Assignment 6 - Sorting (Tracing Sorts)

Vishak Srikanth

October 11, 2021

1 Problem 1 - Elementary Sorts

1.1 Insertion Sort, Selection Sort, & Bubblesort

Trace after each outer loop for sorting the sequence $I, L, O, V, E, A, L, G, O, R, I, T, H, M, S$ using various elementary sorts.

a. Progress of Insertion Sort algorithm

Trace of Insertion sort algorithm with array contents just after each insertion (outer loop). (Note: entries in **red** represent **a[j]**, entries in grey are **elements that do not move**, and entries in **black get moved one position right for insertion**. **Input** and **output** array contents are shown in **blue**

i	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input:		**I**	**L**	**O**	**V**	**E**	**A**	**L**	**G**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
1	**1**	I	**L**	O	V	E	A	L	G	O	R	I	T	H	M	S
2	**2**	I	L	**O**	V	E	A	L	G	O	R	I	T	H	M	S
3	**3**	I	L	O	**V**	E	A	L	G	O	R	I	T	H	M	S
4	**0**	**E**	**I**	**L**	**O**	**V**	A	L	G	O	R	I	T	H	M	S
5	**0**	**A**	**E**	**I**	**L**	**O**	**V**	L	G	O	R	I	T	H	M	S
6	**4**	A	E	I	L	**L**	**O**	**V**	G	O	R	I	T	H	M	S
7	**2**	A	E	**G**	**I**	**L**	**L**	**O**	**V**	O	R	I	T	H	M	S
8	**7**	A	E	G	I	L	L	O	**O**	**V**	R	I	T	H	M	S
9	**8**	A	E	G	I	L	L	O	O	**R**	**V**	I	T	H	M	S
10	**4**	A	E	G	I	**I**	**L**	**L**	**O**	**O**	**R**	**V**	T	H	M	S
11	**10**	A	E	G	I	I	L	L	O	O	R	**T**	**V**	H	M	S
12	**3**	A	E	G	**H**	**I**	**I**	**L**	**L**	**O**	**O**	**R**	**T**	**V**	M	S
13	**8**	A	E	G	H	I	I	L	L	**M**	**O**	**O**	**R**	**T**	**V**	S
14	**12**	A	E	G	H	I	I	L	L	M	O	O	R	**S**	**T**	**V**
Output:		**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**

b. Progress of Selection Sort algorithm

Trace of Selection sort algorithm with array contents for each iteration of outer loop. (Note: entries in **red** represent **a[min]**, entries in **grey** are **elements that are in their final positions**, and entries in **black are examined to find the minimum**. **Input** and **output** array contents are shown in **blue**

i	min	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input:		**I**	**L**	**O**	**V**	**E**	**A**	**L**	**G**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
0	**5**	**I**	**L**	**O**	**V**	**E**	**A**	**L**	**G**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
1	**4**	**A**	**L**	**O**	**V**	**E**	**I**	**L**	**G**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
2	**7**	**A**	**E**	**O**	**V**	**L**	**I**	**L**	**G**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
3	**12**	**A**	**E**	**G**	**V**	**L**	**I**	**L**	**O**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
4	**5**	**A**	**E**	**G**	**H**	**L**	**I**	**L**	**O**	**O**	**R**	**I**	**T**	**V**	**M**	**S**
5	**10**	**A**	**E**	**G**	**H**	**I**	**L**	**L**	**O**	**O**	**R**	**I**	**T**	**V**	**M**	**S**
6	**6**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**O**	**O**	**R**	**L**	**T**	**V**	**M**	**S**
7	**10**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**O**	**O**	**R**	**L**	**T**	**V**	**M**	**S**
8	**13**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**O**	**R**	**O**	**T**	**V**	**M**	**S**
9	**10**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**R**	**O**	**T**	**V**	**O**	**S**
10	**13**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**R**	**T**	**V**	**O**	**S**
11	**13**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**T**	**V**	**R**	**S**
12	**14**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**V**	**T**	**S**
13	**13**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
14	**14**	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
Output:		**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**

