**Algorithms and Theoretical Time Complexities**

Bubble Sort Algorithm

public static int[] bubbleSort(int n[]){

int array[]=n.clone();

int temp=0;

for(int i=0;i<array.length-1;i++){

for(int j=1;j<array.length-i;j++){

if(array[j-1]>array[j]){

temp=array[j-1];

array[j-1]=array[j];

array[j]=temp;

}

}

}

return array;

}

Theoretical Time Complexity: O()

Insertion Sort Algorithm

public static int[] InsertionSort(int n[]){

int array[]=n.clone();

int temp=0;

for(int i=1;i<array.length;i++){

temp=array[i];

int j;

for(j=i-1; j>=0 && temp<array[j];j--){

array[j+1]=array[j];

}

array[j+1]=temp;

}

return array;

}

Theoretical Time Complexity: O()

Selection Sort Algorithm

public static int[] SelectionSort(int n[]){

int array[]=n.clone();

int index=0;

int temp=0;

for(int i=0;i<array.length-1;i++){

index=i;

for(int j=i+1;j<array.length;j++){

if(array[j]<array[index]){

index=j;

}

}

temp=array[index];

array[index]=array[i];

array[i]=temp;

}

return array;

}

Theoretical Time Complexity: O()

**Methodology**

All three algorithms were combined into a single program that sorts an unsorted array. The program takes a user input to determine the size of the unsorted array to be fixed. The unsorted array is generated by the program itself and contains unique values. The unsorted array is sorted using each algorithm and the execution time in nanoseconds is taken note of. To obtain the execution time, the current time is taken before the function containing the algorithm is run. After the function is executed, the current time is also taken. The difference of the two time values taken will produce the execution time of the function. Three timestamps are computed independently of each other, to obtain the execution times of the three algorithms. The whole program is run a hundred times, and all the execution times taken are averaged to produce a running time that is fairly accurate. As an output, the program displays the average execution time of each sorting algorithm.

**Graphs and Findings**

|  |  |  |  |
| --- | --- | --- | --- |
| **Array Size** | **Execution time (in nanoseconds)** | | |
| **Bubble Sort** | **Insertion Sort** | **Selection Sort** |
| 10 | 8547 | 4478 | 6375 |
| 20 | 22509 | 10181 | 12685 |
| 30 | 38798 | 16104 | 24853 |
| 40 | 101284 | 67855 | 76312 |
| 50 | 81686 | 40026 | 49743 |
| 60 | 48975 | 33231 | 35350 |
| 70 | 100438 | 51327 | 76349 |
| 80 | 126979 | 80085 | 95459 |
| 90 | 62514 | 58910 | 42013 |
| 100 | 135473 | 61242 | 104897 |

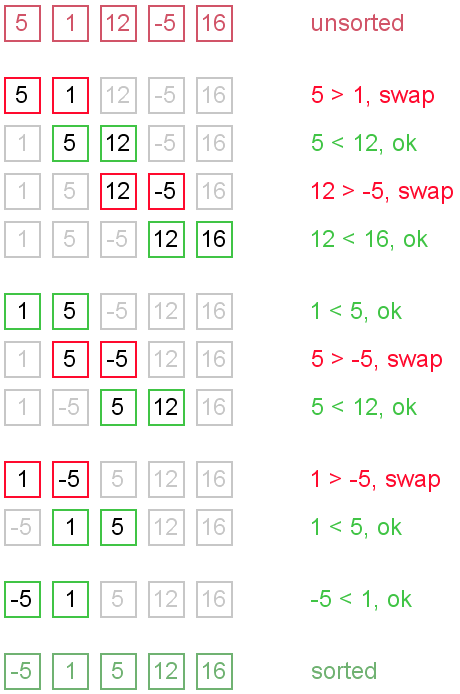
**Conclusions**

The Bubble Sort algorithm sorts values by swapping two values continuously (the smaller value goes on the left, while the larger value goes to the right), then moving to the next set of values until all values are completely sorted.

