# Today’s Date: 23-Nov-2024

Vineeth: what is the easiest sorting algorithms anyone can understand? explain that sort process for me

Microsoft Copilot:

One of the easiest sorting algorithms to understand is **Bubble Sort**. It's simple and intuitive, making it a great starting point for learning about sorting algorithms. Here's how Bubble Sort works:

**Bubble Sort Process**

1. **Start at the Beginning**: Begin with the first element of the list.
2. **Compare Adjacent Elements**: Compare the current element with the next element.
3. **Swap if Necessary**: If the current element is greater than the next element, swap them.
4. **Move to the Next Pair**: Move to the next pair of elements and repeat the comparison and swap steps.
5. **Repeat for Each Element**: Continue this process for each element in the list.
6. **Pass Through the List**: After one complete pass through the list, the largest element will have "bubbled up" to its correct position at the end of the list.
7. **Repeat Passes**: Repeat the entire process for the remaining elements, excluding the last sorted elements, until no more swaps are needed.

**Example**

Let's sort the list [5, 3, 8, 4, 2] using Bubble Sort:

1. **First Pass**:
   * Compare 5 and 3: Swap → [3, 5, 8, 4, 2]
   * Compare 5 and 8: No swap → [3, 5, 8, 4, 2]
   * Compare 8 and 4: Swap → [3, 5, 4, 8, 2]
   * Compare 8 and 2: Swap → [3, 5, 4, 2, 8]
2. **Second Pass**:
   * Compare 3 and 5: No swap → [3, 5, 4, 2, 8]
   * Compare 5 and 4: Swap → [3, 4, 5, 2, 8]
   * Compare 5 and 2: Swap → [3, 4, 2, 5, 8]
   * 8 is already in its correct position.
3. **Third Pass**:
   * Compare 3 and 4: No swap → [3, 4, 2, 5, 8]
   * Compare 4 and 2: Swap → [3, 2, 4, 5, 8]
   * 5 and 8 are in their correct positions.
4. **Fourth Pass**:
   * Compare 3 and 2: Swap → [2, 3, 4, 5, 8]
   * 3, 4, 5, and 8 are in their correct positions.
5. **Fifth Pass**:
   * No swaps needed, the list is sorted.

**Final Sorted List**

[2, 3, 4, 5, 8]

**Key Points**

* **Simplicity**: Easy to understand and implement.
* **Inefficiency**: Not suitable for large datasets due to its $O(n^2)$ time complexity.
* **Use Case**: Good for small or nearly sorted datasets.

Would you like to know more about another sorting algorithm or any other topic?

## what is the pseudocode for bubble sort? explain the pseudocode to me

Here's the pseudocode for Bubble Sort, along with an explanation of each step:

**Pseudocode for Bubble Sort**

function BUBBLESORT(array)

n = length(array)

for i from 0 to n-1

for j from 0 to n-i-2

if array[j] > array[j+1]

swap(array[j], array[j+1])

end if

end for

end for

end function

**Explanation**

1. **Function Definition**:
   * function BUBBLESORT(array): This defines the Bubble Sort function that takes an array as input.
2. **Initialize Length**:
   * n = length(array): Determine the length of the array and store it in n.
3. **Outer Loop**:
   * for i from 0 to n-1: This loop runs from the first element to the last element of the array. It ensures that the process is repeated enough times to sort the entire array.
4. **Inner Loop**:
   * for j from 0 to n-i-2: This loop runs through the array, comparing each pair of adjacent elements. The range decreases with each pass because the largest elements "bubble up" to their correct positions at the end of the array.
5. **Comparison and Swap**:
   * if array[j] > array[j+1]: If the current element is greater than the next element, they are out of order.
   * swap(array[j], array[j+1]): Swap the elements to put them in the correct order.
6. **End of Loops**:
   * The inner loop ends, and the outer loop continues to the next iteration until the entire array is sorted.

