Bubble Sort

DSA PRESENTATION

MHARON VERGARA
VINCENT REGIO
LORENZO PASCO
RALPH VILLAPANDO

01

Definition

Bubble Sort is the simplest **sorting algorithm** that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity are quite high.

Example

Simple example of bubble sort

Bubble Sort

Int array[]

(Light) | 7 | 1 | 5 | 6 | 4 | 3 | 2 | (Heavy)

Compare this adjecent elements and checks if they're in order

Check if the first element is greater than the second element

Simple example of bubble sort

Bubble Sort

Int array[]

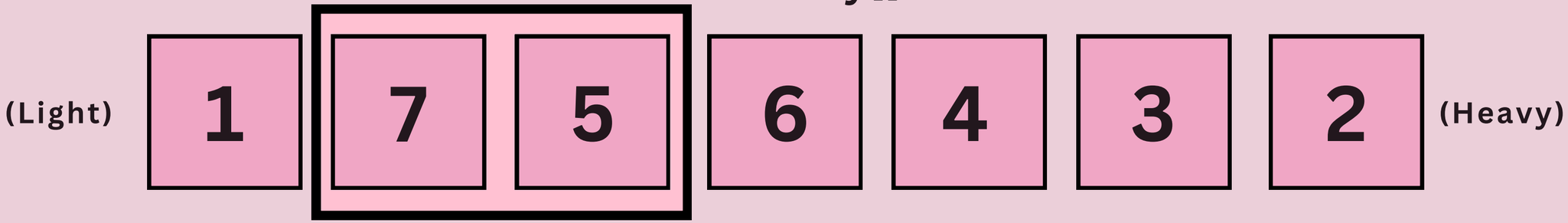
(Light) | 1 | 7 | 5 | 6 | 4 | 3 | 2 | (Heavy)

Swap this two elements because 7 is greater than 1.

Simple example of bubble sort

Bubble Sort

Int array[]



Simple example of bubble sort

Bubble Sort

Int array[]

(Light) | 1 | 5 | 7 | 6 | 4 | 3 | 2 | (Heavy)

Simple example of bubble sort

Bubble Sort

Int array[]

Simple example of bubble sort

Bubble Sort

Int array[]

(Light) | 1 | 5 | 6 | 7 | 4 | 3 | 2 | (Heavy)

Simple example of bubble sort

Bubble Sort

Int array[]

Simple example of bubble sort

Bubble Sort

Int array[]

(Light) | 1 | 5 | 6 | 4 | 7 | 3 | 2 | (Heavy)

Simple example of bubble sort

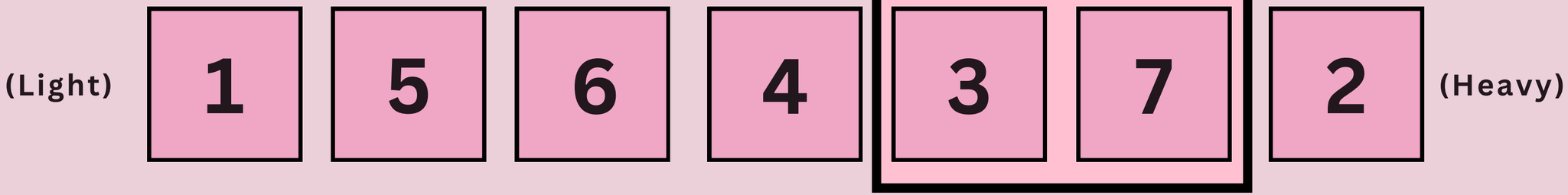
Bubble Sort

Int array[]

Simple example of bubble sort

Bubble Sort

Int array[]



Simple example of bubble sort

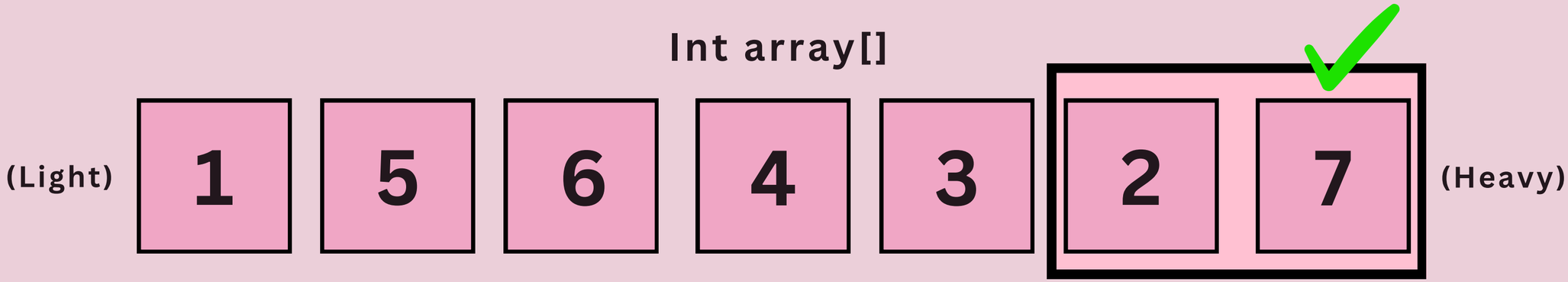
Bubble Sort

Int array[]