On the other hand, the Insertion Sort algorithm sorts values by starting with an array of size 2 to sort, then with an array size of 3 next, and so on. It works by sorting values inside the current array scope, which has a size initially set to the value 2, and after the values inside the current array scope are sorted, another value from the unsorted array is added to increase the size of the array scope. Adding and sorting values continuously occur until the scope covers the whole size of the unsorted array and all values are in place.

Lastly, the Selection Sort algorithm works by splitting the array to be sorted into two parts: sorted and unsorted. At the start of the algorithm, the sorted area is empty; while the unsorted area constitutes the whole array to be sorted. At the beginning of sorting, the smallest value in the unsorted area is taken and placed into the first space on the sorted area. The unsorted area’s size is then decreased by 1, and the smallest value from it is again taken to fill a space in the sorted area. This new value placed at the sorted area will increase the sorted area’s size by 1. Sorting continues until all elements in the unsorted area are transferred to the sorted area, resulting in a sorted array.

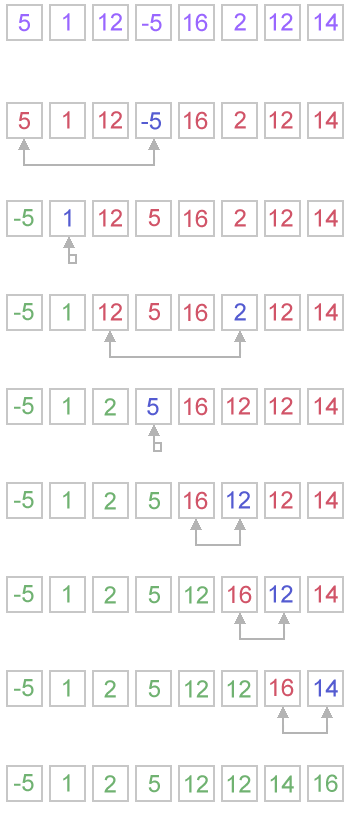
**Bubble Sort [2]**



**Insertion Sort [3]**



**Selection Sort [1]**



While all of the algorithms have a big-Oh of O(), their execution times vary because of the difference in their implementations. An unsorted array may have values that Bubble Sort can sort more quickly than Insertion Sort can, for example. If the array has values that can be sorted by simply swapping the values, the Bubble Sort algorithm will naturally have a greater execution time. If however, the smaller values are scattered all over the array to be sorted, the Selection sort or Insertion Sort will naturally have an easier time sorting the array than Bubble Sort can. The Bubble Sort works by swapping values until the whole array is sorted, and if for example the smallest value is placed at the end of the array, the algorithm will have to keep on swapping values until it realizes that the last value of the array is the smallest one. Once the smallest value is reached, it will naturally be swapped with the value on its left because it is smaller. Once swapping occurs, the algorithm will do another run on the array and will see that the smallest value is still not placed at the beginning of the array, causing value swapping to be done once again. Swapping or sorting will continuously occur until the smallest value is placed on the beginning of the array and all values are in their proper places. Contrary to the Insertion Sort and Selection Sort which sorts the array by looking for the smallest value inside the unsorted array first, and placing it at the beginning of the array immediately. Even though the Insertion Sort and Selection Sort are implemented differently, they behave the same in concept. Insertion Sort arranges the values in the array by increasing the area of the array to be sorted, while the Selection Sort divides the array into an unsorted and sorted area, then transferring values from each area. Because of this, both algorithms produce relatively similar execution times. As can be seen in the results of the program, the execution time of Insertion Sort is sometimes faster than that of Selection Sort and sometimes the inverse is true. However, the Bubble Sort algorithm always performed the worst among all the algorithms. While all algorithms have the same big-Oh, Bubble Sort implements its solution in such a way that it needs to go through the array more often compared to Insertion Sort and Selection Sort.

**Sources**

[1] "Selection Sort." *www.algolist.net*. Web. 25 Feb. 2015.

[2] "Bubble Sort." *www.algolist.net*. Web. 25 Feb. 2015.

[3] "Insertion Sort." *www.algolist.net*. Web. 25 Feb. 2015.