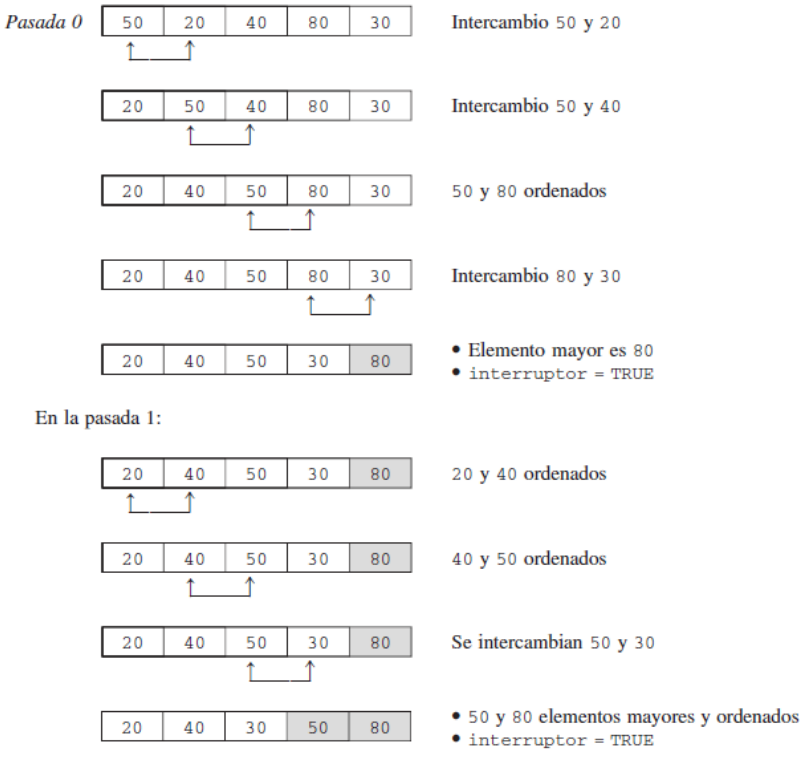
Suarez Vázquez Brandon Abraham 17211566

Algoritmos de ordenamiento interno:

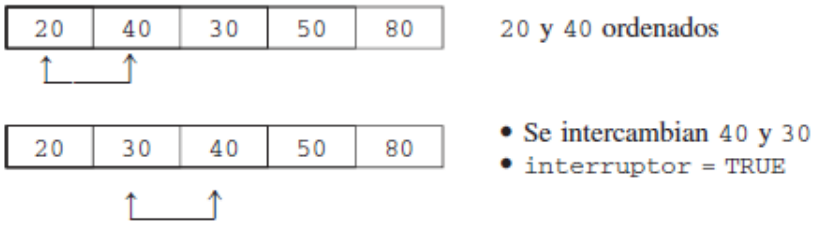
Ordenamiento por Burbuja:

El método de *ordenación por burbuja* es el más conocido y popular entre estudiantes y aprendices de programación, por su facilidad de comprensión y programación; por el contrario, es el menos eficiente y por ello, normalmente, se aprende su técnica, pero no suele utilizarse. La técnica utilizada se denomina ***ordenación por burbuja*** u ***ordenación por hundimiento*** debido a que los valores más pequeños «burbujean» gradualmente (suben) hacia la cima o parte superior del array de modo similar a como suben las burbujas en el agua, mientras que los valores mayores se hunden en la parte inferior del array. La técnica consiste en hacer varias pasadas a través del array. En cada pasada, se comparan parejas sucesivas de elementos. Si una pareja está en orden creciente (o los valores son idénticos), se dejan los valores como están. Si una pareja está en orden decreciente, sus valores se intercambian en el array.

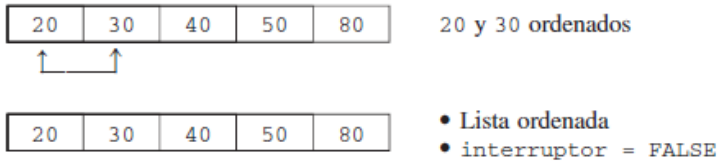
En el caso de un array (lista) con n elementos, la ordenación por burbuja requiere hasta $n - 1$ pasadas. Por cada pasada se comparan elementos adyacentes y se intercambian sus valores cuando el primer elemento es mayor que el segundo elemento. Al final de cada pasada, el elemento mayor ha «burbujeado» hasta la cima de la sublista actual. Por ejemplo, después que la pasada 0 está completa, la cola de la lista $A[n - 1]$ está ordenada y el frente de la lista permanece desordenado.



