Bubble Sort Vs Heap Sort

Submitted by - Anu Thomas

Submitted on - 07/09/2022

BUBBLE SORT

Bubble sort is a sorting algorithm that matches two adjacent pairs things and change them until they are in the order you want.

Working of Bubble Sort:

1. Initial step

   From the first index, compare the first and the second elements. If the first element is greater than the second element, they are swapped. Then, compare the second and the third elements. Swap them if they are not in order. This process goes on until the last element.

2. Remaining Iteration

   The same process goes on for the remaining iterations. After each iteration, the largest element among the unsorted elements is placed at the end. In each iteration, the comparison takes place up to the last unsorted element. The array is sorted when all the unsorted elements are placed at their correct positions.

Algorithm

Step 1: Start

Step 2: Get the list number of items to be sorted.

Step 3: Determine the number of external passes (n - 1) to be performed. Its length is list minus one.

Step 4: Perform inner passes (n – 1) times for outer pass 1. Get the first element value and compare the value with the second value. If the second value is less than the first value, then swap the positions(in the case of ascending order).

Step 5: Repeat step 4 until you reach the outer pass . Get the next element in the list then repeat the process that was performed in step 4 until all the values have been placed in their correct ascending order.

Step 6: Return the result when all passes have been done.

Step 7: Print result.

Step 8: Stop

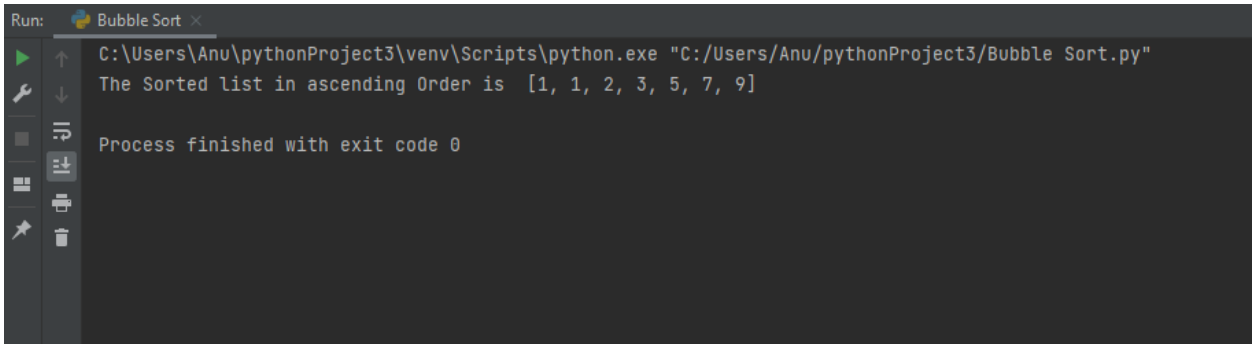
For Example here is the python program for doing bubble sort in ascending order.

```
def bubble_sort(number_list):
    # loop to access each array element
    for looping in range(len(number_list)-1, 0, -1):
        for num in range(looping):
            # compare two adjacent elements
            # change > to < to sort in descending order. For now it is ending order
            if number_list[num] > number_list[num+1]:
                # swapping elements if elements
                # are not in the intended order
                number_list[num + 1], number_list[num] = number_list[num], number_list[num+1]


def main():
    number_list = [3, 1, 9, 1, 5, 7, 2]
    bubble_sort(number_list)
    print("The Sorted list in ascending Order is ", number_list)


main()
```

And the output is :



Optimized Bubble Sort:

If the given list is already sorted, comparing all values is a waste of time and resources. Optimizing the bubble sort helps us to avoid unnecessary iterations and save time and resources.

Optimization is done using the following steps:

Step 1: Create variables that monitors if any swapping has occurred or not

Step 2: If the values have swapped positions, continue to the next iteration

Step 3: If the benefits have not swapped positions, terminate the inner loop, and continue with the outer loop.

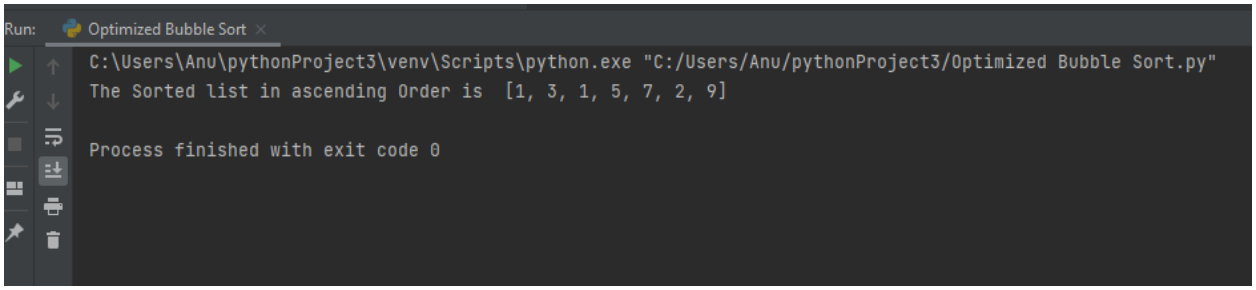
Here is the Python program for doing optimized bubble sort in ascending order:

```python
def bubble_sort(number_list):
    # loop through each element of number_list
    for i in range(len(number_list)):
        # keep track of swapping
        swapped = 0
        # loop to access each number_list element
        for num in range(0, len(number_list) - i - 1):
            # compare two adjacent elements
            # change > to < to sort in descending order. For now it is ending order
            if number_list[num] > number_list[num+1]:
                # swapping elements if elements
                # are not in the intended order
                number_list[num + 1], number_list[num] = number_list[num], number_list[num+1]
                swapped = 1
        # no swapping means the array is already sorted
        if not swapped == 0:
            break


def main():
    number_list = [3, 1, 9, 1, 5, 7, 2]
    bubble_sort(number_list)
    print("The Sorted list in ascending Order is ", number_list)


main()
```

