James Quintero

ShiftSort Sorting Algorithm

Shift Sort is a stable, adaptive, divide-and-conquer sorting algorithm. It's similar to 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

Shift Sort has a best-case time complexity of O(n) because it uses a secondary list, and if the secondary list is empty, the array is already sorted and the algorithm stops after n iterations.

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.

Main parts of Shift Sort:

1) Derivative List Creation
2) Splitting of Derivative list
3) Merging Sublists

1. Derivative List Creation:

Shift Sort starts at the last index of the array and traverses backwards until reaching the first index. While traversing, Shift Sort wants to find sublists that are already in ascending order, or make some itself. To do this, Shift Sort checks if array[x] > array[x-1]. If it's not, then it knows that the array[x] is probably part of a sublist in descending order. To make sure, Shift Sort checks if array[x-2] > array[x-1]. If true, then Shift Sort knows that it's now at the back of a length 3 descending order sublist; array[x-2] to array[x].

EX:

…	15	48	32	16	72	…

x

array[x] is 16, and array[x-1] is 32. array[x] < array[x-1], and array[x-1]<array[x-2], so Shift Sort knows that array[x-2] to array[x] is in descending order:

…	15	48	32	16	72	…
				x		

To make things easier in steps 2 and 3, Shift Sort swaps array[x] and array[x-2]. This turns the 3 length *descending* order sublist into a 3 length *ascending* order sublist.

…	15	16	32	48	72	…
				x		

Shift Sort continues traversing. If array[x]<array[x-1] but array[x-1]>=array[x-2], then just add x to the derivative list.

EX:

Index	0	1	2	3	4	5
array	95	17	23	15	48	…
				x		

… Continue until x=0.

Array before:

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	16	72	50	30	42	49	16	30	48	10	27	31	27	13

Array after:

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	16	32	48	30	50	72	42	49	16	30	48	10	27	13	27	31
		0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	0	1	1

1s indicate that those elements are in proper order in respect to their immediate left element, and 0s 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, and are therefore added to the derivative list. The sublists end once they reach another 0 or the end of the array. Because of this, the end of the array should be added to the derivative list since it will indicate the ending of the last sorted sublist in the array. 0 will, by default, also be added.

Derivative List:

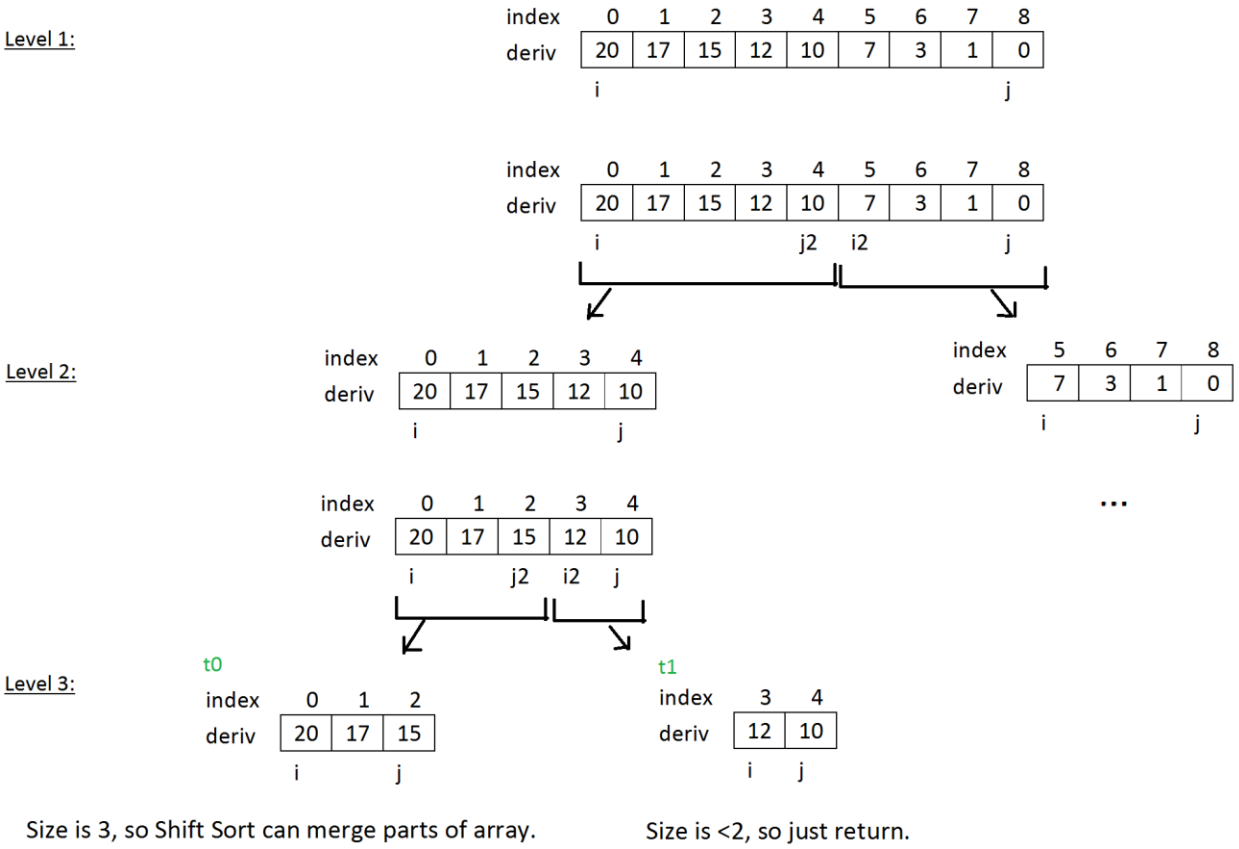
20	17	15	12	10	7	3	1	0

If read from right to left, deriv[0] will be the index of the start of a sublist in array, deriv[1] would be the index of the start of a 2nd sublist and therefore the end of the first sublist, and deriv[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 this derivative array instead of the actual array. This will save on dividing, and therefore time. The smaller the derivative list, the less recursion and merging, and the quicker Shift Sort is.

2. Splitting of Derivative List:

The derivative list will be split in half until a portion of the list is a base-case length of 2 or 3. (i,j) represent the left-most and right-most bounds of the portion of deriv being dealt with.



Size is 3, so Shift Sort can merge parts of array.

Size is <2, so just return.

For t1, deriv size is <2, so just return.

For t0, deriv size is 3, so Shift Sort can merge parts of array. 3 indices of array are given by deriv, and in this case, they are 15, 17, 20. The sublist from array[15] to array[17-1] is going to merge with the sublist from array[17] to array[20-1]. The resulting sublist will be sorted from array[15] to array[20-1], so resulting derivative list will be:

index	0	1	2
deriv	20		15
	i		j

Going back up the recursion tree, deriv will look like:

index	0	1	2	3	4
deriv	20		15	12	10
	i		j2	i2	j

As Shift Sort traverses back up, it merges the remaining sublists. In this case, the sublist in array with starting and ending indices at deriv[i2], deriv[j2]-1 will merge with the sublist in array with starting and ending indices at zeroes[j2], zeroes[i]-1. So array[12] to array[14] will be merged with array[15] to array[19]. *See Merging Sublists for merging technique*.

Deriv list after merging sublists in array:

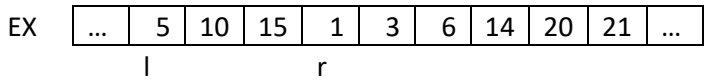
index	0	1	2	3	4
deriv	20			12	10
	i			i2	j

Shift Sort does the same with the remaining elements before returning. After a return, it's assumed that the portion of array between the two indices deriv[i] and deriv[j], is now sorted.

3. Merging Sublists

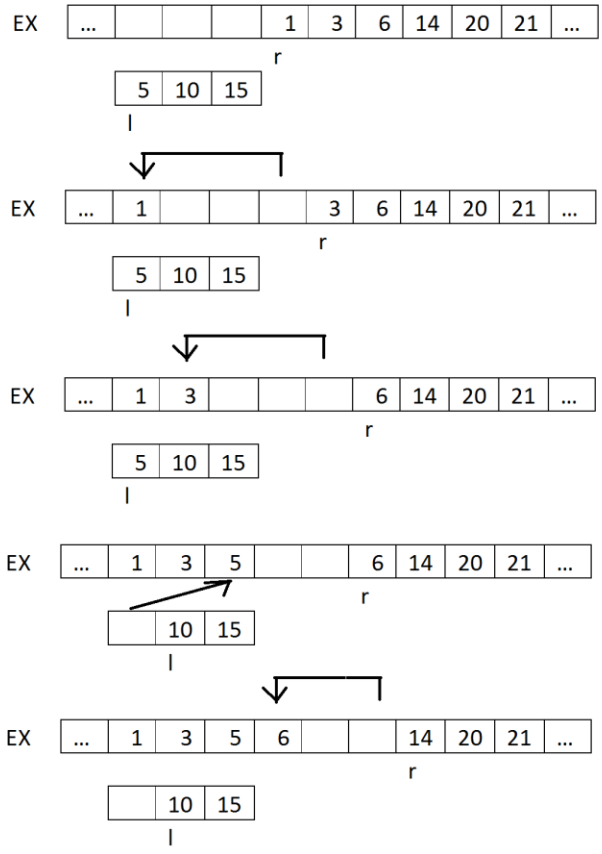
The in-place sorted sublists allows for easy merging by shifting. This is almost in-place merging. In a general sense, the merging is going to require a temporary array whose size is that of the smallest of the two sublists. If the 1st sublist is size 3 and 2nd sublist is size 10, Shift Sort will merge 1st sublist into 2nd sublist.

