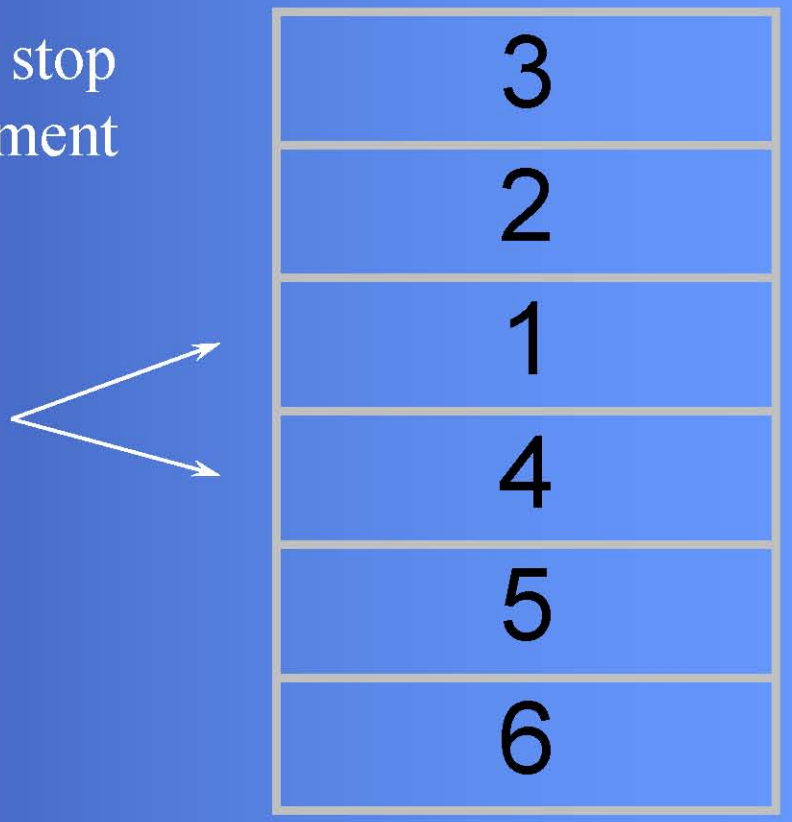


A Bubble Sort Example



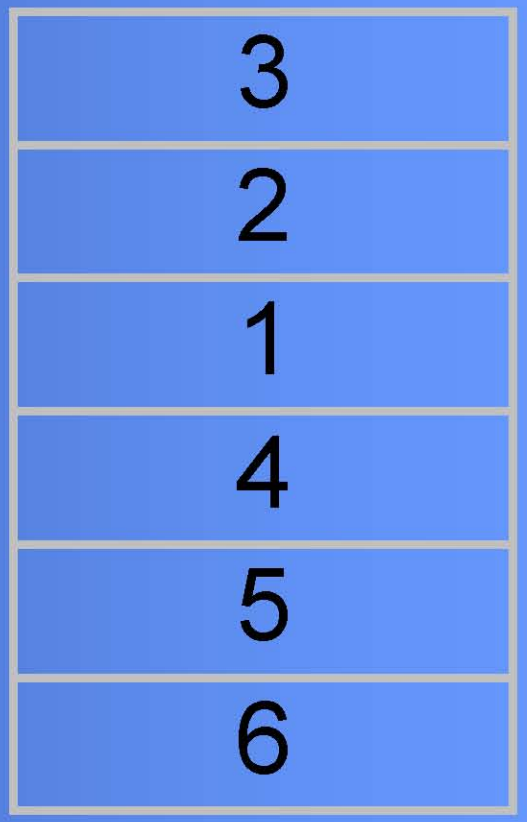
A Bubble Sort Example

At the end of the third pass, we stop comparing and swapping at element number n - 2.



