

Author: Michael LaPan
Assignment: Program 10

The purpose of this program is to see how many swaps and comparisons are made between the different sorting algorithms and to gauge how efficient they are. Below you will see that even tho the heap sort is the generally the

most efficient. Having the almost the lowest number for swaps and for comparisons. Thus making it the best choice. But the heap sort did have an increased number of swaps for the almost sorted file. Adding a little bit to the processing time, but makes up for it with almost half the swaps as the next closest one.

Information is displayed in this format:

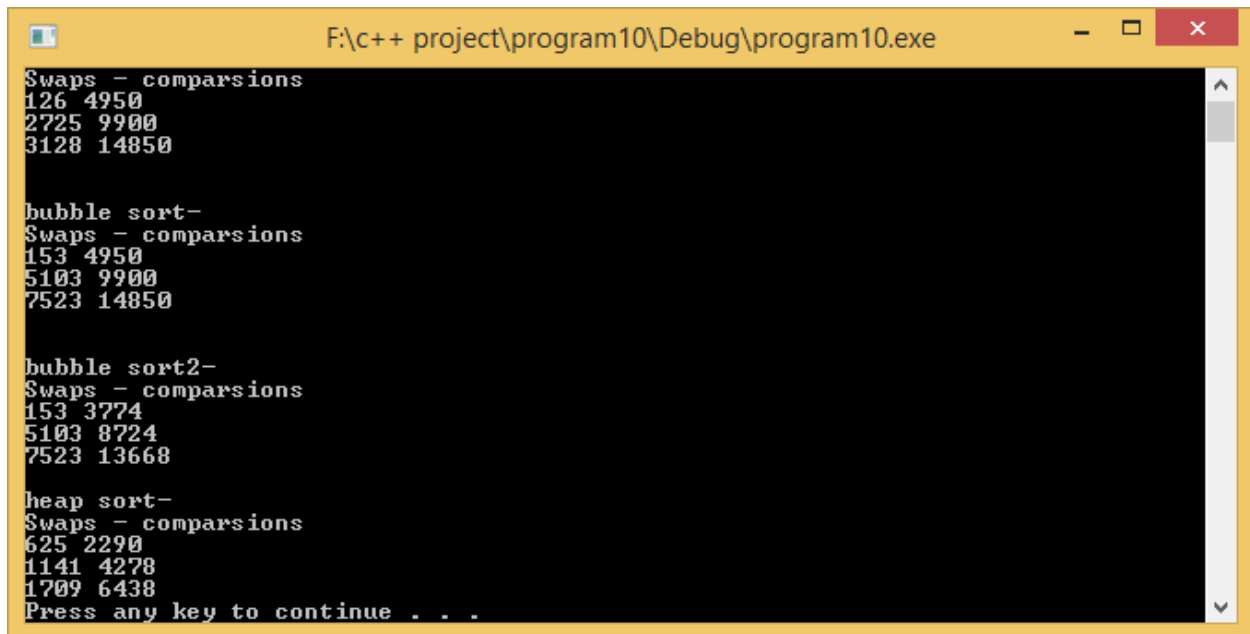
Swaps-Comparisons

File1: xxxxx xxxxx

File2: xxxxx xxxxx

File3: xxxxx xxxxx

File1 is almost sorted, file2 is counting down from 100, and file 3 the numbers are randomly placed.



```
F:\c++ project\program10\Debug\program10.exe

Swaps - comparisons
126 4950
2725 9900
3128 14850

bubble sort-
Swaps - comparisons
153 4950
5103 9900
7523 14850

bubble sort2-
Swaps - comparisons
153 3774
5103 8724
7523 13668

heap sort-
Swaps - comparisons
625 2290
1141 4278
1709 6438
Press any key to continue . . .
```

Below this is my nasty copy and past code I used to put this experiment together, enjoy!

```

#include <iostream>
#include <string>
#include <fstream>
#include "heapSort.cpp"
using namespace std;

// Function prototypes
void selectionSort(int array[], int elems);
void showArray(int array[], int elems);
void GetList(int[],int&,int,string);
int sl(void);
int bs1() ;
int bs2() ;
void sortArray2(int array[], int elems) ;
void sortArray(int array[], int elems) ;
int HPS() ;
int swaps = 0, comps = 0;

int main(void)
{
    sl();
    swaps = 0;
    comps = 0;
    bs1();
    swaps = 0;
    comps = 0;
    bs2();
    swaps = 0;
    comps = 0;
    HPS();
    system("pause");
    return 0;
}

// This program uses the selection sort algorithm to sort an
// array in ascending order.

// Function prototypes

const int MAX_LIST_SIZE = 100;    // Max list size

int sl(void)
{
    cout << "selection sort- \nSwaps - comparisons" << endl;
    int InList[MAX_LIST_SIZE];    // Array of list elements
    int ListSize;                  // Index of last list element

    int InList2[MAX_LIST_SIZE];    // Array of list elements

