**Client**

**Data Structures Used:**

int SortingArray[listSize]

A dynamic array that is allocated at run-time that will test the various sorting algorithms

**Function: main()**

int\* selectionArray

int\* heapArray

int\* quickArray

int selectionComp

int heapComp

int quickComp

int selectionSwap

int heapSwap

int quickSwap

ifstream inputFile

string fileName

int listSize1, listSize2, listSize3, commonListSize

char response

string inputFileNumString

int inputFileNumInt

int I = 0

prompt user for list size

read in commonListSize

listSize1 = commonListSize

listSize2 = commonListSize

listSize3 = commonListSize

selectionArray = new int[listSize1]

heapArray = new int[listSize2]

quickArray = new int[listSize3]

prompt user to randomly generate numbers

read in response

if(response == “Y”)

RandomizeNumbers(selectionArray, listSize1, heapArray, listSize2, quickArray, listSize3)

else

prompt user for input file name

read in fileName

inputFile.open(fileName.c\_str())

PopulateArrays(selectionArray, listSize1, heapArray, listSize2, quickArray, listSize3, inputFile)

endif

RunSelectionSort(selectionArray, listSize1, selectionComp, selectionSwap)

RunHeapSort(heapArray, listSize2, heapComp, heapSwap)

RunQuickSort(quickArray, listSize3, quickComp, quickSwap)

inputFile.close()

**Function:** void RandomizeNumbers(int ary1[], int listSize1, int ary2[], int listSize2, int ary3[], int listSize3)

Narrative: Randomizes numbers for the arrays to use for the sorts

Pre-condition: user input indicates this function will be run, all list sizes are the same when this function is called

Post-condition: numbers have been randomized for use in the arrays

int randomNumber

for(int I = 0; I < listSize1; i++)

randomNumber = rand() % listSize1

ary1[i] = randomNumber

ary2[i] = randomNumber

ary3[i] = randomNumber

endfor

**Function:** void RunSelectionSort(int ary[], int listSize, int& comparisons, int& swaps)

Narrative: Runs the selection sort on the selection array

Pre-condition: selection array has been populated with values

Post-condition: selection sort has been performed on selection array

int minIndex

for(int I = 0; I < listSize – 2; i++)

minIndex = I

for(int j = I + 1; j < listSize – 1; j++)

if(ary[j] < a[minIndex])

comparisons++

minIndex = j

endif

endfor

if(minIndex != I)

Swap(a[i], a[minIndex])

swaps++

endif

endfor

**Function:** void RunHeapSort(int ary[], int listSize, int& comparisons, int& swaps)

Narrative: Runs the heap sort on the heap array

Pre-condition: heap array has been populated with values

Post-condition: heap sort has been performed on heap array

heapListSize = listSize

int heapAry[heapListSize]

int temp

heapAry[0] = ary[0]

for(int I = 1; I < heapListSize; i++)

heapAry[i] = ary[floor(i/2)]

heapAry[i+1] = ary[2i]

verify heap property //if statement for max heap condition

heapAry[i+2] = ary[2i + 1]

verify heap property //if statement for max heap condition

endfor

for(int I = listSize – 1; I >=0; i--)

temp = ary[0]

ary[0] = heapAry[0]

ary[i] = temp

HeapifyDown(heapAry, heapListSize)

heapListSize--

endfor

**Function:** void RunQuickSort(int ary[], int listSize, int low, int high, int& comparisons, int& swaps)

Narrative: Runs the quick sort on the quick array

Pre-condition: quick array has been populated with values

Post-condition: quick sort has been performed on the quick array

int partitionIndex

if(low < high)

partitionIndex = Partition(ary, low, high)

RunQuickSort(ary, low, partitionIndex – 1)

RunQuickSort(ary, partitionIndex + 1, high)

**Function:** void PopulateArrays(int ary1[], int listSize1, int ary2[], int listSize2, int ary3[], int listSize3, ifstream& inFile)

Narrative: reads in data from a file and populates all arrays with this data

Pre-condition: RandomizeNumbers not called, all list sizes are the same when this function is called

Post-condition: all arrays have been populated with values

string numFromFileString

int numFromFileInt

int I = 0

getline(inputFile, numFromFileString)

while(! End of inputFile)

numFromFileInt = stoi(numFromFileString, nullptr)

ary1[i] = numFromFileInt

ary2[i] = numFromFileInt

ary3[i] = numFromFileInt

i++

getline(inputFile, numFromFileString)

endwhile

**Function:** void Swap(int ary[], int listSize, int index1, int index2)

Narrative: swaps two numbers in terms of their index within the array

Pre-condition: Selection Sort is being performed, indices are valid within the array

Post-condition: two numbers have been swapped within the array passed in

int temp = ary[index2]

ary[index2] = ary[index1]

ary[index1] = temp

**Function**:void HeapifyDown(int ary[], int listSize, int index)

Narrative: Readjusts the heap array properly to maintain heap property

Pre-condition: Heapsort is being performed

Post-condition: Heap is properly sorted as a max heap, heap property is maintained

int largestValue = index

int leftChild = 2\*index + 1

int rightChild = 2\*index + 2

if(leftChild < listSize && ary[letfChild] > ary[largestValue])

largestValue = leftChild

if(rightChild < listSize && ary[rightChild] > ary[largestValue])

largestValue = rightChild

if(largestValue != index)

Swap(ary, index, largestValue)

HeapifyDown(ary, listSize, largestValue)

**Function:** int Partition(int ary[], int listSize, int low, int high)

Narrative: Partitions the list into subsections

Pre-condition: RunQuickSort function has been called

Post-condition: Partition has been created

int pivot = ary[low]

while(low < high and ary[high] >= pivot)

high = high – 1

if(low < high)

a[low] = a[high]

while(low < high and a[low] <= pivot)

low = low + 1

if(low < high)

a[high] = a[low]

while(low < high)

a[low] = pivot

return(low)