j	Action	Array contents
0	since A[0] <= A[7], i=i+1=-1+1=0	
	we then exchange A[i] w/ A[j], A[0] w/ A[0]	[1,5,3,2,0,4,6,8]
1	since A[1]<= 8, i=i+1=0+1=1	
	we then exchange A[i] w/ A[j], A[1] w/ A[1]	[1,5,3,2,0,4,6,8]
2	since A[2]<=8,i=i+1=1+1=2	
	we exchange A[i] w/ A[j], A[2] w/ A[2]	[1,5,3,2,0,4,6,8]
3	since A[3]<=8, i=i+1=2+1=3	
	we exchange A[i] w/ A[j], A[3] w/ A[3]	[1,5,3,2,0,4,6,8]

4	since A[4]<=8,i=i+1=3+1=4	
	we exchange A[i] w/ A[j], A[4] w/ A[4]	[1,5,3,2,0,4,6,8]
5	since A[5]<=8, i=i+1=4+1=5	
	we exchange A[i] w/ A[j], A[5] w/ A[5]	[1,5,3,2,0,4,6,8]
6	since A[6]<=8,i=i+1=5+1=6	
	we exchange A[i] w/ A[j], A[6] w/ A[6]	[1,5,3,2,0,4,6,8]
	out of for loop, A[7] exchange with A[7], result array is the same	
	partition returns i+1=6+1=7	

2a	selection sort
1st	[5,7,1,9,45,23,7]
2nd	[45,7,1,9,5,23,7]
3rd	[45,23,1,9,5,7,7]
4th	[45,23,9,1,5,7,7]
5th	[45,23,9,7,5,1,7]
6th	[45,23,9,7,7,1,5]
7th	[45,23,9,7,7,5,1]
2b	insertion sort
1st	[7,5,1,9,45,23,7]

2nd [7,5,1,9,45,23,7]

3rd [9,7,5,1,45,23,7]

4th [45,9,7,5,1,23,7]

5th [45,23,9,7,5,1,7]

6th [45,23,9,7,7,5,1]

7th

2c bubble sort

1st [5,7,7,9,23,45,1]

2nd [7,7,9,23,45,5,1]

3rd [7,9,23,45,7,5,1]

4th [9,23,45,7,7,5,1]

5th [23,45,9,7,7,5,1]

6th [45,23,9,7,7,5,1]

p3 For (i=1 to n) {

Initialize storeIndex to keep track of the next index to replace with negative

If negative found, switch the element at i with element at storeIndex

Increment storeIndex

}

return arranged array

p4 1. not efficient to sort 1st

For (i=1 to size_of_array) {

```
        If (elem_In_Index_Of_array > elem_stored_in_var)

                elem_stored_in_var = elem_In_Index_Of_array

}

return elem_stored_in_var
```

2. efficient to sort, because to find mode I have to find out the maximum repetition of each value; thus

in this case, it is better to sort the values within the array

than to create a new array the size of the unsorted array just to store the count of each value.

3. efficient to sort because to find the median number we have to put all the elements in a order 1st.

and if the total number of elements are odd, the median is the middle number; .

else the median will be the mean of the two central numbers

4. not efficient to sort

```
For (i=1 to size_of_array) {
        If (array[i] == particular_item){
            Increment particular_item_count
        }
Return particular_item_count
```

5.not efficient to sort

```
For (i=1 to size_of_array) {
            If ( array[i] < store_min_value ){
                store_min_value = array[i]
```

```
            }

}
```

Return store_min_value

j	Action	Array contents
0	since A[0] <= A[7], i=i+1=-1+1=0	
	we then exchange A[i] w/ A[j], A[0] w/ A[0]	[1,5,3,2,0,4,6,8]
1	since A[1]<= 8, i=i+1=0+1=1	
	we then exchange A[i] w/ A[j], A[1] w/ A[1]	[1,5,3,2,0,4,6,8]
2	since A[2]<=8,i=i+1=1+1=2	
	we exchange A[i] w/ A[j], A[2] w/ A[2]	[1,5,3,2,0,4,6,8]
3	since A[3]<=8, i=i+1=2+1=3	
	we exchange A[i] w/ A[j], A[3] w/ A[3]	[1,5,3,2,0,4,6,8]

4	since A[4]<=8,i=i+1=3+1=4	
	we exchange A[i] w/ A[j], A[4] w/ A[4]	[1,5,3,2,0,4,6,8]
5	since A[5]<=8, i=i+1=4+1=5	
	we exchange A[i] w/ A[j], A[5] w/ A[5]	[1,5,3,2,0,4,6,8]
6	since A[6]<=8,i=i+1=5+1=6	
	we exchange A[i] w/ A[j], A[6] w/ A[6]	[1,5,3,2,0,4,6,8]
	out of for loop, A[7] exchange with A[7], result array is the same	
	partition returns i+1=6+1=7	

2a	selection sort
1st	[5,7,1,9,45,23,7]
2nd	[45,7,1,9,5,23,7]
3rd	[45,23,1,9,5,7,7]
4th	[45,23,9,1,5,7,7]
5th	[45,23,9,7,5,1,7]
6th	[45,23,9,7,7,1,5]
7th	[45,23,9,7,7,5,1]
2b	insertion sort
1st	[7,5,1,9,45,23,7]

2nd [7,5,1,9,45,23,7]

3rd [9,7,5,1,45,23,7]

4th [45,9,7,5,1,23,7]

5th [45,23,9,7,5,1,7]

6th [45,23,9,7,7,5,1]

7th

2c bubble sort

1st [5,7,7,9,23,45,1]

2nd [7,7,9,23,45,5,1]

3rd [7,9,23,45,7,5,1]

4th [9,23,45,7,7,5,1]

5th [23,45,9,7,7,5,1]

6th [45,23,9,7,7,5,1]

p3 For (i=1 to n) {

Initialize storeIndex to keep track of the next index to replace with negative

If negative found, switch the element at i with element at storeIndex

Increment storeIndex

}

return arranged array

p4 1. not efficient to sort 1st

For (i=1 to size_of_array) {

```
        If (elem_In_Index_Of_array > elem_stored_in_var)

                elem_stored_in_var = elem_In_Index_Of_array

}

return elem_stored_in_var
```

2. efficient to sort, because to find mode I have to find out the maximum repetition of each value; thus

in this case, it is better to sort the values within the array

than to create a new array the size of the unsorted array just to store the count of each value.

3. efficient to sort because to find the median number we have to put all the elements in a order 1st.

and if the total number of elements are odd, the median is the middle number; .

else the median will be the mean of the two central numbers

4. not efficient to sort

```
For (i=1 to size_of_array) {
        If (array[i] == particular_item){
            Increment particular_item_count
        }
Return particular_item_count
```

5.not efficient to sort

```
For (i=1 to size_of_array) {
            If ( array[i] < store_min_value ){
                store_min_value = array[i]
```

```
            }

}
```

Return store_min_value