A Bubble Sort Example

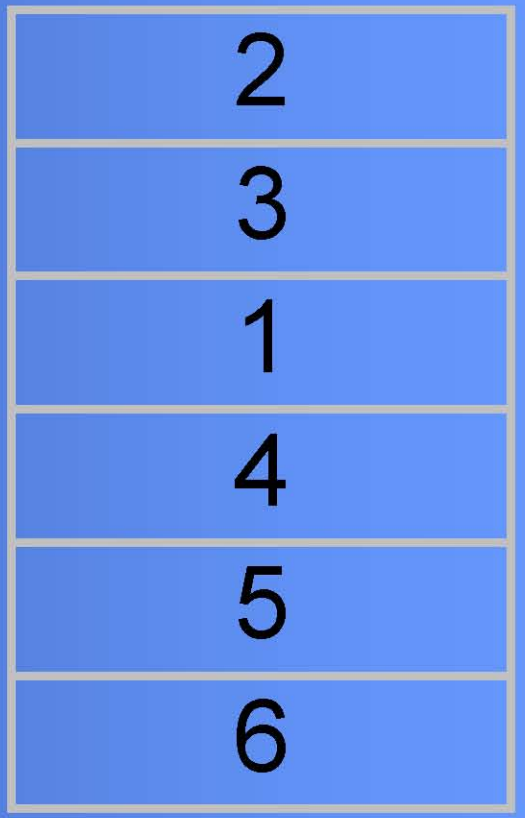
Compare



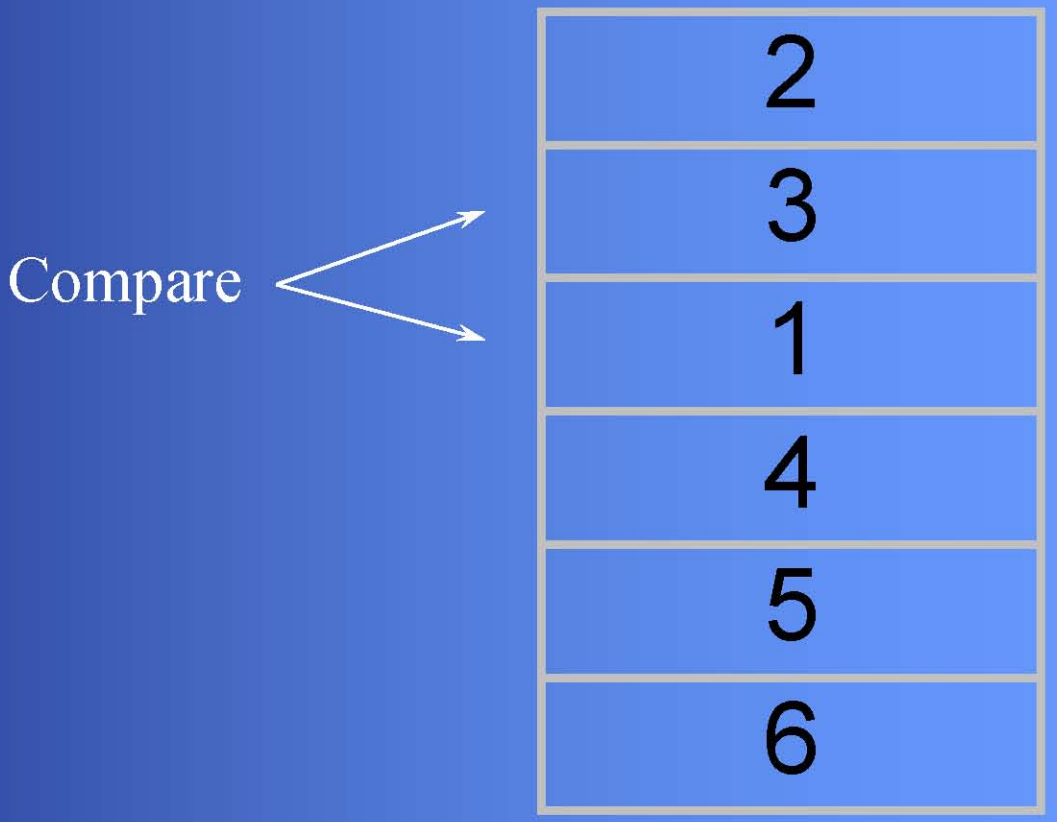
The beginning of the fourth pass...