En la pasada 2, sólo se hacen dos comparaciones:



En la pasada 3, se hace una única comparación de 20 y 30, y no se produce intercambio:



En consecuencia, el algoritmo de ordenación de burbuja mejorado contempla dos bucles anidados: el *bucle externo* controla la cantidad de pasadas (al principio de la primera pasada todavía no se ha producido ningún intercambio, por tanto la variable interruptor se pone a *valor falso* (0); el *bucle interno* controla cada pasada individualmente y cuando se produce un intercambio, cambia el valor de interruptor a *verdadero* (1). El algoritmo terminará, bien cuando se termine la última pasada (*n* – 1) o bien cuando el valor del interruptor sea falso (0), es decir, no se haya hecho ningún intercambio. La condición para realizar una nueva pasada se define en la expresión lógica (pasada < n-1) && interruptor

Ordenamiento por ShellSort:

La **ordenación Shell** debe el nombre a su inventor, D. L. Shell. Se suele denominar también *ordenación por inserción con incrementos decrecientes*. Se considera que el método Shell es una mejora de los métodos de inserción directa. En el álgoritmo de inserción, cada elemento se compara con los elementos contiguos de su izquierda, uno tras otro. Si el elemento a insertar es el más pequeño hay que realizar muchas comparaciones antes de colocarlo en su lugar definitivo. El algoritmo de Shell modifica los saltos contiguos resultantes de las comparaciones por saltos de mayor tamaño y con ello se consigue que la ordenación sea más rápida. Generalmente se toma como salto inicial *n*/2 (siendo *n* el número de elementos), luego se reduce el salto a la mitad en cada repetición hasta que el salto es de tamaño 1.

6 1 5 2 3 4 0

El número de elementos que tiene la lista es 6, por lo que el salto inicial es 6/2 = 3. La siguiente tabla muestra el número de recorridos realizados en la lista con los saltos correspondiente.

Recorrido	*Salto*	*Intercambios*	*Lista*
1	3	(6,2),(5,4),(6,0)	2 1 4 0 3 5 6
2	3	(2, 0)	0 1 4 2 3 5 6
3	3	*ninguno*	0 1 4 2 3 5 6
salto 3/2 = 1			
4	1	(4,2),(4,3)	0 1 2 3 4 5 6
5	1	*ninguno*	0 1 2 3 4 5 6

Los pasos a seguir por el algoritmo para una lista de n elementos son:
1. Dividir la lista original en n/2 grupos de dos, considerando un incremento o salto entre los elementos de n/2.
2. Clarificar cada grupo por separado, comparando las parejas de elementos, y si no están ordenados, se intercambian.
3. Se divide ahora la lista en la mitad de grupos (n/4), con un incremento o salto entre los elementos también mitad (n/4), y nuevamente se clasifica cada grupo por separado.
4. Así sucesivamente, se sigue dividiendo la lista en la mitad de grupos que en el recorrido anterior con un incremento o salto decreciente en la mitad que el salto anterior, y luego clasificando cada grupo por separado.
5. El algoritmo termina cuando se consigue que el tamaño del salto es 1.

Ordenación por Quicksort:

El algoritmo conocido como *quicksort* (ordenación rápida) recibe el nombre de su autor, Tony Hoare. La idea del algoritmo es simple, se basa en la división en particiones de la lista a ordenar, por lo que se puede considerar que aplica la técnica *divide y vencerás*. El método es, posiblemente, el más pequeño de código, más rápido, más elegante, más interesante y eficiente de los algoritmos de ordenación conocidos. El método se basa en dividir los *n* elementos de la lista a ordenar en dos partes particiones separadas por un elemento: una *partición izquierda*, un elemento *central* denominado *pivote* o elemento de partición, y una *partición derecha*. La partición o división se hace de tal forma que todos los elementos de la primera sublista (partición izquierda) son menores que todos los elementos

de la segunda sublista (partición derecha). Las dos sublistas se ordenan entonces independientemente. Para dividir la lista en particiones (*sublistas*) se elige uno de los elementos de la lista y se utiliza como *pivote* o *elemento de partición*. Si se elige una lista cualquiera con los elementos en orden aleatorio, se puede seleccionar cualquier elemento de la lista como pivote, por ejemplo, el primer elemento de la lista. Si la lista tiene algún orden parcial conocido, se puede tomar otra
decisión para el pivote. Idealmente, el pivote se debe elegir de modo que se divida la lista exactamente por la mitad, de acuerdo al tamaño relativo de las claves. Por ejemplo, si se tiene una lista de enteros de 1 a 10, 5 o 6 serían pivotes ideales, mientras que 1 o 10 serían elecciones «pobres» de pivotes. Una vez que el pivote ha sido elegido, se utiliza para ordenar el resto de la lista en dos sublistas: una tiene todas las claves menores que el pivote y la otra, todos los elementos (claves) mayores que o iguales que el pivote (o al revés). Estas dos listas parciales se ordenan recursivamente utilizando el mismo algoritmo; es decir, se llama sucesivamente al propio algoritmo *quicksort*. La lista final ordenada se consigue concatenando la primera sublista, el pivote y la segunda lista, en ese orden, en una única lista. La primera etapa de *quicksort* es la división o «particionado» recursivo de la lista
hasta que todas las sublistas constan de sólo un elemento.
Se ordena una lista de números enteros aplicando el algoritmo quicksort, como pivote se elige el primer elemento de la lista.