James Quintero

ShiftSort Sorting Algorithm

Shift Sort is a merging algorithm like Merge Sort, but is more selective on what it merges. Merge Sort splits its array in half continuously until reaching its base case of 2 elements, swaps if needed, and then merges as it returns. Shift Sort uses a derivative array to split in half continuously until reaching a base case of 2 or 3 elements, uses the results to determine what parts of the array to merge, and then merges as it returns.

Shift Sort time and space complexities:

Best	O(n)
Average	O(nlogn)
Worst	O(nlogn)
Space	n

With best-case complexity, Shift Sort's O(n) beats Merge Sort's O(nlogn). Shift Sort's average-case complexity is the same as Merge Sort's, but in real world testing, Shift Sort out performs Merge Sort. For space, at worst, Shift Sort will initialize a second and third list of about n/2, putting the total at n.

Shift Sort:

Starting array:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	31	27	13

Shift Sort first traverses the array from back to front.

For visualization, there will be a second list ("zeroes") of 1s and 0s where a 0 indicates the start of a sorted sublist, and 1 indicating an element of a sorted sublist

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	31	27	13
																				x

If array[x] into is less than the element at array[x-1] index, mark a 0. Otherwise, mark 1 and decrement x.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	31	27	13
																				x
																				0

If array[x-1] < array[x-2], then this indicates a sublist of length 3 that's in descending order. Swap array[x] with array[x-2] to put this sublist in ascending order.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	13	27	31
																				x
																				1

Mark array[x] and array[x-1] as 1 since they're now in sorted order. Check that array[x+1] didn't get out of order. If it did, mark array[x+1] as 0. Decrement x by 2.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	13	27	31
																		x		
																			1	1

If array[x] into is less than the element at array[x-1] index, mark a 0.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	13	27	31
																		x		
																		0	1	1

Because array[x-1] > array[x-2], don't swap. Add x to a list of zeroes since it now indicates the start of a sorted sublist. Mark x-1 as a 1, and decrement x by 2.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	13	27	31
																x				
																	1	0	1	1

array[x] is less than array[x-1], so mark a 0.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	31	27	13
																x				
																0	1	0	1	1

array[x-1] is not less than array[x-2], so add x to the list of zeroes, mark x-1 as 1, and decrement x by 2.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	13	27	31
														x						
															1	0	1	0	1	1

array[x]>array[x-1], so mark as 1 and decrement x.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	95	17	23	15	48	32	13	72	50	30	42	49	16	30	48	10	27	13	27	31
													x							
														1	1	0	1	0	1	1

… Continue until x=0

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	32	48	30	50	72	42	49	16	30	48	10	27	13	27	31
	x																			
		1	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	0	1	1

Ones indicate that those elements are in proper order in respect to their immediate left element, and zeroes indicate that those elements are not in proper order in respect to their immediate left element. This also means that indices with a 0 are starting indices to sorted sublists. The sublists end once their reach another 0 or the end of the array. Because of this, the end of the array should be considered a starting index since it will indicate the ending of the last sorted sublist in the order. 0 will, by default, also be a sublist index.

List of indices of sorted sublists (now referred as the zeroes array):

20	17	15	12	10	7	3	0

If read from right to left, zeroes[0] will be the index of the start of a sublist in array, zeroes[1] would be the index of the start of a 2nd sublist and therefore the end of the first sublist, and zeroes[2] would be the end of the 2nd sublist (along with being the start of another sublist, and so on).

Unlike Merge Sort, Shift Sort will split zeroes instead of the actual array.

Start split of zeroes:

Level 1:

List 1:

index	0	1	2	3	4	5	6	7
zeroes	20	17	15	12	10	7	3	0
	i							j

List 1:

index	0	1	2	3	4	5	6	7
zeroes	20	17	15	12	10	7	3	0
	i			j2	i2			j

Level 2:

List 1:

Index	0	1	2	3
zeroes	20	17	15	12
	i			j

List 1:

index	0	1	2	3
zeroes	20	17	15	12
	i	j2	i2	j

Level 3:

index	0	1
zeroes	20	17
	i	j

Need 3 elements of zeroes in order to merge 2 lists, so do nothing and return.

index	2	3
zeroes	15	12
	i	j

Need 3 elements of zeroes in order to merge 2 lists, so do nothing and return.

Level 2:

List 1:

index	0	1	2	3
zeroes	20	17	15	12
	i	j2	i2	j

After splitting the zeroes array, Shift Sort starts the sublists. The sublist in array with starting and ending indices at zeroes[i2], zeroes[j2-1] will merge with the sublist in array with starting and ending indices at zeroes[j2], zeroes[i-1]. So array[15] to array[16] will be merged with array[17] to array[19].