A Bubble Sort Example

Swap



A Bubble Sort Example



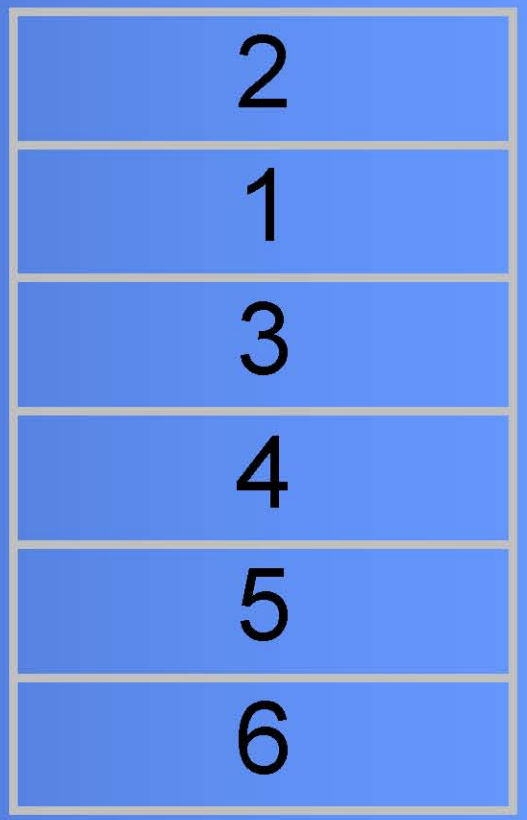
A Bubble Sort Example

Swap

The end of the fourth pass stops at element number n - 3.

A Bubble Sort Example

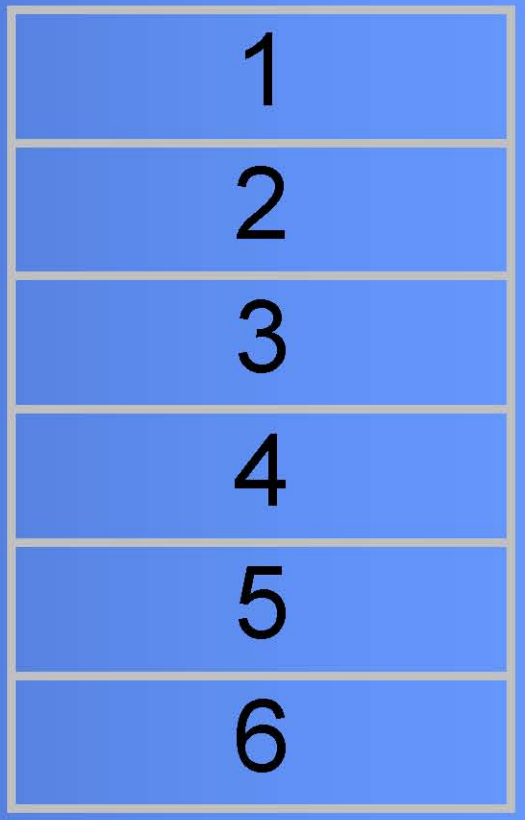
Compare



The beginning of the fifth pass...