Output:



An optimized bubble sort is more efficient as it only executes the necessary steps and skips those that are not required.

Bubble sort advantages

The following are some of the advantages of the bubble sort algorithm:

- Easy to understand
- It does not require extensive memory.
- It is easy to write the code for the algorithm.
- The space requirements are minimal compared to other sorting algorithms.

Bubble sort Disadvantages

The following are some of the disadvantages of the bubble sort algorithm:

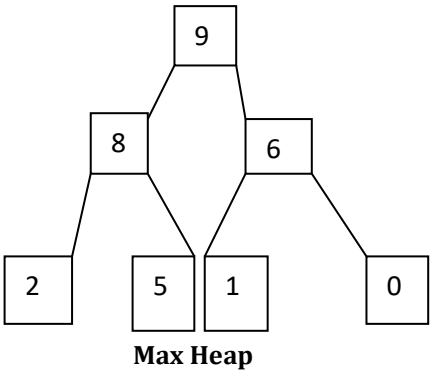
- It does not perform well when sorting large lists.
- It takes too much time and resources.
- It's mostly used for academic purposes and not the real-world application.
- The number of steps required to sort the list is of the order n^2.

HEAP SORT

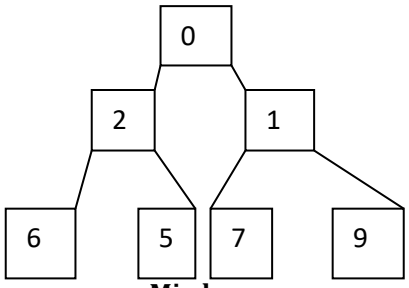
Heap Sort is a popular and efficient sorting algorithm in computer programming. Heap sort works by visualizing the elements of the array as a special kind of complete binary tree called a heap.

Working of Bubble Sort:

A complete binary tree has an interesting property that we can use to find the children and parents of any node.



Max Heap



Min heap

- Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
- Swap: Remove the root element and put at the end of the array (nth position) Put the last item of the tree (heap) at the vacant place.
- Remove: Reduce the size of the heap by 1.
- Heapify: Heapify the root element again so that we have the highest element at root.
- The process is repeated until all the items of the list are sorted.

Algorithm

Step1: Start

Step 2: Get the list number of items to be sorted and get the length.

Step 3: Create a max heap and pass list, length and index.

Step4: Create pointers for parent(i) and child node(for left= 2*i+1, for right= 2*i+2).

Step5: Check if left and right child of root exist and greater than root. And if child is greater than parent set greater value to parent .

Step 6: If parent is change then do swapping between parent index and largest index. And call max heap

Step7: Swap last element and root and call max heap.

Step 8: Repeat step 4, 5, 6 and 7 until we get the sorted list

Step 9: Print result.

Step 10: Stop

Program for doing Max heap:

```
def heapify(number_list, length, i):
    largest = i  # Initialize largest as root
    left = 2 * i + 1
    right = 2 * i + 2

    # Checking if left child of root exists
    #  and greater than root
    if left < length and number_list[i] < number_list[left]:
        largest = left

    # Checking if right child of root exists and
    # greater than root
    if right < length and number_list[largest] < number_list[right]:
        largest = right

    # Change root, if needed
    if largest != i:
        (number_list[i], number_list[largest]) = (number_list[largest], number_list[i])  # swap
        heapify(number_list, length, largest)  # Heapify the root.


# The main function to do heap sort the list
def heap_sort(number_list):
    length = len(number_list)

    # Build a Max Heap.
    for i in range(length // 2 - 1, -1, -1):
        heapify(number_list, length, i)
```

```
    # One by one extract elements
    for i in range(length - 1, 0, -1):
        (number_list[i], number_list[0]) = (number_list[0], number_list[i])  # swap
        heapify(number_list, i, 0)


def main():
    number_list = [3, 1, 9, 1, 5, 7, 2]
    heap_sort(number_list)
    length = len(number_list)
    sorted_list = []
    for i in range(length):
        sorted_list.append(number_list[i])
    print('Heap Sorted list is', sorted_list)


main()
```

Output:

Comparison between Bubble sort and Heap sort:

Bubble Sort	Heap Sort
Compare adjacent elements pairs and swaps if they are not in the intended order.	Proceed by heapify , swap and insert
Used to sort small data set.	Used to sort big data set
Not efficient	More efficient
Slower	Faster
Extremely efficient in terms of memory usage	Memory usage is minimal
Simple code	Complex than bubble sort