Shift Sort merges by creating a copy of the first list, then traversing through both the copy of the first list and the second list. If the element in the second list is less than the element in the first list copy, the element is shifted to the left for its final resting place. This is where the name *Shift* sort was derived. This continues until the first list copy has been fully traversed.

Before:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	32	48	30	50	72	42	49	16	30	48	10	27	13	27	31
		1	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	0	1	1

After:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	32	48	30	50	72	42	49	16	30	48	10	13	27	27	31
		1	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

Result on zeroes:

index	0	1	2	3
zeroes	20		15	12
	i		i2	j

Shift Sort will then merge the sublist in array with starting and ending indices at zeroes[j], zeroes[i2-1] with the sublist in array with starting and ending indices at zeroes[i2], zeroes[i-1]. So array[12] to array[14] will be merged with array[15] to array[19].

Before:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	32	48	30	50	72	42	49	16	30	48	10	13	27	27	31
		1	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

After:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	32	48	30	50	72	42	49	10	13	16	27	27	30	31	48
		1	1	0	1	1	1	0	1	1	0	1	0	1	1	1	1	1	1	1

Result on zeroes:

index	0	1	2	3
zeroes	20			12
	i			j

Return to Level 1 where Shift Sort will continue recursion on the second half of zeroes.

Level 2:

List 2:

index	4	5	6	7
zeroes	10	7	3	0
	i			j

List 2:

index	4	5	6	7
zeroes	10	7	3	0
	i	j2	i2	j

Level 3:

List 2:

index	4	5
zeroes	10	7
	i	j

Need 3 elements of zeroes in order to merge 2 lists, so do nothing and return.

List 2:

index	6	7
zeroes	3	0
	i	j

Need 3 elements of zeroes in order to merge 2 lists, so do nothing and return.

Level 2:

List 2:

index	4	5	6	7
zeroes	10	7	3	0
	i	j2	i2	j

Like with the first half of the list, Shift Sort starts merging the sublists. The sublist in array with starting and ending indices at zeroes[i2], zeroes[j2-1] will merge with the sublist in array with starting and ending indices at zeroes[j2], zeroes[i-1]. So array[3] to array[6] will be merged with array[7] to array[9].

Before:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	32	48	30	50	72	42	49	10	13	16	27	27	30	31	48
		1	1	0	1	1	1	0	1	1	0	1	0	1	1	1	1	1	1	1

After:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	30	32	48	50	72	42	49	10	13	16	27	27	30	31	48
		1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1

Result on zeroes:

index	4	5	6	7
zeroes	10		3	0
	i		i2	j

Shift Sort will then merge the sublist in array with starting and ending indices at zeroes[j], zeroes[i2-1] with the sublist in array with starting and ending indices at zeroes[i2], zeroes[i-1]. So array[0] to array[2] will be merged with array[3] to array[9].

Before:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	30	32	48	50	72	42	49	10	13	16	27	27	30	31	48
		1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1

After:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	15	17	23	30	32	48	50	72	95	42	49	10	13	16	27	27	30	31	48
		1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1

Result on zeroes:

index	4	5	6	7
zeroes	10			0
	i	j2	i2	j

Return 1 level 1 where Shift Sort will have completed the split process and will start the merge process.

Zeroes:

index	0	1	2	3	4	5	6	7
zeroes	20			12	10			0
	i			j2	i2			j

Merging is just like with previous levels. array[i2] to array[j2-1] and array[j2] to array[i] will merge before array[j] to array[i2-1] and array[i2] to array[i] merges.

Before:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	13	17	95	15	23	30	32	48	50	72	42	49	10	13	16	27	27	30	31	48
		1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1

Final:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
array	10	13	13	15	16	17	23	27	27	30	30	31	32	42	48	48	49	50	72	95
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Shift Sort finishes execution with array now sorted.

10	13	13	15	16	17	23	27	27	30	30	31	32	42	48	48	49	50	72	95

James Quintero

Comparing Shift Sort to Merge Sort:

Language:	**C++**	**C++**
Sort algo:	Shift Sort	Merge Sort
Complexity:	Best	Best
Array len:	1,000,000	1,000,000
Runs	100	100
Avg run time:	2.0E6 ns	8.80E7 ns

Language:	**C++**	**C++**
Sort algo:	Shift Sort	Merge Sort
Complexity:	Average	Average
Array len:	1,000,000	1,000,000
Runs	100	100
Avg run time:	1.92E8 ns	1.97E8 ns

Language:	**Java**	**Java**
Sort algo:	Shift Sort	Merge Sort
Complexity:	Average	Average
Num tests:	100	100
Array len:	100,000	100,000
Runs per test:	100	100
Avg run time:	1.22E7 ns	1.26E7 ns