```

```

    int ListSize2;           // Index of last list element

    int InList3[MAX_LIST_SIZE]; // Array of list elements
    int ListSize3;           // Index of last list element

    GetList(InList,ListSize,MAX_LIST_SIZE, "Almost.txt");
    selectionSort(InList,100);
    cout<< swaps << " " << comps << endl;

    GetList(InList2,ListSize2,MAX_LIST_SIZE, "Inverse.txt");
    selectionSort(InList2,100);
    cout<< swaps << " " << comps << endl;

    GetList(InList3,ListSize3,MAX_LIST_SIZE, "Random.txt");
    selectionSort(InList3,100);
    cout<< swaps << " " << comps << "\n\n"<< endl;

    return 0;
}

void GetList(int inList[],int& listsize,int maxlistsize, string file)
{
    ifstream InFile (file);
    int ListElement;

    int i = 0;
    InFile >> ListElement;
    while(InFile && i < maxlistsize)
    {
        inList[i] = ListElement;
        i++;
        InFile >> ListElement;
    }
    listsize = i;    // Size of list
}

//*****
// Definition of function selectionSort.          *
// This function performs an ascending order selection sort on *
// array. elems is the number of elements in the array.      *
//*****
void selectionSort(int array[], int elems)
{
    int startScan, minIndex, minValue;

    for (startScan = 0; startScan < (elems - 1); startScan++)
    {
        swaps++;

        minIndex = startScan;
        minValue = array[startScan];

```

```

        for(int index = startScan + 1; index < elems; index++)
        {
            comps++;

            if (array[index] < minValue)
            {
                swaps++;

                minValue = array[index];
                minIndex = index;
            }
        }
        array[minIndex] = array[startScan];

        array[startScan] = minValue;
    }
}

```

```

int bs1()
{
    cout << "bubble sort- \nSwaps - comparisons" << endl;

    int InList[MAX_LIST_SIZE]; // Array of list elements
    int ListSize;              // Index of last list element

    int InList2[MAX_LIST_SIZE]; // Array of list elements
    int ListSize2;              // Index of last list element

    int InList3[MAX_LIST_SIZE]; // Array of list elements
    int ListSize3;              // Index of last list element

    GetList(InList,ListSize,MAX_LIST_SIZE, "Almost.txt");
    sortArray(InList,100);
    cout<< swaps << " " << comps << endl;

    GetList(InList2,ListSize2,MAX_LIST_SIZE, "Inverse.txt");
    sortArray(InList2,100);
    cout<< swaps << " " << comps << endl;

    GetList(InList3,ListSize3,MAX_LIST_SIZE, "Random.txt");
    sortArray(InList3,100);
    cout<< swaps << " " << comps << "\n\n"<<endl;

    return 0;
}

```

```

/*****
/* This function receives a list of values of type int    */
/* as an array. The function performs a bubble sort and */

```

```

/* returns the list in ascending order.          */
/* Note: elems is number of elements in list    */
/*****/
void sortArray(int array[], int elems)
{
    int temp, end;

    for (end = elems - 1; end >= 0; end--)
    {
        for (int count = 0; count < end; count++)
        {
            comps++;
            if (array[count] > array[count + 1])
            {
                swaps++;
                temp = array[count];
                array[count] = array[count + 1];
                array[count + 1] = temp;
            }
        }
    }
}

```

```

int bs2()
{
    cout << "bubble sort2- \nSwaps - comparsions" << endl;

    int InList[MAX_LIST_SIZE]; // Array of list elements
    int ListSize;              // Index of last list element

    int InList2[MAX_LIST_SIZE]; // Array of list elements
    int ListSize2;              // Index of last list element

    int InList3[MAX_LIST_SIZE]; // Array of list elements
    int ListSize3;              // Index of last list element

    GetList(InList, ListSize, MAX_LIST_SIZE, "Almost.txt");
    sortArray2(InList, 100);
    cout << swaps << " " << comps << endl;

    GetList(InList2, ListSize2, MAX_LIST_SIZE, "Inverse.txt");
    sortArray2(InList2, 100);
    cout << swaps << " " << comps << endl;

    GetList(InList3, ListSize3, MAX_LIST_SIZE, "Random.txt");
    sortArray2(InList3, 100);
    cout << swaps << " " << comps << endl;
}

```

```

        return 0;
    }

```

```

/*****
/* This function receives a list of values of type int */
/* as an array. The function performs a bubble sort and */
/* returns the list in ascending order. The sorting is */
/* halted if no swaps are made in any given pass. */
/* Note: elems is number of elements in list */
*****/
void sortArray2(int array[], int elems)
{
    bool swap;
    int temp;

    int end = elems - 1; // To control stopping point

    do
    {
        swap = false; // Assume no swap this pass
        for (int count = 0; count < end; count++)
        {
            comps++;
            if (array[count] > array[count + 1])
            {
                swaps++;
                temp = array[count];
                array[count] = array[count + 1];
                array[count + 1] = temp;
                swap = true; // Mark that swap occurred
            }
        }
        end--; // Move stopping point up

        // Continue if a swap occurred that pass
    } while (swap);
}

```

```

int HPS()
{
    cout << "\nheap sort- \nSwaps - comparisons" << endl;

    int InList[MAX_LIST_SIZE]; // Array of list elements
    int ListSize; // Index of last list element

    int InList2[MAX_LIST_SIZE]; // Array of list elements
    int ListSize2; // Index of last list element
}

```

```
int InList3[MAX_LIST_SIZE]; // Array of list elements
int ListSize3;              // Index of last list element

// HeapSort(InList,ListSize);

GetList(InList,ListSize,MAX_LIST_SIZE, "Almost.txt");
HeapSort(InList,100);

GetList(InList2,ListSize2,MAX_LIST_SIZE, "Inverse.txt");
HeapSort(InList2,100);

GetList(InList3,ListSize3,MAX_LIST_SIZE, "Random.txt");
HeapSort(InList3,100);
// Write list after sort
return 0;
}
```