Name :Mohammad Nael Shrateh

Number : 1141258

Dr .Iyad Jaber

5-sorting type

1- comb sort

The   **Comb Sort**   is a variant of the [Bubble Sort](https://rosettacode.org/wiki/Bubble_Sort).

Like the [Shell sort](https://rosettacode.org/wiki/Shell_sort), the Comb Sort increases the gap used in comparisons and exchanges (dividing the gap by {\displaystyle (1-e^{-\varphi })^{-1}\approx 1.247330950103979} works best, but 1.3 may be more practical).

Best time = O(n) when the array is sorted

Worst case and average case =O(n2)

sort**(**E**[]** input**)** **{**  
 **int** gap = input.**length**;  
 **boolean** swapped = **true**;  
 **while** **(**gap > **1** && swapped**)** **{**  
 **if** **(**gap > **1)** **{**  
 gap = **(int)** **(**gap / **1.3)**;  
 **}**  
 swapped = **false**;  
 **for** **(int** i = **0**; i + gap < input.**length**; i++**)** **{**  
 **if** **(**input**[**i**]**.**compareTo(**input**[**i + gap**])** > **0)** **{**  
 E t = input**[**i**]**;  
 input**[**i**]** = input**[**i + gap**]**;  
 input**[**i + gap**]** = t;  
 swapped = **true**;  
 **}**  
 **}**  
 **}**  
**}**

### 2- Odd-Even Transposition Sort:

Odd-Even Transposition Sort is a parallel sorting algorithm. It is based on the Bubble Sort technique of comparing two numbers and switching them if the first is greater than the second, to achieve a left to right ascending ordering. So what if each number were a PE, (Processing Element)? Each number would look to it's right neighbor and if it were greater, it would switch them. If every PE is doing this in parallel, there is going to be chaos because there will be multiple writes of different data to the same memory location.

Best time = O(n) when the array is sorted

Worst case and average case =O(n2)

**function** oddEvenSort(list) {

**function** swap( list, i, j ){

**var** temp = list[i];

list[i] = list[j];

list[j] = temp;

}

**var** sorted = **false**;

**while**(!sorted)

{

sorted = **true**;

**for**(**var** i = 1; i < list.length-1; i += 2)

{

**if**(list[i] > list[i+1])

{

swap(list, i, i+1);

sorted = **false**;

}

}

**for**(**var** i = 0; i < list.length-1; i += 2)

{

**if**(list[i] > list[i+1])

{

swap(list, i, i+1);

sorted = **false**;

}

}

}

3- cocktail sort

The cocktail shaker sort is an improvement on the [Bubble Sort](https://rosettacode.org/wiki/Bubble_Sort). The improvement is basically that values "bubble" both directions through the array, because on each iteration the cocktail shaker sort bubble sorts once forwards and once backwards

Best time = O(n) when the array is sorted

Worst case and average case =O(n2)

.

**public** **static** **void** cocktailSort**(** **int[]** A **){**  
 **boolean** swapped;  
 **do** **{**  
 swapped = **false**;  
 **for** **(int** i =**0**; i<= A.**length** - **2**;i++**)** **{**  
 **if** **(**A**[** i **]** > A**[** i + **1** **])** **{**  
 *//test whether the two elements are in the wrong order*  
 **int** temp = A**[**i**]**;  
 A**[**i**]** = A**[**i+**1]**;  
 A**[**i+**1]**=temp;  
 swapped = **true**;  
 **}**  
 **}**  
 **if** **(**!swapped**)** **{**  
 *//we can exit the outer loop here if no swaps occurred.*  
 **break**;  
 **}**  
 swapped = **false**;  
 **for** **(int** i= A.**length** - **2**;i>=**0**;i--**)** **{**  
 **if** **(**A**[** i **]** > A**[** i + **1** **])** **{**  
 **int** temp = A**[**i**]**;  
 A**[**i**]** = A**[**i+**1]**;  
 A**[**i+**1]**=temp;  
 swapped = **true**;  
 **}**  
 **}**  
 *//if no elements have been swapped, then the list is sorted*  
 **}** **while** **(**swapped**)**;  
**}**

4- Gnome sort

It is a [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm) which is similar to [insertion sort](https://en.wikipedia.org/wiki/Insertion_sort), except that moving an element to its proper place is accomplished by a series of swaps, as in [bubble sort](https://en.wikipedia.org/wiki/Bubble_sort). It is conceptually simple, requiring no nested loops. The average, or expected, running time is [*O*](https://en.wikipedia.org/wiki/Big_O_notation)(*n*2), but tends towards *O*(*n*) if the list is initially almost sorted

The algorithm always finds the first place where two adjacent elements are in the wrong order, and swaps them. It takes advantage of the fact that performing a swap can introduce a new out-of-order adjacent pair only next to the two swapped elements. It does not assume that elements forward of the current position are sorted, so it only needs to check the position directly previous to the swapped elements.

**public** **class** **GnomeSort** {

**static** void gnomeSort( int[] theArray ) {

**for** ( int index = 1; index < theArray.length; ) {

**if** ( theArray[index - 1] <= theArray[index] ) {

++index;

} **else** {

int tempVal = theArray[index];

theArray[index] = theArray[index - 1];

theArray[index - 1] = tempVal;

--index;

**if** ( index == 0 ) {

index = 1;

}

}

}

}

5- Bucket sort

works by partitioning an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm.

Worst Case O(n2)

Average Case O(n)

1. sort(**int**[] sequence, **int** maxValue)
2. {
3. *// Bucket Sort*
4. **int**[] Bucket = **new** **int**[maxValue + 1];
5. **int**[] sorted\_sequence = **new** **int**[sequence.length];
7. **for** (**int** i = 0; i < sequence.length; i++)
8. Bucket[sequence[i]]++;
10. **int** outPos = 0;
11. **for** (**int** i = 0; i < Bucket.length; i++)
12. **for** (**int** j = 0; j < Bucket[i]; j++)
13. sorted\_sequence[outPos++] = i;
15. **return** sorted\_sequence;
16. }