**Example Walkthrough**

Let's sort the array [5, 3, 8, 4, 2] using Bubble Sort:

1. **First Pass**:
   * Compare 5 and 3: Swap → [3, 5, 8, 4, 2]
   * Compare 5 and 8: No swap → [3, 5, 8, 4, 2]
   * Compare 8 and 4: Swap → [3, 5, 4, 8, 2]
   * Compare 8 and 2: Swap → [3, 5, 4, 2, 8]
2. **Second Pass**:
   * Compare 3 and 5: No swap → [3, 5, 4, 2, 8]
   * Compare 5 and 4: Swap → [3, 4, 5, 2, 8]
   * Compare 5 and 2: Swap → [3, 4, 2, 5, 8]
   * 8 is already in its correct position.
3. **Third Pass**:
   * Compare 3 and 4: No swap → [3, 4, 2, 5, 8]
   * Compare 4 and 2: Swap → [3, 2, 4, 5, 8]
   * 5 and 8 are in their correct positions.
4. **Fourth Pass**:
   * Compare 3 and 2: Swap → [2, 3, 4, 5, 8]
   * 3, 4, 5, and 8 are in their correct positions.
5. **Fifth Pass**:
   * No swaps needed, the list is sorted.

**Final Sorted List**

[2, 3, 4, 5, 8]

Bubble Sort is easy to understand and implement, making it a great starting point for learning about sorting algorithms. However, it's not the most efficient for large datasets due to its $O(n^2)$ time complexity.

Would you like to explore another sorting algorithm or any other topic?

## c program code for bubble sort

Sure! Here's a simple C program to implement Bubble Sort:

#include <stdio.h>

// Function to perform Bubble Sort

void bubbleSort(int array[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (array[j] > array[j+1]) {

// Swap array[j] and array[j+1]

int temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

}

// Function to print the array

void printArray(int array[], int size) {

for (int i = 0; i < size; i++) {

printf("%d ", array[i]);

}

printf("\n");

}

int main() {

int array[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(array)/sizeof(array[0]);

printf("Unsorted array: \n");

printArray(array, n);

bubbleSort(array, n);

printf("Sorted array: \n");

printArray(array, n);

return 0;

}

**Explanation**

1. **Include Standard I/O Library**:
   * #include <stdio.h>: This includes the standard input-output library for using printf and other I/O functions.
2. **Bubble Sort Function**:
   * void bubbleSort(int array[], int n): This function takes an array and its size as parameters and sorts the array using the Bubble Sort algorithm.
   * The outer loop runs from 0 to n-1, and the inner loop runs from 0 to n-i-1.
   * Inside the inner loop, adjacent elements are compared and swapped if they are in the wrong order.
3. **Print Array Function**:
   * void printArray(int array[], int size): This function prints the elements of the array.
   * It iterates through the array and prints each element followed by a space.
4. **Main Function**:
   * int main(): The main function where the program execution begins.
   * An array of integers is defined and its size is calculated.
   * The unsorted array is printed using printArray.
   * The bubbleSort function is called to sort the array.
   * The sorted array is printed using printArray.

This program demonstrates the basic implementation of Bubble Sort in C. Feel free to run and modify it to see how it works!

Is there anything else you'd like to know or any other algorithm you're interested in?

## JAVA program code for bubble sort

Sure! Here's a simple Java program to implement Bubble Sort:

public class BubbleSort {

// Function to perform Bubble Sort

static void bubbleSort(int[] array) {

int n = array.length;

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (array[j] > array[j+1]) {

// Swap array[j] and array[j+1]

int temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

}

// Function to print the array

static void printArray(int[] array) {

int n = array.length;

for (int i = 0; i < n; i++) {

System.out.print(array[i] + " ");

}

System.out.println();

}

public static void main(String[] args) {

int[] array = {64, 34, 25, 12, 22, 11, 90};

System.out.println("Unsorted array:");

printArray(array);

bubbleSort(array);

System.out.println("Sorted array:");

printArray(array);

}

}

**Explanation**

1. **Class Definition**:
   * public class BubbleSort: Defines the class BubbleSort.
2. **Bubble Sort Function**:
   * static void bubbleSort(int[] array): This static method takes an array as a parameter and sorts it using the Bubble Sort algorithm.
   * The outer loop runs from 0 to n-1, and the inner loop runs from 0 to n-i-1.
   * Inside the inner loop, adjacent elements are compared and swapped if they are in the wrong order.
3. **Print Array Function**:
   * static void printArray(int[] array): This static method prints the elements of the array.
   * It iterates through the array and prints each element followed by a space.
4. **Main Method**:
   * public static void main(String[] args): The main method where the program execution begins.
   * An array of integers is defined.
   * The unsorted array is printed using printArray.
   * The bubbleSort method is called to sort the array.
   * The sorted array is printed using printArray.

This Java program demonstrates the basic implementation of Bubble Sort. You can run and modify it to see how it works!

Is there anything else you'd like to know or any other algorithm you're interested in?

