*Owen Lindsey*

*Professor Demland, David*

*CST-201 Compare Sorting*

*11/03/2024*

Pseudo code for compare sorting program   
**Merge Sort Helper Method:**   
***METHOD*** *Merge(array, startIndex, middleIndex, endIndex) RETURNING metrics*

***SET*** *metrics to new SortMetrics*

***CREATE*** *leftArray of size middleIndex - startIndex + 1*

***CREATE*** *rightArray of size endIndex - middleIndex*

***COPY*** *array from startIndex to middleIndex into leftArray*

***COPY*** *array from middleIndex + 1 to endIndex into rightArray*

***SET*** *leftIndex to 0*

***SET*** *rightIndex to 0*

***SET*** *mergedIndex to startIndex*

***WHILE*** *leftIndex < leftArray length* ***AND*** *rightIndex < rightArray length*

***INCREMENT*** *metrics.Comparisons*

***IF*** *leftArray[leftIndex] <= rightArray[rightIndex]* ***THEN***

***SET*** *array[mergedIndex] to leftArray[leftIndex]*

***INCREMENT*** *leftIndex*

***ELSE***

***SET*** *array[mergedIndex] to rightArray[rightIndex]*

***INCREMENT*** *rightIndex*

***END IF***

***INCREMENT*** *metrics.Exchanges*

***INCREMENT*** *mergedIndex*

***END WHILE***

***WHILE*** *leftIndex < leftArray length*

***SET*** *array[mergedIndex] to leftArray[leftIndex]*

***INCREMENT*** *leftIndex*

***INCREMENT*** *mergedIndex*

***INCREMENT*** *metrics.Exchanges*

***END WHILE***

Pseudo code for compare sorting program   
**Merge Sort Helper Method:**

***WHILE*** *rightIndex < rightArray length*

***SET*** *array[mergedIndex] to rightArray[rightIndex]*

***INCREMENT*** *rightIndex*

***INCREMENT*** *mergedIndex*

***INCREMENT*** *metrics.Exchanges*

***END WHILE***

***RETURN*** *metrics*

***Shuffle:***

***METHOD*** *Shuffle(array)*

***FOR*** *currentIndex = 0 to array length - 1*

***SET*** *randomIndex to random number between 0 and array length - 1*

***SET*** *temporaryValue to array[currentIndex]*

***SET*** *array[currentIndex] to array[randomIndex]*

***SET*** *array[randomIndex] to temporaryValue*

***END FOR***

**SelectionSort:**

***METHOD*** *SelectionSort(array)* ***RETURNING*** *metrics*

***SET*** *metrics to new SortMetrics*

***FOR*** *currentIndex = 0 to array length - 2*

***SET*** *minimumIndex to currentIndex*

***FOR*** *searchIndex = currentIndex + 1 to array length - 1*  
 ***INCREMENT*** *metrics.Comparisons*  
   ***IF*** *array[searchIndex] < array[minimumIndex]* ***THEN***  
  ***SET*** *minimumIndex to searchIndex*  
  ***END IF***  
***END FOR***

Pseudo code for compare sorting program   
**SelectionSort:**

***IF*** *minimumIndex is not currentIndex* ***THEN***  
***SET*** *temporaryValue to array[currentIndex]*  
***SET*** *array[currentIndex] to array[minimumIndex]*  
 ***SET*** *array[minimumIndex] to temporaryValue*  
 ***INCREMENT*** *metrics.Exchanges*  
***END IF***

**END FOR**

**RETURN** metrics

**BubbleSort:**

***METHOD*** *BubbleSort(array)* ***RETURNING*** *metrics*

***SET*** *metrics to new SortMetrics*

***FOR*** *passIndex = 0 to array length – 2*

***FOR*** *compareIndex = 0 to array length - passIndex - 2*

***INCREMENT*** *metrics.Comparisons*

***IF*** *array[compareIndex] > array[compareIndex + 1]* ***THEN***

***SET*** *temporaryValue to array[compareIndex]*

***SET*** *array[compareIndex] to array[compareIndex + 1]*

***SET*** *array[compareIndex + 1] to temporaryValue*

***INCREMENT*** *metrics.Exchanges*

***END IF***

***END FOR***

***END FOR***

***RETURN*** *metrics*

Pseudo code for compare sorting program

**QuickSort:**

***METHOD*** *QuickSort(array, startIndex, endIndex)* ***RETURNING*** *metrics*

***SET*** *metrics to new SortMetrics*

***IF*** *startIndex < endIndex* ***THEN***

***CALL*** *Partition with array, startIndex, endIndex* ***RETURNING*** *pivotIndex, partitionMetrics*

***CALL*** *QuickSort with array, startIndex, pivotIndex - 1* ***RETURNING*** *leftMetrics*

***CALL*** *QuickSort with array, pivotIndex + 1, endIndex* ***RETURNING*** *rightMetrics*

***SET*** *metrics.Comparisons to partitionMetrics.Comparisons + leftMetrics.Comparisons + rightMetrics.Comparisons*  
***SET*** *metrics.Exchanges to partitionMetrics.Exchanges + leftMetrics.Exchanges + rightMetrics.Exchanges*

***END IF***

***RETURN*** *metrics*

**MergeSort:**

**METHOD** *MergeSort(array, startIndex, endIndex)* **RETURNING** *metrics*

***SET*** *metrics to new SortMetrics*

***IF*** *startIndex < endIndex* ***THEN***

***SET*** *middleIndex to startIndex + (endIndex - startIndex) / 2*

***CALL*** *MergeSort with array, startIndex, middleIndex* ***RETURNING*** *leftMetrics*  
 ***CALL*** *MergeSort with array, middleIndex + 1, endIndex* ***RETURNING*** *rightMetrics*  
 ***CALL*** *Merge with array, startIndex, middleIndex, endIndex* ***RETURNING*** *mergeMetrics*  
 ***SET*** *metrics.Comparisons to leftMetrics.Comparisons + rightMetrics.Comparisons + mergeMetrics.Comparisons*  
 ***SET*** *metrics.Exchanges to leftMetrics.Exchanges + rightMetrics.Exchanges + mergeMetrics.Exchanges*

***END IF***

***RETURN*** *metrics*

***Partition:***

**METHOD** Partition(array, lowIndex, highIndex) **RETURNING** pivotIndex, metrics

**SET** metrics to new SortMetrics

**SET** pivotValue to array[highIndex]

**SET** smallerElementIndex to lowIndex - 1

**FOR** currentIndex = lowIndex to highIndex - 1

**INCREMENT** metrics.Comparisons

**IF** array[currentIndex] < pivotValue **THEN**

**INCREMENT** smallerElementIndex

**SET** temporaryValue to array[smallerElementIndex]

**SET** array[smallerElementIndex] to array[currentIndex]

**SET** array[currentIndex] to temporaryValue

**INCREMENT** metrics.Exchanges

**END IF**

**END FOR**

**SET** temporaryValue to array[smallerElementIndex + 1]

**SET** array[smallerElementIndex + 1] to array[highIndex]

**SET** array[highIndex] to temporaryValue

**INCREMENT** metrics.Exchanges

**RETURN** smallerElementIndex + 1, metrics

[***Loom video for comparison sorting application***](https://www.loom.com/share/a77e2eb525e14a2597fd3aaaf50ffc1a)