Example of merging 1st sublist into 2nd sublist:

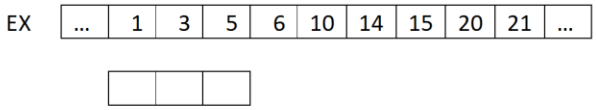


Temporary list is created of 1st sublist.

If the temporary array has the smaller element, then it gets moved into the current spot. If not, then the element already in the array is going to *shift* to the left into the appropriate spot.



Done merging once the temporary array is empty.



Finish:

Shift Sort finishes execution when there are only 2 remaining elements in the derivative array, specifically the 0 and N element. This signifies that there's one big sorted sublist in array that starts at index 0 and ends at index N-1; this is the entire array.

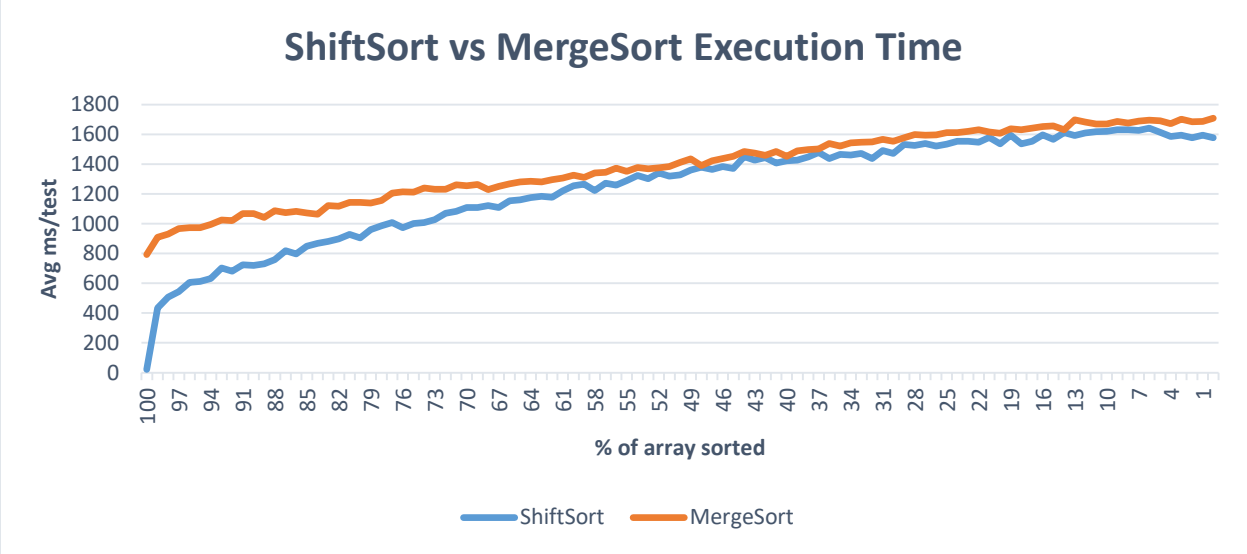
Final derivative list:

index	0	1	2	3	4	5	6	7	8
deriv	20								0

Final array:

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

Benchmarks were run between ShiftSort and MergeSort on 100,000 element arrays. These arrays ranged from being 100% sorted, to random. Because ShiftSort is adaptive, it performs better when portions of the array are already sorted. The absolute execution time doesn't matter, only the relation between Shift Sort's and Merge Sort's.



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

In both C++ and Java, Shift Sort slightly outperforms Merge Sort even though they have the same Big-O.