**CSCI 301 Moustafa Elsayed**

**Computer Science 2**

**Project 9: Comparing Sorting Algorithms**

**Introduction**

Sorting algorithms are useful in organizing data, in this project the program uses and compares four sorting algorithms; Insertion sort, Merge sort, Quick sort, and Heap sort. The program compares these sorting algorithms by counting the number of notations of each sorting algorithm. First the program asks for the number of values to be randomly generated, then asks for the seed value to randomly generate values, then the program asks the user whether to print the sorted arrays or not. Finally the program prints the number of notations each sorting algorithm took to sort the randomly generated values, and prints the sorted arrays if the user wanted to.

**Data Structures**

This program uses 27 data sturctures :-

* A local type int “countMerge”: that will count the number of notations of the merge sort in the merge() function.
* A local type int “countInsert”: that will count the number of notations of the insertion sort in the insertionSort() function.
* A local type int “countQuick”: that will count the number of notations of the quick sort in the partition() function.
* A local type int “heapCount”: that will count the number of notations of the heap sort in both the reheap\_down() function and the heapSort() function.
* A constant local type int “max”: that holds the maximum amount of values that can be generated in this program.
* An int type “randomvalue” in the main function: that holds the randomly generated value temporarily until that value is assigned in all four arrays of the sorting algorithms.
* An int type “seed” in the main function: that holds the seed value to randomly generate values in the srand() funcntion.
* An in type “numValue” in the main function: that holds the number of values the user wants to randomly generate.
* A char type “answer” in the main function: that holds the reply of the user whether to print the sorted arrays or not.
* A pointer int type “merge” in the main function: that will be the array that will hold the randomly generated values for the merge sort.
* A pointer int type “heap” in the main function: that will be the array that will hold the randomly generated value for the heap sort.
* A pointer int type “insert” in the main function: that will be the array that will hold the randomly generated values for the insertion sort.
* A pointer int type “quick” in the main function: that will be the array that will hold the randomly generated values for the quick sort.
* An int type array “b” in the merge() function: that will temporarily hold the value of the input array.
* An int type “i1” in the merge() function: that will be the index of the first half of value of the array.
* An int type “i2” in the merge() function: that will be the index of the second half of values of the array.
* An int type “i” in the merge() function: that will be the index for all the values in of the array.
* An int type “mid” in the mergeSort() function: that will be the index of the middle place in the array.
* An int type “i” in the insertionSort() function: that will be the index of the array except the first element.
* An int type “key” in the insertionSort() function: that will hold the value of the current array in the loop.
* An int type “j” in the insertionSort() function: that will hold the value of i-1.
* An int type “t” in the swap() function: that will temporarily hold the value of the integer in the first parameter.
* An int “pivot” in the partition() function: that will hold the value of the last element in the input array.
* An int “i” in the partition() function: that will hold the index of the first element in the array -1.
* An int “j” in the partition() function: that will hold the index of the first element in the array.
* An int “pi” in the quickSort() function: that will hold the returned value of the partition() function.
* An int “max\_child” in the reheap\_down() function: that will hold the value of the parent parameter multiplied by 2 and +1.
* A Boolean type “done” in the reheap\_down(): that will flag when the loop should stop.

**Functions**

There are 8 functions in this program:-

* Void merge(): This function will divide the parameter array into two halves, then assign an index for the first half of the array, and assign another index for the second half of the array. The function then will compare the value of arrays at the index of the first half and the index of the second half, whichever value is smaller gets assigned to the main array. Then the function assigns the remaining values at both indexes to the main array.
* Void mergeSort(): This function will first make sure that the low parameter is smaller than the high parameter, if this condition is met then the function creates a middle index for the main array, and recurs two times, the first recursion is with the middle index inserted instead of the high index, and the second recursion is with the middle index +1 instead of the low index, then the function calls the merge() function.
* Void insertionSort(): This function assigns the compares each element with the one after it, then assigns the smaller element to the lower index, this process is repeated until that value is right after its smaller value and right before its larger value.
* Void swap(): This function swaps the value of two variables.
* int partition(): This function assigns the last element in the array as a pivot and places all the elements that are larger than pivot after the pivot and all the elements that are smaller than the pivot before the pivot, then the function returns the index at which the function has stopped.
* Void quicksort(): This function checks if the low parameter is smaller than the high parameter, if this condition is met. The function then calls the function partition() and assigns its returned value to the high parameter of the first recursion of the quickSort() function and is +2 and assigned to the low parameter of the second recursion.
* Void reheap\_down(): This function assigns a parent and a child, then compares the index of the child with the child next to it, whichever is larger moves closer to the parent index and the smaller value move farther away from the parent.
* Void heapSort(): This function uses a loop to rearrange the array as a heap by calling the reheap\_down() function, then uses a second loop to swap each element with the first element and calls the reheap\_down() function.

**Main function**

The program first asks the user for the number of values to be sorted, if the number of values are less than or equal to the limit then the program proceed to ask the user for the seed value, then the program asks the user whether to print the sorted values or not. The program then, generates random values and assigns them to four arrays, each array is to be used for its sorting algorithm. The function then calls the four functions to sort the array in four different ways; mergeSort(), heapSort(), insertionSort(), quickSort(). If the user wants to print the sorted values then the program prints the sorted values. If not then the program proceeds to the next step, in which the program prints the number of notations of each sorting algorithm that took to sort this array.