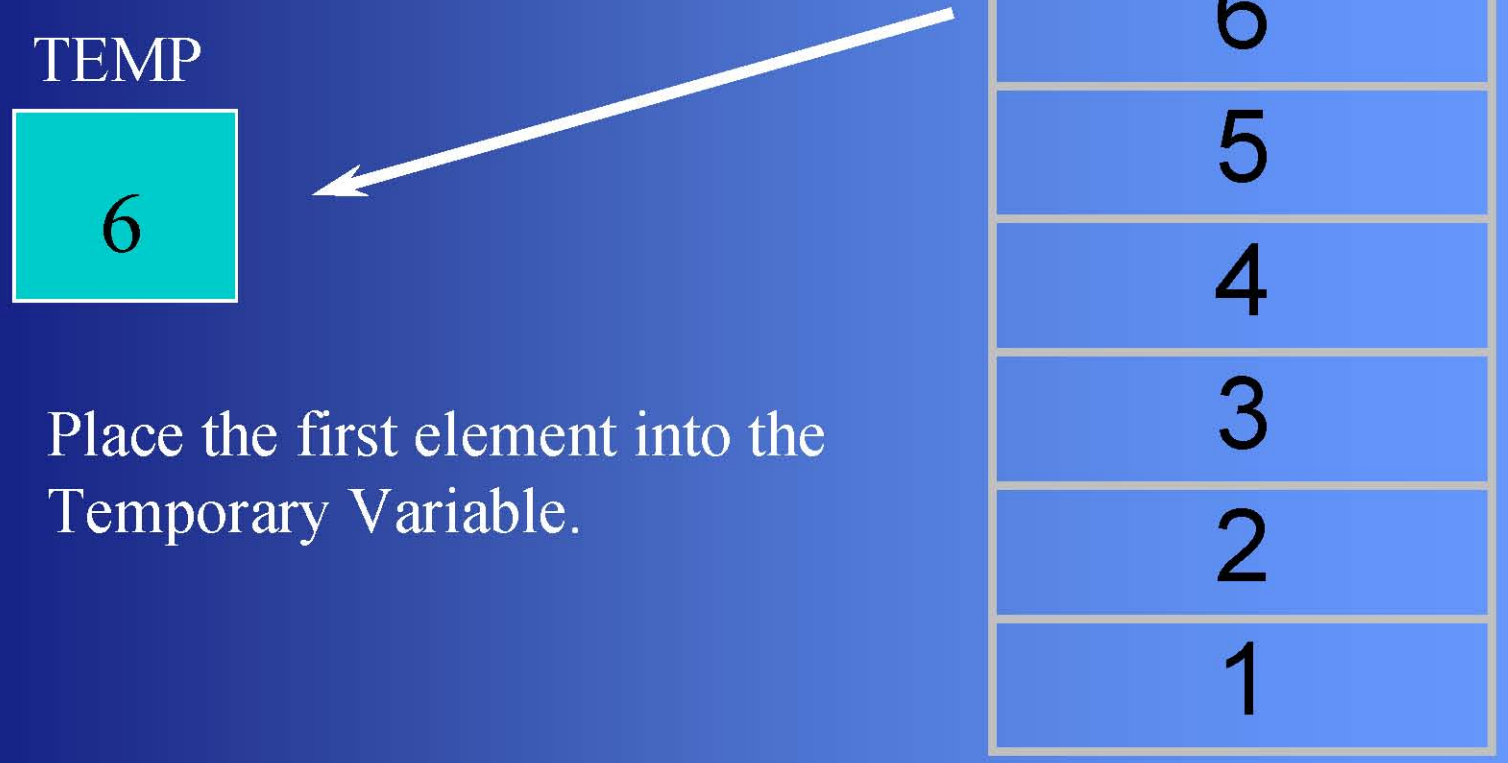
A Bubble Sort Example

Swap



The last pass compares only the first two elements of the List. After this comparison and possible swap, the smallest element has "bubbled" to the top.

What "Swapping" Means



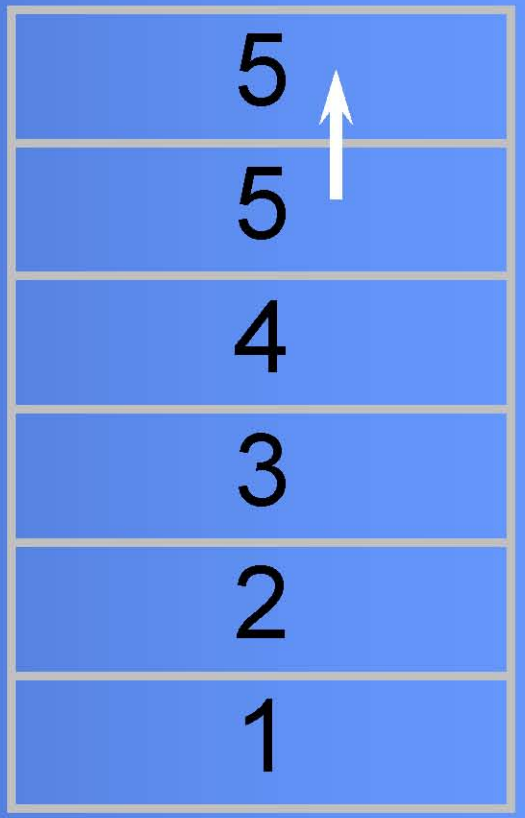
Place the first element into the Temporary Variable.