Simple example of bubble sort

Bubble Sort

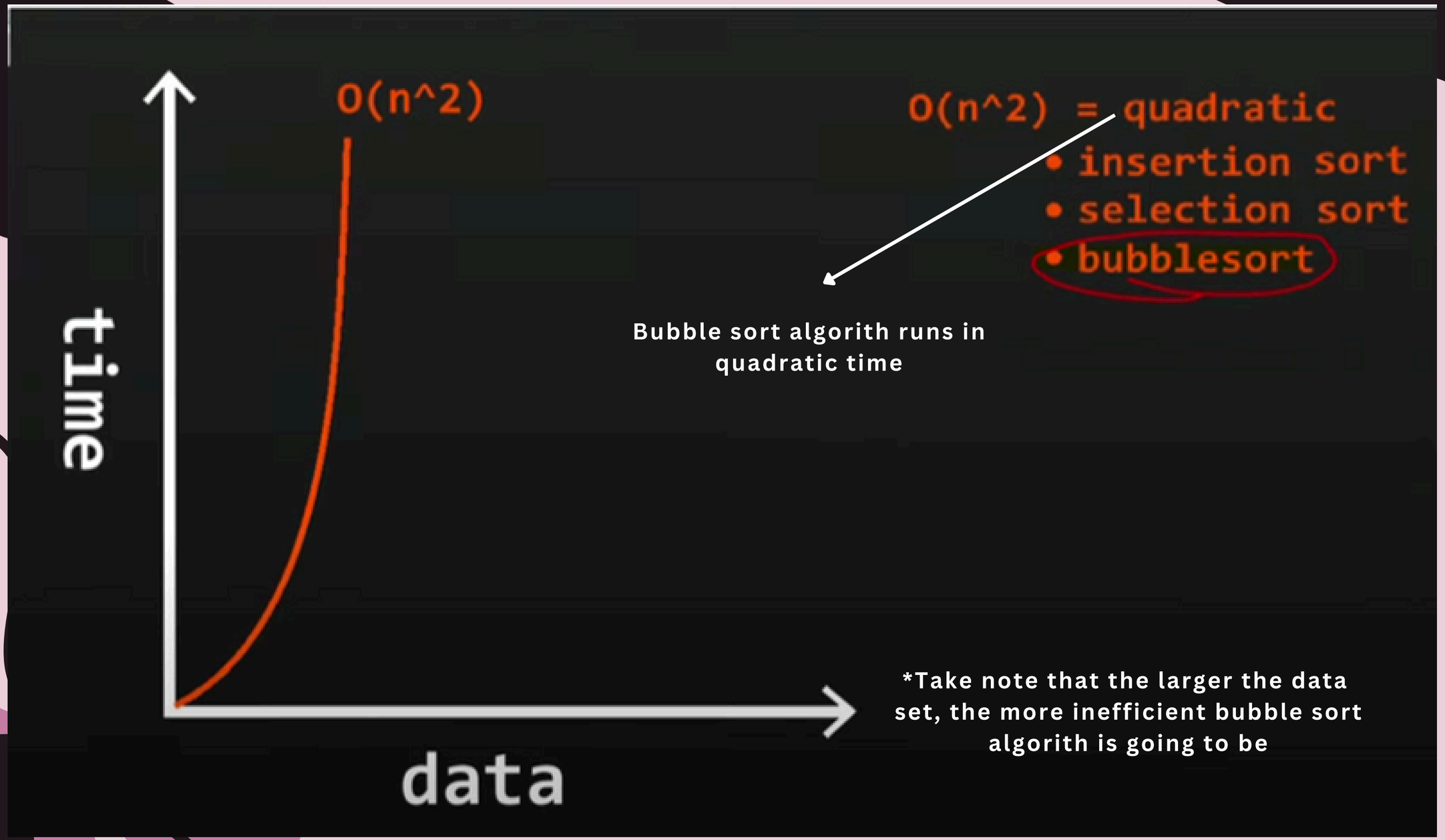
Int array[]



Simple example of bubble sort

Bubble Sort

Int array[]


O(n^2)
time
data
O(n^2) = quadratic
• insertion sort
• selection sort
• bubblesort
Bubble sort algorith runs in quadratic time
*Take note that the larger the data set, the more inefficient bubble sort algorith is going to be

Application

02

Application

When using Bubble sort, the more elements you have, the longer it takes to be sorted.

Here are some scenarios where the Bubble sort algorithm is suitable:

02

Application

Small Data Sets: It is suitable for sorting small datasets or lists where performance is not a critical concern.

Educational Purposes: Bubble sort is often used to teach the basics of sorting algorithms and algorithmic thinking due to its simplicity.

02

Application

Partially Sorted Data: It works well when the data is already nearly sorted, as it has a best-case time complexity of O(n) in such scenarios.

