**Shell Sort**

La clasificación Shell es principalmente una variación de la clasificación por inserción . En la ordenación por inserción, movemos los elementos solo una posición hacia adelante. Cuando es necesario mover un elemento mucho más adelante, intervienen muchos movimientos. La idea de Shell Sort es permitir el intercambio de artículos lejanos. En Shell sort, ordenamos la matriz h para un valor grande de h. Seguimos reduciendo el valor de h hasta que se convierte en 1. Se dice que una matriz está ordenada en h si todas las sublistas de cada h-ésimo elemento están ordenadas.

**Algoritmo:**

Paso 1 - Iniciar  
Paso 2 - Inicialice el valor del tamaño del espacio. Ejemplo: h  
Paso 3 : divida la lista en subpartes más pequeñas. Cada uno debe tener intervalos iguales a h  
Paso 4: ordene estas sublistas mediante ordenación por inserción.  
Paso 5: repita este paso 2 hasta que la lista esté ordenada.  
Paso 6: imprima una lista ordenada.  
Paso 7 – Detente.

**Pseudocódigo:**

PROCEDIMIENTO SHELL\_SORT(ARRAY, N)   
 WHILE GAP < LONGITUD(ARRAY) /3 :  
 GAP = ( INTERVAL \* 3 ) + 1   
 FINAL WHILE LOOP  
 WHILE GAP > 0 :  
 FOR (EXTERNO = GAP; EXTERIOR < LONGITUD(ARRAY); EXTERNO++):  
 INSERTION\_VALUE = ARRAY[EXTERIOR]  
 INTERIOR = EXTERIOR;  
 WHILE INNER > GAP-1 AND ARRAY[INNER – GAP] >= INSERTION\_VALUE:  
 ARRAY[INNER] = ARRAY[INNER – GAP]  
 INNER = INNER – GAP  
 END WHILE LOOP  
 ARRAY[INNER] = INSERTION\_VALUE  
 END FOR LOOP  
 GAP = (GAP - 1) /3;   
 FINALIZAR EL BUCLE MIENTRAS  
FINALIZAR SHELL\_SORT

A continuación se muestra la implementación de ShellSort.

// C++ implementation of Shell Sort

#include <iostream>

**using** **namespace** std;

/\* function to sort arr using shellSort \*/

**int** shellSort(**int** arr[], **int** n)

{

// Start with a big gap, then reduce the gap

**for** (**int** gap = n/2; gap > 0; gap /= 2)

{

// Do a gapped insertion sort for this gap size.

// The first gap elements a[0..gap-1] are already in gapped order

// keep adding one more element until the entire array is

// gap sorted

**for** (**int** i = gap; i < n; i += 1)

{

// add a[i] to the elements that have been gap sorted

// save a[i] in temp and make a hole at position i

**int** temp = arr[i];

// shift earlier gap-sorted elements up until the correct

// location for a[i] is found

**int** j;

**for** (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

arr[j] = arr[j - gap];

// put temp (the original a[i]) in its correct location

arr[j] = temp;

}

}

**return** 0;

}

**void** printArray(**int** arr[], **int** n)

{

**for** (**int** i=0; i<n; i++)

cout << arr[i] << " ";

}

**int** main()

{

**int** arr[] = {12, 34, 54, 2, 3}, i;

**int** n = **sizeof**(arr)/**sizeof**(arr[0]);

cout << "Array before sorting: \n";

printArray(arr, n);

shellSort(arr, n);

cout << "\nArray after sorting: \n";

printArray(arr, n);

**return** 0;

}

**Conclusiones**

En resumen, Shell Sort es una variante eficiente de la clasificación por inserción que permite el intercambio de elementos distantes, mejorando significativamente el rendimiento del algoritmo. Su enfoque de dividir la lista en sublistas y ordenarlas gradualmente resulta en una ordenación más rápida y eficiente en comparación con la clasificación por inserción tradicional.

**Referencia**

Jain, S. (2023, January 10). *ShellSort*. GeeksforGeeks. Retrieved February 4, 2024, from https://www.geeksforgeeks.org/shellsort/