## PYTHON CODE FOR BUBBLE SORT

Sure! Here's a simple Python program to implement Bubble Sort:

def bubble\_sort(array):

n = len(array)

for i in range(n):

for j in range(0, n-i-1):

if array[j] > array[j+1]:

# Swap array[j] and array[j+1]

array[j], array[j+1] = array[j+1], array[j]

def print\_array(array):

for i in array:

print(i, end=" ")

print()

# Example usage

array = [64, 34, 25, 12, 22, 11, 90]

print("Unsorted array:")

print\_array(array)

bubble\_sort(array)

print("Sorted array:")

print\_array(array)

**Explanation**

1. **Bubble Sort Function**:
   * def bubble\_sort(array): This function takes a list (array) as a parameter and sorts it using the Bubble Sort algorithm.
   * The outer loop runs from 0 to n-1, and the inner loop runs from 0 to n-i-1.
   * Inside the inner loop, adjacent elements are compared and swapped if they are in the wrong order.
2. **Print Array Function**:
   * def print\_array(array): This function prints the elements of the list.
   * It iterates through the list and prints each element followed by a space.
3. **Example Usage**:
   * An example list of integers is defined.
   * The unsorted list is printed using print\_array.
   * The bubble\_sort function is called to sort the list.
   * The sorted list is printed using print\_array.

This Python program demonstrates the basic implementation of Bubble Sort. You can run and modify it to see how it works!

Is there anything else you'd like to know or any other algorithm you're interested in?

# Today’s Date: 24-Nov-2024

### create a new repository on the command line

echo "# 2024-GIT" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/vineethmourya/2024-GIT.git

git push -u origin main

**learned how to sync local with git:**

1. create a git hub repository 2024-GIT.
2. create local files in documents/my\_learning/2024\_git/(all my files).
3. Open shell
4. Check by typing ‘git’ in CLI. Verify if you see some related info (to ensure git bash is installed).
5. Cd to “C:\Users\VMANTHI1\Documents\MY\_LEARNING\2024\_GIT”
6. Issue “git init”
7. Issue “git status”
8. Issue “git add .”
9. Now commit, issue “git commit -m "first commit"”
10. Issue “git remote add origin <https://github.com/vineethmourya/2024-GIT.git>”
11. Issue “git push -u origin master”

Since github.com is blocked in ford, I cannot push my files from local.

## write a db2 query to check if each item in a list is present in the table or not:

SELECT T1.SPN,

CASE

WHEN E14.SPART\_PART\_NBR IS NOT NULL THEN 'Present'

ELSE 'Not Present'

END AS status

FROM (

SELECT 'HUK15' AS SPN FROM SYSIBM.SYSDUMMY1 UNION ALL

SELECT 'ABCD' AS SPN FROM SYSIBM.SYSDUMMY1 UNION ALL

SELECT '1234' AS SPN FROM SYSIBM.SYSDUMMY1 UNION ALL

SELECT 'S41A' AS SPN FROM SYSIBM.SYSDUMMY1

) AS T1

LEFT JOIN DB23.LPU@PDB.SMMPE14\_FPSD\_SV\_PT E14

ON T1.SPN = E14.SPART\_PART\_NBR

WITH UR;

---------------------------------------------------------------------

DOES ‘WITH UR;’ INCREASE EXEC SPEED OF QUERY?

WITH UR: Query execution time => 5 s: 801 ms

WITHOUT UR: Query execution time => 5 s: 803 ms

WITH CS: Query execution time => 5 s: 693 ms

WITH RR: Query execution time => 6 s: 75 ms

WITH RS: -904 ISSUE

SELECT ROW\_NUMBER() OVER (ORDER BY E14.SPART\_PART\_NBR) AS #,

E14.SPART\_PART\_NBR,

E14.SPART\_ESTBLISH\_Y,

E14.P\_LAST\_UPDATE\_S

FROM DB23.LPU@PDB.SMMPE14\_FPSD\_SV\_PT E14

WHERE NOT EXISTS

(SELECT E12.SBASEPRT\_NBR,

E12.SPRFXPRT\_PREFIX\_R,

E12.SUFXPRT\_SUFFIX\_NBR

FROM DB23.LPU@PDB.SMMPE12\_SENG\_EQL E12

WHERE E12.SBASEPRT\_NBR = E14.SBASEPRT\_NBR

AND E12.SPRFXPRT\_PREFIX\_R = E14.SPRFXPRT\_PREFIX\_R

AND E12.SUFXPRT\_SUFFIX\_NBR = E14.SUFXPRT\_SUFFIX\_NBR

)

WITH RR;