

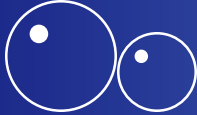
The background is a blue gradient with several white-outlined circles of varying sizes, each containing a small white dot, resembling bubbles. These are scattered around the central text.

The Bubble Sort

by

Mr. Dave Clausen

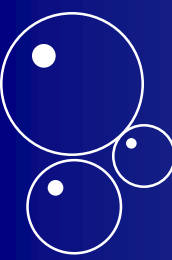
La Cañada High School



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 largest element “bubbles” to the bottom. Repeat this process each time stopping one indexed element less, until you compare only the first two elements in the list. We know that the array is sorted after both of the nested loops have finished.



A Bubble Sort Example

Compare



6
5
4
3
2
1

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

This list is an example of a “worst case” scenario for sorting, because the List is in the exact opposite order from the way we want it sorted (the computer program does not know this).

A Bubble Sort Example

Swap



5
6
4
3
2
1

A Bubble Sort Example

Compare



5
6
4
3
2
1

A Bubble Sort Example

Swap



5
4
6
3
2
1

A Bubble Sort Example

Compare



5
4
6
3
2
1

A Bubble Sort Example

Swap



5
4
3
6
2
1

A Bubble Sort Example

Compare



5
4
3
6
2
1

A Bubble Sort Example

Swap



5
4
3
2
6
1

A Bubble Sort Example

Compare



5
4
3
2
6
1

A Bubble Sort Example

As you can see, the largest number has “bubbled” down to the bottom of the List after the first pass through the List.

Swap



5
4
3
2
1
6

A Bubble Sort Example

Compare



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

5
4
3
2
1
6

A Bubble Sort Example

Swap



4
5
3
2
1
6

A Bubble Sort Example

Compare



4
5
3
2
1
6

A Bubble Sort Example

Swap



4
3
5
2
1
6

A Bubble Sort Example

Compare



4
3
5
2
1
6

A Bubble Sort Example

Swap



4
3
2
5
1
6

A Bubble Sort Example

Compare



4
3
2
5
1
6

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.

Swap



4
3
2
1
5
6

A Bubble Sort Example

Compare



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

4
3
2
1
5
6

A Bubble Sort Example

Swap



3
4
2
1
5
6

A Bubble Sort Example

Compare



3
4
2
1
5
6

A Bubble Sort Example

Swap



3
2
4
1
5
6

A Bubble Sort Example

Compare



3
2
4
1
5
6

A Bubble Sort Example

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

Swap



3
2
1
4
5
6

A Bubble Sort Example

Compare



The beginning of the fourth pass...

3
2
1
4
5
6

A Bubble Sort Example

Swap



2
3
1
4
5
6

A Bubble Sort Example

Compare



2
3
1
4
5
6

A Bubble Sort Example

Swap



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

2
1
3
4
5
6

A Bubble Sort Example

Compare



2
1
3
4
5
6

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.

1
2
3
4
5
6

What “Swapping” Means

TEMP



Place the first element into the
Temporary Variable.

6
5
4
3
2
1

What “Swapping” Means

TEMP

6

Replace the first element with
the second element.

5
5
4
3
2
1

What “Swapping” Means

TEMP



Replace the second element
with the Temporary Variable.

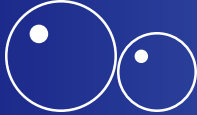
5
6
4
3
2
1

Python Code For Bubble Sort

```
def bubbleSort(listCopy):  
    n = len(listCopy)  
    while n > 1:  
        i = 1  
        while i < n:  
            if listCopy[i] < listCopy[i-1]:  
                swap(listCopy, i, i-1)  
            i += 1  
        n -= 1
```

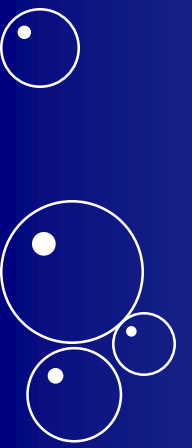
Python Code for Swap Procedure

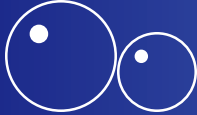
```
def swap(listCopy, x, y):  
    temp = listCopy [x]  
    listCopy [x] = listCopy [y]  
    listCopy [y] = temp
```



Java Code For Bubble Sort

```
public static void bubbleSort(int[ ] list){  
    int k = 0;  
    boolean exchangeMade = true;  
  
    // Make up to n - 1 passes through array, exit early if no exchanges  
    // are made on previous pass  
  
    while ((k < list.length - 1) && exchangeMade){  
        exchangeMade = false;  
        k++;  
        for (int j = 0; j < list.length - k; j++){  
            if (list[j] > list[j + 1]){  
                swap(list, j, j + 1);  
                exchangeMade = true;  
            }  
        }  
    }  
}
```





Java Code for Swap Procedure

```
public static void swap(int[] list, int x, int y)
{
    int temp = list[x];
    list[x] = list[y];
    list[y] = temp;
}
```



C++ Code For Bubble Sort

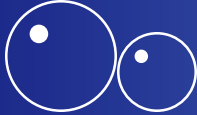
```
void Bubble_Sort (int array, int length)
{
    int element, index;

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


C++ Code for Swap Procedure

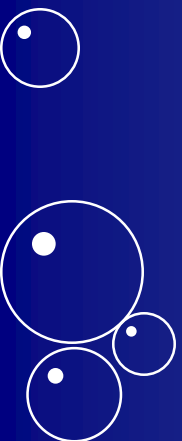
```
void Swap_Data (int &number1, int &number2)
{
    int temp;

    temp = number1;
    number1 = number2;
    number2 = temp;
}
//Swap_Data
```



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}
```



Pascal Code for Swap Procedure

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

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.