Tracing Selection and Insertion Sort

This document shows how to trace the execution of selection and insertion sort. The examples are taken from the slides on sorting algorithms.

Note that different sources use slightly different versions of these algorithms. Please use the versions given in this document when answering questions on the homework and exams.

Selection Sort: Below is a method that performs selection sort on a perfect size array of integers.

```
public static void selectionSort(int[] array)
{
    for (int i = 0; i < array.length - 1; ++i)
    {
        int idxMin = i;
        for (int j = i + 1; j < array.length; ++j)
        {
            if (array[j] < array[idxMin])
            {
                idxMin = j;
            }
        }

        int temp = array[i];
        array[i] = array[idxMin];
        array[idxMin] = temp;
    }
}
```

On each iteration of the outer loop, the array consists of a sorted and an unsorted part. The elements at indices less than i are in their final sorted positions, and the elements at indices greater than or equal to i are unsorted.

The inner loop finds the minimum element in the unsorted part of the array. The minimum is then swapped with the element at index i, increasing the size of the sorted part by one element.

Note that the outer loop stops after sorting the second-to-last element (i < array.length – 1). This is because if all the elements other than the last are in their final sorted positions, then the last element must also be in its sorted position.

Tracing Selection Sort: When tracing selection sort, use a separate row to show the values swapped on each iteration of the outer loop. If two values are swapped, two values will appear on the row. If a value is swapped with itself, only one value will appear.

Selection Sort Example 1:

9	**4**	**2**	**1**	**7**	**8**	**3**	**6**	**5**
1			9					
	2	4						
		3				4		
			4			9		
				5				7
					6		8	
						7		9
							8	

Selection Sort Example 2:

1	**4**	**7**	**9**	**5**	**3**	**2**
1						
	2					4
		3			7	
			4			9
				5		
					7	

Insertion Sort: Below is a method that performs insertion sort on a perfect size array of integers.

```
public static void insertionSort(int[] array)
{
    for (int i = 1; i < array.length; ++i)
    {
        int temp = array[i];

        int j;
        for (j = i; j > 0; --j)
        {
            if (array[j-1] > temp)
            {
                array[j] = array[j-1];
            }
            else {
                break;
            }
        }

        array[j] = temp;
    }
}
```

On each iteration of the outer loop, the array consists of a sorted and an unsorted part. The elements at indices less than i are in the sorted part, and the elements at indices greater than or equal to i are in the unsorted part.

Unlike selection sort, the elements in the sorted part are not necessarily in their final sorted positions. Instead, they are only sorted with respect to the other elements in the sorted part of the array.

The inner loop finds the index where element i, the first element in the unsorted part, must be inserted to increase the size of the sorted part by one element. The inner loop makes room for the new element by shifting the values that are larger than element i to the right.[1]

Note that the outer loop starts at the second element (i = 1), which means that the sorted part of the array initially consists of the first element. This is possible with insertion sort because the elements in this part are only in relative sorted order, and a single element is always sorted with respect to itself.

Tracing Insertion Sort: When tracing insertion sort, use a separate row to show each assignment to an element of the array or temporary storage. Only a single value will appear on

[1] The algorithm does not swap element i with each of the larger values in the sorted part. Instead, it copies element i to temporary storage, assigns each larger value to the element to its right, and then copies the value in temporary storage into the correct position in the sorted part.

each row. This illustrates how values in the sorted part are shifted to make room for the new value.

Insertion Sort Example 1:

9	4	2	1	7	8	3	6	5	temp
									4
	9								
4									
									2
		9							
	4								
2									
									1
			9						
		4							
	2								
1									
									7
				9					
			7						
									8
					9				
				8					
									3
						9			
					8				
				7					
			4						
		3							
									6
							9		
						8			
					7				
				6					
									5
								9	
							8		
						7			
					6				
				5					

Insertion Sort Example 2:

1	4	7	9	5	3	2	temp
							4
	4						
							7
		7					
							9
			9				
							5
				9			
			7				
		5					
							3
					9		
				7			
			5				
		3					
							2
						9	
					8		
				5			
			3				
		2					