c. Progress of Bubble Sort algorithm

Trace of Bubble sort algorithm with array contents just after each iteration of outer loop.
(Note: entries in **red** represent **the entries that are put in their final position in current iteration**, entries in **grey** are **elements are considered for swapping**, and entries in **black that have already been placed in their correct final position earlier**. **Input** and **output** array contents are shown in **blue**

	i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Input:		**I**	**L**	**O**	**V**	**E**	**A**	**L**	**G**	**O**	**R**	**I**	**T**	**H**	**M**	**S**
	0	I	L	O	E	A	L	G	O	R	I	T	H	M	S	**V**
	1	I	L	E	A	L	G	O	O	I	R	H	M	S	**T**	**V**
	2	I	E	A	L	G	L	O	I	O	H	M	R	**S**	**T**	**V**
	3	E	A	I	G	L	L	I	O	H	M	O	**R**	**S**	**T**	**V**
	4	A	E	G	I	L	I	L	H	M	O	**O**	**R**	**S**	**T**	**V**
	5	A	E	G	I	I	L	H	L	M	**O**	**O**	**R**	**S**	**T**	**V**
	6	A	E	G	I	I	H	L	L	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	7	A	E	G	I	H	I	L	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	8	A	E	G	H	I	I	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	9	A	E	G	H	I	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	10	A	E	G	H	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	11	A	E	G	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	12	A	E	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
	13	**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**
Output:		**A**	**E**	**G**	**H**	**I**	**I**	**L**	**L**	**M**	**O**	**O**	**R**	**S**	**T**	**V**

Based on the above tables of the traces, we can make the following comparisons:

1. Selection sort uses $\approx n^2/2$ comparisons and n exchanges for the array of size n as we see about half of the trace table is in grey color. Since the diagonal elements correspond to exchanges there are $\approx n$ exchanges. Since the process to determine the smallest does not keep track of position of smallest element, this algorithm runs in the same time for partially sorted or fully sorted arrays so it is not sensitive to input ordering. But it has the advantage that number of exchanges is linear in problem size n

2. Insertion sort uses $\approx n^2/4$ comparisons and $\approx n^2/4$ exchanges as we observe from the number of entries below diagonal in the trace table. Number of compares is number of exchanges and an additional term for number of insertions. Insertion sort works well for partially or fully ordered arrays where the running time approaches linear.

3. For bubblesort, the outer loop iterates iterate once for each element in the data set (of size n) while the inner loop iterates n times the first time it is entered, n-1 times the second, and so on. Therefore, for each pass the next largest element of the data is moved to its proper place which means to get all n elements in their correct places, the outer loop must be executed n times.The best case scenario for bubble sort occurs when the list is already sorted or partially sorted because once the list is sorted, bubble sort will terminate after that specific iteration, when no swaps are made. Any time that a pass is made through the list and no swaps were made, we can be sure that the list has already been sorted. For scenarios where when one random element needs to be sorted into a sorted list, it can be relatively fast. The absolute worst case for bubble sort is when the smallest element of the list is at the end. Because in each iteration only the largest unsorted element gets put in its proper location, when the smallest element is at the end, it will have to be swapped each time through the list, and it wont get to the front of the list until all n iterations have occurred. In this worst case, it take n iterations of $n/2$ swaps so the order is n^2.