What "Swapping" Means

TEMP

6

Replace the first element with the second element.



What "Swapping" Means

TEMP

6

5
6
4
3
2
1

Replace the second element with the Temporary Variable.

C + + Examples of The Bubble Sort

C++ Animation For Bubble Sort:

Here Are Some Animated Examples:

On the Net:

http://compsci.exeter.edu/Winter99/CS320/Resources/sortDemo.html

http://www.aist.go.jp/ETL/~suzaki/AlgorithmAnimation/index.html

C++ Code for Swap Procedure

```
void Swap_Data (int &number1, int &number2)
{
    int temp;

    temp = number1;
    number1 = number2;
    number2 = temp;
}
```

C++ Code For Bubble Sort

```
void Bubble_Sort (apvector <int> & v)
{
   int  element, index;

  for (element = 1; element < v.length( ); ++element)
    for (index = v.length( )-1; index >=  element; --index)
      if ( v [index-1] > v [index])
        Swap_Data ( v [index-1], v [index]);
}
```

A More Efficient Algorithm

```
Initialize counter k to zero
Initialize boolean exchange_made to true
While (k < n - 1) and exchange_made
   Set exchange_made to false
   Increment counter k
   For each j from 0 to n - k
      If item in jth position > item in (j + 1)st position
         Swap these items
         Set exchange_made to true
```

More Efficient C++ Source Code

```
void Bubble_Sort(apvector<int> &v)
{ int k = 0;
   bool exchange_made = true;
   while ((k < v.length() - 1) && exchange_made)
    {
      exchange_made = false;
      ++k;
      for (int j = 0; j < v.length() - k; ++j)
         if (v[j] > v[j + 1])
         {
            Swap_Data(v[j], v[j + 1]);
            exchange_made = true;
         }
       } // Make up to n - 1 passes through vector, exit early if no
     exchanges
}     // are made on previous pass
```

Swap_Data Helper Function

```
void Swap_Data (int &number1, int &number2)
{
    int temp;
    temp = number1;
    number1 = number2;
    number2 = temp;
}
// End of Swap_Data function
```

Pascal Code for Swap Procedure

```
procedure Swap (var number1, number2: integer);
var
  temp: integer;
begin
  temp := number1;
  number1 := number2;
  number2 := temp
end;     {Swap}
```

Pascal Code For Bubble Sort

```
procedure BubbleSort (var IntArray: IntNumbers);
  var
    element, index: integer;
begin
  for element := 1 to MaxNum do
    for index := MaxNum downto (element + 1) do
      if IntArray [index] < IntArray [index - 1]
        then Swap (IntArray [index], IntArray [index - 1])
end;     {BubbleSort}
```

BASIC Code For Bubble Sort

```
8000 REM ####################
8010 REM Bubble Sort
8020 REM ####################
8030 FOR ELEMENT = 1 TO MAXNUM - 1
8040 ::FOR INDEX = 1 TO MAXNUM - 1
8050 ::::::IF N (INDEX) < = N (INDEX + 1) THEN GOTO 8090
8060 ::::::TEMP = N (INDEX + 1)
8070 ::::::N (INDEX + 1) = N (INDEX)
8080 ::::::N (INDEX) = TEMP
8090 ::NEXT INDEX
8100 NEXT ELEMENT
8110 RETURN
```

Big - O Notation

Big - O notation is used to describe the efficiency of a search or sort. The actual time necessary to complete the sort varies according to the speed of your system. Big - O notation is an approximate mathematical formula to determine how many operations are necessary to perform the search or sort. The Big - O notation for the Bubble Sort is $O(n^2)$, because it takes approximately n^2 passes to sort the elements.