Sorting Algorithms

Bubble Sort

Concept: Repeatedly swapping adjacent elements if they are in the wrong order.

Initial array: 5, 4, 3, 2, 1

**5 4** 3 2 1 -> 4 5 3 2 1

4 **5 3** 2 1 -> 4 3 5 2 1

4 3 **5 2** 1 -> 4 3 2 5 1

4 3 2 **5 1** -> 4 3 2 1 5 (Last i element is in place)

**4 3** 2 1 5 -> 3 4 2 1 5

3 **4 2** 1 5 -> 3 2 4 1 5

3 2 **4 1** 5 -> 3 2 1 4 5

……

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pseudo Code:

FOR i=0 to Size -1

FOR j=0 to Size - i - 1

IF arr[j] > arr[j+1]

Swap arr[j] and arr[j+1]

ENDIF

j++

i++

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Space Complexity: O(1) in terms of auxiliary array | O(n) in terms of memory.

Time Complexity: O(n^2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Selection Sort

Concept: Repeatedly find the minimum value and put it in the i-th place.

Initial array: 40, 35, 12, 20, 5

40 35 12 20 5 -> 5 40 35 12 20

5 40 35 12 20 -> 5 12 40 35 20

5 12 40 35 20 -> 5 12 20 40 35

5 12 20 40 35 -> 5 12 20 35 40

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pseudo Code:

FOR i = 0 to Size-1

min = Array[i]

index = i

FOR j = i + 1 to Size-1

If Array[j] < min

min= Array[j]

index = j

ENDIF

ENDFOR

Swap Array[i] and Array[index]

ENDFOR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Space Complexity: O(1) in terms of auxiliary array | O(n) in terms of memory.

Time Complexity: O(n^2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Insertion Sort

Concept: Repeatedly comparing the current element to all the other elements on its left and swapping them if they’re smaller.

Initial array: 6, 5, 3, 8, 1

6 5 3 8 1 -> 6 5 3 8 1

6 5 3 8 1 -> 5 6 3 8 1

5 6 3 8 1 -> 3 5 6 8 1

3 5 6 8 1 -> 3 5 6 8 1

3 5 6 8 1 -> 1 3 5 6 8

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pseudo Code:

FOR i =0 to N-1

j=i-1

temp = Array[i]

WHILE j>0 and Array[j]>temp

Array[j+1] = Array[j]

j- -

ENDWHILE

Array[j+1]=temp

ENDFOR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Space Complexity: O(1) in terms of auxiliary array | O(n) in terms of memory.

Time Complexity: O(n^2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Merge Sort

Concept: It is a Divide and Conquer algorithm.  It divides the unsorted array in two halves, calls itself for the two halves and then merges the two sorted halves.

Initial array: 2, 7, 3, 9, 12, 10

2 7 3 9 12 10

2 7 3 9 12 10

2 7 3 9 12 10

2 7 3 9 12 10l

2 7 3 9 10 12l

2 3 7 9l 10 12l

2 3 7 9 10 12l

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pseudo Code:

MergeSort (Array,Start,End):

IF (Start>=End)

Return

Mid = (Start+End)/2

MergeSort (Array,Start,Mid)

MergeSort (Array,Mid+1,End)

Merge(Array, Start, Mid, End)

Merge(Array, Start, Mid, End):

i=Start , j=Mid+1

WHILE i<=Mid and j<=End

If (Array[i] < Array[j])

Array.append(A[i++])

ELSE

Array.append(A[j++])

END WHILE

WHILE i<=Mid

Array.append(A[i++])

END WHILE

WHILE j<=End

Array.append(A[j++])

END WHILE

FOR i=0 to END

A[Start++]=Array[i]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Space Complexity: O(n) in terms of auxiliary array [Not In Place] .

Time Complexity: O(n Log n).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Quick Sort

Concept: It is a Divide and Conquer algorithm. It picks an element as pivot and partitions the array around the picked pivot (Smaller values on the left and larger values on the right).  
Here, The Pivot is picked randomly.

Initial array: 6, 5, 1, 3, 8, 4, 7, 9, 2

6 5 1 3 8 4 7 9 2

6 5 1 3 8 4 7 9 2

6 5 1 3 8 4 7 9 2

1 5 6 3 8 4 7 9 2

1 2 6 3 8 4 7 9 5

1 2 6 3 8 4 7 9 5

1 2 6 3 8 4 7 9 5

1 2 3 6 8 4 7 9 5

1 2 3 6 8 4 7 9 5

1 2 3 6 8 4 7 9 5

1 2 3 4 8 6 7 9 5

1 2 3 4 5 6 7 9 8

…..

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pseudo Code:

Partition(low,high):

Pivot=Array[high]

Index=low-1

For i=low to high

IF Array[i]<low

Swap(Array[i],Array[Index]

END IF

END FOR

Swap(Array[Index+1],Array[high])

Return Index+1

Quicksort(low,high):

IF low<high

Pivot=(low+high)/2

Swap(Array[high,Array[Pivot])

PivotIndex=Partition(low,high)

Quicksort(low,PivotIndex-1)

Quicksort(PivotIndex+1,high)

END IF

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Space Complexity: O(1) in terms of auxiliary array [In Place] .

Time Complexity: O(n Log n) [Average] , O(n^2) [Worst].