Based on the traces and the above discussions we can compare the complexity of these algorithms as shown in the table below:

Comparison of complexity of Bubblesort, Selection sort and Insertion sort algorithms

	Time Complexity			Space Complexity	
Sorting Algorithm	**Best Case**	**Average Case**	**Worst Case**	**Worst Case**	**Remark**
Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$	O(1)	Stop after reaching sorted array
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$	O(1)	Even a perfectly sorted input requires scanning the entire array
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$	O(1)	In the best case (already sorted), every insert requires constant time

1.2 Shellsort

The table below shows in the style of the example discussed in class, how shell sort works for the array $E, A, S, Y, S, H, E, L, L, S, O, R, T, Q, U, E, S, T, I, O, N$

1.2 Progress of ShellSort algorithm

Trace of Shell sort algorithm with increment sequence which uses decreasing values in $\frac{1}{2}(3^k - 1)$ starting at smallest increment greater than or equal to $\lfloor n/3 \rfloor$ and decreasing to 1 and implementing insertion sort for each h-subsequence. The table below shows the array contents just after each iteration of outer loop. (Note: entries in **red** represent **the entries that are inserted during the current $h-sort$ step**, entries in **grey are not affected in current step**, and entries in **black are other elements that are sorted in current $h-sort$ step**.**Input** and **output** array contents are shown in **blue**.

