# Bubble sort in Python

def bubbleSort(array):

# loop to access each array element

for i in range(len(array)):

# loop to compare array elements

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

# compare two adjacent elements

# change > to < to sort in descending order

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

# swapping elements if elements

# are not in the intended order

temp = array[j]

array[j] = array[j+1]

array[j+1] = temp

data = [-2, 45, 0, 11, -9]

bubbleSort(data)

print('Sorted Array in Ascending Order:')

print(data)

OUTPUT:

Sorted Array in Ascending Order:

[-9, -2, 0, 11, 45]

**Insertion Sort**

[**https://www.youtube.com/watch?v=yCxV0kBpA6M**](https://www.youtube.com/watch?v=yCxV0kBpA6M)

**1. Time Complexity**

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n) |
| **Average Case** | O(n2) |
| **Worst Case** | O(n2) |

* **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of insertion sort is **O(n)**.
* **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of insertion sort is **O(n2)**.
* **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of insertion sort is **O(n2)**.

**2. Space Complexity**

|  |  |
| --- | --- |
| **Space Complexity** | O(1) |
| **Stable** | YES |

* The space complexity of insertion sort is O(1). It is because, in insertion sort, an extra variable is required for swapping.

### 3. Algorithm

insertionSort(array)

mark first element as sorted

for each unsorted element X

'extract' the element X

for j <- lastSortedIndex down to 0

if current element j > X

move sorted element to the right by 1

break loop and insert X here

end insertionSort

**# Insertion sort program in Python**

def insertionSort(array):

#range starts from 1 since position 0 is considered fixed in 1st iteration

for step in range(1, len(array)):

temp = array[step]

j = step - 1

# Compare temp with each element on the left of it until an element smaller than it is found

# For descending order, change temp<array[j] to temp>array[j].

while j >= 0 and temp < array[j]:

array[j + 1] = array[j]

j = j - 1

# Place temp at after the element just smaller than it.

array[j + 1] = temp

data = [9, 5, 1, 4, 3]

insertionSort(data)

print('Sorted Array in Ascending Order:')

print(data)

OUTPUT:

Sorted Array in Ascending Order:

[1, 3, 4, 5, 9]

**Merge Sort**

[**https://www.youtube.com/watch?v=jlHkDBEumP0**](https://www.youtube.com/watch?v=jlHkDBEumP0)

# MergeSort in Python

def mergeSort(array):

if len(array) > 1:

# r is the point where the array is divided into two subarrays

# // gives floor value

r = len(array)//2

L = array[:r]

# in L from start to r

M = array[r:]

# in M after r to end

# Sort the two halves

mergeSort(L)

mergeSort(M)

i = j = k = 0

# Until we reach either end of either L or M, pick larger among

# elements L and M and place them in the correct position at A[p..r]

while i < len(L) and j < len(M):

if L[i] < M[j]:

array[k] = L[i]

i =i+1

else:

array[k] = M[j]

j =j+1

k =k+1

# When we run out of elements in either L or M,

# pick up the remaining elements and put in A[p..r]

while i < len(L):

array[k] = L[i]

i =i+1

k =k+1

while j < len(M):

array[k] = M[j]

j =j+1

k =k+1

# Print the array

def printList(array):

for i in range(len(array)):

print(array[i], end=" ")

print()

array = [6, 5, 12, 10, 9, 1]

#here length of array is 6

mergeSort(array)

print("Sorted array is: ")

printList(array)

**OUTPUT:**

Sorted array is:

1 5 6 9 10 12

**Algorithm**

MergeSort(A, l, h):

if l > h

return

mid = (l+h)/2

mergeSort(A, l, mid)

mergeSort(A, mid+1, h)

merge(A, l, mid, h)

**Heap Sort**

[**https://www.youtube.com/watch?v=Q\_eia3jC9Ts**](https://www.youtube.com/watch?v=Q_eia3jC9Ts)

# Heap Sort in python

def heapify(arr, n, i):

# Find largest among root and children

largest = i

l = 2 \* i + 1

r = 2 \* i + 2

if l < n and arr[l] > arr[largest]:

largest = l

if r < n and arr[largest] < arr[r]:

largest = r

# If root is not largest, swap with largest and continue heapifying

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

heapify(arr, n, largest)

def heapSort(arr):

n = len(arr)

# Build max heap

for i in range(n//2, -1, -1):

heapify(arr, n, i)

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

# Swap and above for loop is for deleting

arr[i], arr[0] = arr[0], arr[i]

# Heapify root element

heapify(arr, i, 0)

arr = [1, 12, 9, 5, 6, 10]

n = len(arr)

heapSort(arr)

n = len(arr)

print("Sorted array is")

for i in range(n):

print("%d " % arr[i], end='')