Insertion sort algorithm

Source: ***https://github.com/rusevrosen/codehub.github.io/tree/main/Algorithms/Sorting_Algorithms***

1. *What is the Insertion Sort algorithm?*

Insertion sort is an algorithm that sorts a list of items by repeatedly comparing an element to the elements that come before it and swapping them if they are out of order. It works by iterating through the list of items, taking one element at a time, and inserting it into its correct position in the sorted portion of the list.

Characteristics of Insertion Sort algorithm:

- This algorithm is one of the simplest algorithms with a simple implementation.
- Basically, Insertion sort is efficient for small data values.
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets that are already partially sorted.

2. *Code example.*

```
#include <iostream>

using namespace std;

void insertionSort(int arr[], int n) {
  int i, j, temp;
  for (i = 1; i < n; i++) {
    j = i;
    while (j > 0 && arr[j] < arr[j-1]) {
      temp = arr[j];
      arr[j] = arr[j-1];
      arr[j-1] = temp;
      j--;
    }
  }
}

int main() {
  int arr[] = {5, 2, 4, 6, 1, 3};
  int n = sizeof(arr) / sizeof(arr[0]);

  insertionSort(arr, n);

  for (int i = 0; i < n; i++) {
    cout << arr[i] << " ";
  }

  return 0;
}
```

The output of the example is: 1 2 3 4 5 6.

3. *Code explanation.*

The **insertionSort** function performs the insertion sort algorithm. It takes an array **arr** and the size of the array **n** as arguments.

The function begins by declaring three variables: **i, j**, and **temp**. **i** is a loop counter that will be used to iterate through the array, **j** is another loop counter that will be used to find the correct position for the current element, and **temp** is a temporary variable that will be used to store the value of an element while it is being swapped.

The outer for loop iterates through the array, starting at the second element (index 1). The inner while loop compares the current element (**arr[j]**) to the elements that come before it (**arr[j-1]**), and swaps them if they are out of order. This continues until the current element is in its correct position in the sorted portion of the array.

The main function of the program.

It first declares an array **arr** with the values **{5, 2, 4, 6, 1, 3}** and a variable **n** that stores the size of the array.

The function then calls the **insertionSort** function, passing it the array and its size as arguments. This sorts the array in ascending order.

The function uses a for **loop** to iterate through the sorted array and print each element, separated by a space. The output of the program will be the sorted version of the input array: **1 2 3 4 5 6**.

4. *Time complexity.*

The time complexity of the insertion sort algorithm, as shown in the C++ example above, is **O(n^2).**

This is because the algorithm uses nested loops to iterate through the array, and the time taken to complete the sort increases quadratically with the size of the input.

In the worst case, where the array is initially in reverse order, the inner loop will have to make **n-1** comparisons and swaps for each element, leading to a total of **(n-1) + (n-2) + ... + 1** comparison, which is equal to **n(n-1)/2**. This is a quadratic function, so the time complexity is **O(n^2).**

In the best case, where the array is already sorted, the inner loop will not have to make any comparisons or swaps, so the time complexity will be **O(n).** However, the average case time complexity is still **O(n^2),** so insertion sort is generally not considered to be a very efficient algorithm for large inputs.

. **I wish you happy learning,**

Your Rosen Rusev