	h	i	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Input:				**E**	**A**	**S**	**Y**	**S**	**H**	**E**	**L**	**L**	**S**	**O**	**R**	**T**	**Q**	**U**	**E**	**S**	**T**	**I**	**O**	**N**
	13	**13**	**13**	E	A	S	Y	S	H	E	L	L	S	O	R	T	**Q**	U	E	S	T	I	O	N
	13	**14**	**14**	E	A	S	Y	S	H	E	L	L	S	O	R	T	Q	**U**	E	S	T	I	O	N
	13	**15**	**2**	E	A	**E**	Y	S	H	E	L	L	S	O	R	T	Q	U	**S**	S	T	I	O	N
	13	**16**	**3**	E	A	E	**S**	S	H	E	L	L	S	O	R	T	Q	U	S	**Y**	T	I	O	N
	13	**17**	**17**	E	A	E	S	S	H	E	L	L	S	O	R	T	Q	U	S	Y	**T**	I	O	N
	13	**18**	**18**	E	A	E	S	S	H	E	L	L	S	O	R	T	Q	U	S	Y	T	**I**	O	N
	13	**19**	**19**	E	A	E	S	S	H	E	L	L	S	O	R	T	Q	U	S	Y	T	I	**O**	N
	13	**20**	**20**	E	A	E	S	S	H	E	L	L	S	O	R	T	Q	U	S	Y	T	I	O	**N**
	4	**4**	**4**	E	A	E	S	**S**	H	E	L	L	S	O	R	T	Q	U	S	Y	T	I	O	N
	4	**5**	**5**	E	A	E	S	S	**H**	E	L	L	S	O	R	T	Q	U	S	Y	T	I	O	N
	4	**6**	**6**	E	A	E	S	S	H	**E**	L	L	S	O	R	T	Q	U	S	Y	T	I	O	N
	4	**7**	**3**	E	A	E	**L**	S	H	E	**S**	L	S	O	R	T	Q	U	S	Y	T	I	O	N
	4	**8**	**4**	E	A	E	L	**L**	H	E	S	**S**	S	O	R	T	Q	U	S	Y	T	I	O	N
	4	**9**	**9**	E	A	E	L	L	H	E	S	S	**S**	O	R	T	Q	U	S	Y	T	I	O	N
	4	**10**	**10**	E	A	E	L	L	H	E	S	S	S	**O**	R	T	Q	U	S	Y	T	I	O	N
	4	**11**	**7**	E	A	E	L	L	H	E	**R**	S	S	O	**S**	T	Q	U	S	Y	T	I	O	N
	4	**12**	**12**	E	A	E	L	L	H	E	R	S	S	O	S	**T**	Q	U	S	Y	T	I	O	N
	4	**13**	**9**	E	A	E	L	L	H	E	R	S	**Q**	O	S	T	**S**	U	S	Y	T	I	O	N
	4	**14**	**14**	E	A	E	L	L	H	E	R	S	Q	O	S	T	S	**U**	S	Y	T	I	O	N
	4	**15**	**15**	E	A	E	L	L	H	E	R	S	Q	O	S	T	S	U	**S**	Y	T	I	O	N
	4	**16**	**16**	E	A	E	L	L	H	E	R	S	Q	O	S	T	S	U	S	**Y**	T	I	O	N
	4	**17**	**17**	E	A	E	L	L	H	E	R	S	Q	O	S	T	S	U	S	Y	**T**	I	O	N
	4	**18**	**10**	E	A	E	L	L	H	E	R	S	Q	**I**	S	T	S	**O**	S	Y	T	**U**	O	N
	4	**19**	**7**	E	A	E	L	L	H	E	**O**	S	Q	I	**R**	T	S	O	**S**	Y	T	U	**S**	N
	4	**20**	**8**	E	A	E	L	L	H	E	O	**N**	Q	I	R	**S**	S	O	S	**T**	T	U	S	**Y**
	1	**1**	**0**	**A**	E	E	L	L	H	E	O	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**2**	**2**	A	E	**E**	L	L	H	E	O	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**3**	**3**	A	E	E	**L**	L	H	E	O	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**4**	**4**	A	E	E	L	L	H	E	O	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**5**	**3**	A	E	E	**H**	**L**	**L**	E	O	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**6**	**3**	A	E	E	**E**	**H**	**L**	**L**	O	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**7**	**7**	A	E	E	E	H	L	L	**O**	N	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**8**	**7**	A	E	E	E	H	L	L	**N**	**O**	Q	I	R	S	S	O	S	T	T	U	S	Y
	1	**9**	**9**	A	E	E	E	H	L	L	N	O	**Q**	I	R	S	S	O	S	T	T	U	S	Y
	1	**10**	**5**	A	E	E	E	H	**I**	**L**	**L**	**N**	**O**	**Q**	R	S	S	O	S	T	T	U	S	Y
	1	**11**	**11**	A	E	E	E	H	I	L	L	N	O	Q	**R**	S	S	O	S	T	T	U	S	Y
	1	**12**	**12**	A	E	E	E	H	I	L	L	N	O	Q	R	**S**	S	O	S	T	T	U	S	Y
	1	**13**	**13**	A	E	E	E	H	I	L	L	N	O	Q	R	S	**S**	O	S	T	T	U	S	Y
	1	**14**	**10**	A	E	E	E	H	I	L	L	N	O	**O**	**Q**	**R**	**S**	**S**	S	T	T	U	S	Y
	1	**15**	**15**	A	E	E	E	H	I	L	L	N	O	O	Q	R	S	S	**S**	T	T	U	S	Y
	1	**16**	**16**	A	E	E	E	H	I	L	L	N	O	O	Q	R	S	S	S	**T**	T	U	S	Y
	1	**17**	**17**	A	E	E	E	H	I	L	L	N	O	O	Q	R	S	S	S	T	**T**	U	S	Y
	1	**18**	**18**	A	E	E	E	H	I	L	L	N	O	O	Q	R	S	S	S	T	T	**U**	S	Y
	1	**19**	**16**	A	E	E	E	H	I	L	L	N	O	O	Q	R	S	S	S	**S**	**T**	**T**	**U**	Y
	1	**20**	**20**	A	E	E	E	H	I	L	L	N	O	O	Q	R	S	S	S	S	T	T	U	**Y**
Output:				**A**	**E**	**E**	**E**	**H**	**I**	**L**	**L**	**N**	**O**	**O**	**Q**	**R**	**S**	**S**	**S**	**S**	**T**	**T**	**U**	**Y**

2 Problem 2 - Mergesort

Solution The tables below give traces, in the style discussed in class, showing how the keys $I, L, O, V, E, A, L, G, O, R, I, T, H, M, S$ are sorted using 2 different mergesort algorithms.

