**Project 10 - Sorting**

**Name(s):** Mimi, Henry, Ozaner

**Problem:** This project will give you the opportunity to explore a few sorting algorithms. The particular algorithms that you will document and analyze will be determined by your group as given below. You may submit one project per group.

1. **Bubble sort**
2. **Shell sort**
3. **Merge sort**

For each of your assigned sorting algorithms, complete the following pages. Fill out your name(s) above and submit to your Google Drive Hand-in folder.

In addition to studying the supplied code, you may need to look up a description of the algorithm (in Wikipedia, for example) to help you determine how it works and what its time and space complexity.

**Name of sorting algorithm**: Bubble Sort

**Time Complexity (Big-O notation):** O(n2)

**Space Complexity (Big-O notation):** O(1)

**Explain how the sort algorithm works in your own words.**

The algorithm starts by comparing the first 2 elements and reordering them if necessary. It then does the same for the second and third element and so on until it reaches the last 2 elements. After this first run-through it is certain the the last number is the highest and is therefore sorted. As a result the second run-through can ignore the second to last element and third run-through can ignore the third to last element until the last run through which only compares the second and third elements. An array of length n will take n-2 run-throughs to be completely sorted.

**Explain how the sort algorithm would sort {12, 3, 6, 19, 4, 8}**

* Compare 1st and 2nd elements, 12 and 3, move larger number in front.**{3, 12, 6, 19, 4, 8}**
* Compare 2st and 3rd elements, 12 and 6, move larger number in front.**{3, 6, 12, 19, 4, 8}**
* Compare 3rd and 4th elements, 12 and 19, move larger number in front.**{3, 6, 12, 19, 4, 8}**
* Compare 4rd and 5th elements, 19 and 4, move larger number in front.**{3, 6, 12, 4, 19, 8}**
* Compare 5rd and 6th elements, 19 and 8, move larger number in front.**{3, 6, 12, 4, 8, 19}**
* Run through list again but only go up to 5th element. **{3, 6, 4, 8, 12, 19}**
* Run through list again but only go up to 4th element. **{3, 4, 6, 8, 12, 19}**
* It is now sorted but will run through 2 more times.
* Run through list again but only go up to 3rd element. **{3, 4, 6, 8, 12, 19}**
* Run through list again but only go up to 2nd element. **{3, 4, 6, 8, 12, 19}**

**Copy and paste the sort() method for the sort algorithm here:**

public void sort() {

int temp;

do {

temp = a[0];

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

if (a[i-1] > a[i]) {

temp = a[i];

a[i] = a[i-1];

a[i-1] = temp;

}

}

} while (temp != a[0]);

}

**Name of sorting algorithm**: Merge Sort

**Time Complexity (Big-O notation):** O(n log(n))

**Space Complexity (Big-O notation):** O(n)

**Explain how the sort algorithm works in your own words.**

The algorithm splits the original array in half into two new arrays and sorts those arrays. Then it compares the first item in each of the two sorted arrays and picks the item that should come first and places it into the original array. It does this until no items remain in either of the arrays, and when this occurs the array is sorted.

**Explain how the sort algorithm would sort {12, 3, 6, 19, 4, 8}**

1. It would split it into two arrays

{12, 3, 6} and {19,4,8}

2. It sorts each array

{3, 6, 12} and {4, 8, 19}

3. It compares the first item in each array that has not been selected and puts the one that should come first into the original array. It continues to do this until no items remain

{3, 6, 12} and {4, 8, 19}

3<4 {3, \_, \_, \_, \_, \_}

6>4 {3, 4, \_, \_, \_, \_}

6<8 {3, 4, 6, \_, \_, \_}

12>8 {3, 4, 6, 8, \_, \_}

12<19 {3, 4, 6, 8, 12, \_ }

19 {3, 4, 6, 8, 12, 19}

**Copy and paste the sort() method for the sort algorithm here:**

public void sort()

{

if (a.length <= 1) return;

int[] first = new int[a.length / 2];

int[] second = new int[a.length - first.length];

System.arraycopy(a, 0, first, 0, first.length);

System.arraycopy(a, first.length, second, 0, second.length);

MergeSorter firstSorter = new MergeSorter(first);

MergeSorter secondSorter = new MergeSorter(second);

firstSorter.sort();

secondSorter.sort();

merge(first, second);

}

**Name of sorting algorithm**: Shellsort

**Time Complexity (Big-O notation):** O(nlog(n))

**Space Complexity (Big-O notation):** O(n)

**Explain how the sort algorithm works in your own words.**

The algorithm starts by determining a value for the int h, the element at which the searching will begin. In a for loop, all of the elements from h to the end of the array are compared to the element at the index k - h, k being the current value of the for loop counter. As long as this element is greater than the current element being examined in the for loop, the current element will be replaced by the other element and k will become k - h. This will continue until k becomes less than h when the the element at k will be replaced by the original element being searched.