Pseudocode

02 Pseudocode

Bubble Sort Algorithm repeatedly compares adjacent elements in an array, swapping them if they are in the wrong order. This process continues until the array is completely sorted, with larger elements 'bubbling' to the end after each pass. It is simple to implement but inefficient for large datasets due to its O(n2), This means the time taken by the algorithm increases quadratically with the size of the input.

```
Function bubbleSort(arr, n):
    for i = 0 to n - 2 do:                  // Outer loop for passes
        for j = 0 to n - i - 2 do:          // Inner loop for comparisons
            if arr[j] > arr[j + 1] then:    // Check if adjacent elements are out of order
                temp = arr[j]               // Swap elements
                arr[j] = arr[j + 1]
                arr[j + 1] = temp
    return arr                              // Return the sorted array

Main:
    arr = [64, 34, 25, 12, 22, 11, 90]      // Input array
    n = length of arr                       // Find the size of the array
    sortedArr = bubbleSort(arr, n)          // Call bubbleSort function
    print sortedArr                         // Output the sorted array
```

02 Pseudocode

1. Function bubbleSort(arr, n):

- This defines a function named bubbleSort that takes an array arr and its size n as inputs.

2. for i = 0 to n - 2 do:

- The outer loop runs through the array multiple times (n-1 passes).
- Each pass ensures the largest unsorted element is moved to its correct position.

3. for j = 0 to n - i - 2 do:

- The inner loop compares adjacent elements.
- It only checks the unsorted portion of the array because the largest elements are already sorted after each pass.

4. if arr[j] > arr[j + 1] then:

- Compares two adjacent elements.
- If the current element (arr[j]) is larger than the next element (arr[j+1]), they need to be swapped.

5. temp = arr[j]:

- Temporarily stores the value of the larger element in a variable called temp.

6. arr[j] = arr[j + 1]:

- Moves the smaller element to the current position.

7. arr[j + 1] = temp:

- Places the larger element in the next position, completing the swap.

8. return arr:

- Once all elements are sorted, the function returns the sorted array.

9. Main:

- This is where the function is called to perform sorting on a specific input array.

10. arr = [64, 34, 25, 12, 22, 11, 90]:

- An example array is defined as input to the bubbleSort function.

11. n = length of arr:

- Calculates the number of elements in the input array.

12. sortedArr = bubbleSort(arr, n):

- Calls the bubbleSort function with the input array and stores the sorted result in sortedArr.

13. print sortedArr:

- Outputs the sorted array to the screen.

```
Function bubbleSort(arr, n):
    for i = 0 to n - 2 do:                  // Outer loop for passes
        for j = 0 to n - i - 2 do:          // Inner loop for comparisons
            if arr[j] > arr[j + 1] then:    // Check if adjacent elements are out of order
                temp = arr[j]               // Swap elements
                arr[j] = arr[j + 1]
                arr[j + 1] = temp
    return arr                              // Return the sorted array

Main:
    arr = [64, 34, 25, 12, 22, 11, 90]      // Input array
    n = length of arr                       // Find the size of the array
    sortedArr = bubbleSort(arr, n)          // Call bubbleSort function
    print sortedArr                         // Output the sorted array
```

Implementation

02 Implementation

The core function, bubbleSort, takes a reference to a vector of integers. Its primary role is to sort the input vector in ascending order. The function uses two nested loops. The outer loop iterates over the vector, while the inner loop compares and potentially swaps adjacent elements. To optimize the algorithm, a Boolean flag, swapped, is introduced. This flag monitors whether any swaps occurred during an iteration. If no swaps are detected, the function breaks out of the loop early, as the list is already sorted. This enhancement significantly reduces unnecessary comparisons in cases where the input is nearly sorted.

```cpp
#include <bits/stdc++.h>
using namespace std;

// An optimized version of Bubble Sort
void bubbleSort(vector<int>& arr) {
    int n = arr.size();
    bool swapped;

    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

// Function to print a vector
void printVector(const vector<int>& arr) {
    for (int num : arr)
        cout << " " << num;
}

int main() {
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
    bubbleSort(arr);
    cout << "Sorted array: \n";
    printVector(arr);
    return 0;
}
```

02 Implementation

The program also includes a utility function, printVector, which outputs the elements of the vector to the console. This function iterates through the vector and displays each element, formatted for readability.

The main function ties the implementation together. It initializes a sample vector with unsorted integers, calls the bubbleSort function, and then displays the sorted result using printVector. This sequence demonstrates how the algorithm operates and verifies its correctness.

```cpp
#include <bits/stdc++.h>
using namespace std;

// An optimized version of Bubble Sort
void bubbleSort(vector<int>& arr) {
    int n = arr.size();
    bool swapped;

    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

// Function to print a vector
void printVector(const vector<int>& arr) {
    for (int num : arr)
        cout << " " << num;
}

int main() {
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
    bubbleSort(arr);
    cout << "Sorted array: \n";
    printVector(arr);
    return 0;
}
```

Thank You