2a. MergeSort Top Down Sorting Trace

Trace of merge results for top-down mergesort showing the sequence of merges after sorting the subarrays at the indices shown. At each merge step the merge of arrays **a[lo...mid] and a[mid+1...hi]** is done (each subarray is already independently sorted via recursive calls). The merge process merges adjacent arrays of length 1 then adjacent ones of length 2 and so on. At each merge step, the boundaries of the subarray being merged in current step are shown in **red** for the **lo, hi** arguments in the merge call. Note: Entries in **black are being merged in the current step**. Entries in **grey are elements that are not impacted in the current merge step**. **Input** and **output** array contents are shown in **blue**

							0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input:							I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
			merge(a,	0,	0,	1)	I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
			merge(a,	2,	2,	3)	I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
		merge(a,	0,	1,	3)		I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
			merge(a,	4,	4,	5)	I	L	O	V	A	E	L	G	O	R	I	T	H	M	S
			merge(a,	6,	6,	7)	I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
		merge(a,	4,	5,	7)		I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
	merge(a,	0,	3,	7)			A	E	G	I	L	L	O	V	O	R	I	T	H	M	S
			merge(a,	8,	8,	9)	A	E	G	I	L	L	O	V	O	R	I	T	H	M	S
			merge(a,	10,	10,	11)	A	E	G	I	L	L	O	V	O	R	I	T	H	M	S
		merge(a,	8,	9,	11)		A	E	G	I	L	L	O	V	I	O	R	T	H	M	S
			merge(a,	12,	12,	13)	A	E	G	I	L	L	O	V	I	O	R	T	H	M	S
		merge(a,	12,	13,	14)		A	E	G	I	L	L	O	V	I	O	R	T	H	M	S
	merge(a,	8,	11,	14)			A	E	G	I	L	L	O	V	H	I	M	O	R	S	T
merge(a,	0,	7,	14)				A	E	G	H	I	I	L	L	M	O	O	R	S	T	V
Output:							A	E	G	H	I	I	L	L	M	O	O	R	S	T	V

2b. MergeSort Bottoms Up Sorting Trace

Trace of merge results for bottoms-up mergesort showing the sequence of merges after sorting the subarrays at the indices shown. The merge process merges all arrays of length 1 to start with their neighbor, subsequently the adjacent merged subarrays of length 2 are merged into subarrrays of length 4 and so on. At each merge step, the boundaries of the subarray being merged in current step are shown in **red** for the **lo, hi** arguments in the merge call. Note: Entries in **black are being merged in the current step**. Entries in **grey are elements that are not impacted in the current merge step**. **Input** and **output** array contents are shown in **blue**.

							0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input:							I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
len = 1																					
			merge(a,	0,	0,	1)	I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
			merge(a,	2,	2,	3)	I	L	O	V	E	A	L	G	O	R	I	T	H	M	S
			merge(a,	4,	4,	5)	I	L	O	V	A	E	L	G	O	R	I	T	H	M	S
			merge(a,	6,	6,	7)	I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
			merge(a,	8,	8,	9)	I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
			merge(a,	10,	10,	11)	I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
			merge(a,	12,	12,	13)	I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
len = 2																					
		merge(a,	0,	1,	3)		I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
		merge(a,	4,	5,	7)		I	L	O	V	A	E	G	L	O	R	I	T	H	M	S
		merge(a,	8,	9,	11)		I	L	O	V	A	E	G	L	I	O	R	T	H	M	S
		merge(a,	12,	13,	14)		I	L	O	V	A	E	G	L	I	O	R	T	H	M	S
len = 4																					
	merge(a,	0,	3,	7)			A	E	G	I	L	L	O	V	I	O	R	T	H	M	S
	merge(a,	8,	11,	14)			A	E	G	I	L	L	O	V	H	I	M	O	R	S	T
len = 8																					
merge(a,	0,	7,	14)				A	E	G	H	I	I	L	L	M	O	O	R	S	T	V
Output:							A	E	G	H	I	I	L	L	M	O	O	R	S	T	V