**Explain how the sort algorithm would sort {12, 3, 6, 19, 4, 8}**

1. It compares the 1st and 5th elements (12 and 4). Since the 1st is greater than the 5th, it switches their order in the array: **{4, 3, 6, 19, 12, 8}**
2. Compares the 2nd and 6th elements (3 and 8), 6th is greater than 2nd, nothing changes: **{4, 3, 6, 19, 12, 8}**
3. Compares the 1st and 2nd elements (4 and 3), 1st is greater than 2nd, switches their order: **{3, 4, 6, 19, 12, 8}**
4. Compares the 2nd and 3rd elements (4 and 6), 3rd is greater than 2nd, nothing changes: **{3, 4, 6, 19, 12, 8}**
5. Compares the 3rd and 4th elements (6 and 19), 4th is greater than 3rd, nothing changes: **{3, 4, 6, 19, 12, 8}**
6. Compares the 4th and 5th elements (19 and 12), 4th is greater than 5th, 5th element is set to 19: **{3, 4, 6, 19, 19, 8}**
7. Since k is greater than h, the while loop repeats again. The 3rd element (6) is compared to the temp, which is still equal to 12. The temp is greater than the 3rd element, nothing is changed. After the while loop, the 4th element is set equal to 12: **{3, 4, 6, 12, 19, 8}**
8. Compares the 5th and 6th elements (19 and 8), 5th is greater than 6th, 6th element is set to 19: **{3, 4, 6, 12, 19, 19}**
9. Since k is greater than h, the while loop repeats again. The 4th element (12) is compared to the temp, which is still equal to 8. The 4th element is greater than the temp, so the 5th element is set to 12: **{3, 4, 6, 12, 12, 19}**
10. k is still greater than h, so the while loop repeats again. The 3rd element (6) is compared to the temp, which is still 8. The temp is greater than the 3rd element, nothing is changed. After the loop ends, the 4th element is set to 8: **{3, 4, 6, 8, 12, 19}**

**Copy and paste the sort() method for the sort algorithm here:**

public void sort() {

int h = 1;

int temp;

int k;

do {

h = 3 \* h + 1;

} while (h <= a.length);

do {

h /= 3;

for (int j = h; j < a.length; j++) {

temp = a[j];

k = j;

while (a[k - h] > temp) {

a[k] = a[k - h];

k -= h;

if (k < h)

break;

}

a[k] = temp;

}

} while (h != 1);

}

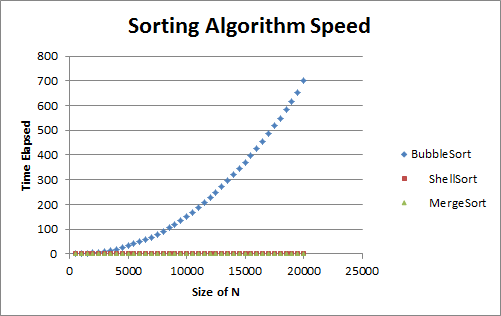
private int[] a;

}

Using the techniques illustrated in the SortingStudy.java program, run your algorithms 5 times each on arrays of size 500 to 20,000 elements with an increment of 500. Each array element is between 0 and 99. Fill in this table with your sorting method names and timing results.

|  |  |  |  |
| --- | --- | --- | --- |
| n | BubbleSort | ShellSort | MergeSort |
| 500 | 0 | 0 | 0 |
| 1000 | 1 | 0 | 0 |
| 1500 | 3 | 0 | 0 |
| 2000 | 4 | 0 | 0 |
| 2500 | 7 | 0 | 0 |
| 3000 | 10 | 0 | 0 |
| 3500 | 14 | 0 | 0 |
| 4000 | 19 | 0 | 0 |
| 4500 | 25 | 0 | 1 |
| 5000 | 32 | 0 | 0 |
| 5500 | 40 | 0 | 0 |
| 6000 | 50 | 0 | 0 |
| 6500 | 57 | 0 | 0 |
| 7000 | 67 | 0 | 0 |
| 7500 | 79 | 0 | 0 |
| 8000 | 92 | 0 | 1 |
| 8500 | 105 | 0 | 1 |
| 9000 | 119 | 0 | 1 |
| 9500 | 136 | 0 | 1 |
| 10000 | 151 | 0 | 1 |
| 10500 | 169 | 0 | 1 |
| 11000 | 188 | 0 | 1 |
| 11500 | 207 | 0 | 1 |
| 12000 | 228 | 1 | 1 |
| 12500 | 249 | 1 | 1 |
| 13000 | 272 | 1 | 1 |
| 13500 | 295 | 1 | 1 |
| 14000 | 319 | 1 | 1 |
| 14500 | 344 | 1 | 1 |
| 15000 | 371 | 1 | 1 |
| 15500 | 398 | 1 | 2 |
| 16000 | 426 | 1 | 1 |
| 16500 | 456 | 1 | 2 |
| 17000 | 488 | 1 | 2 |
| 17500 | 518 | 1 | 2 |
| 18000 | 548 | 1 | 2 |
| 18500 | 584 | 1 | 2 |
| 19000 | 616 | 1 | 2 |
| 19500 | 653 | 1 | 2 |
| 20000 | 702 | 1 | 2 |

Produce an Excel scatterplot showing your results and paste it below. Discuss how the plot supports your earlier time analysis.



The result of Bubble Sort supports its time complexity being O(n2)because as it tries to sort more and more items in the array, the time it takes to sort it increases at a very fast rate. Shell Sort and Merge Sort results support their time complexity of nlog n because the results for both are very similar and time elapsed barely increases as the size of n increases.