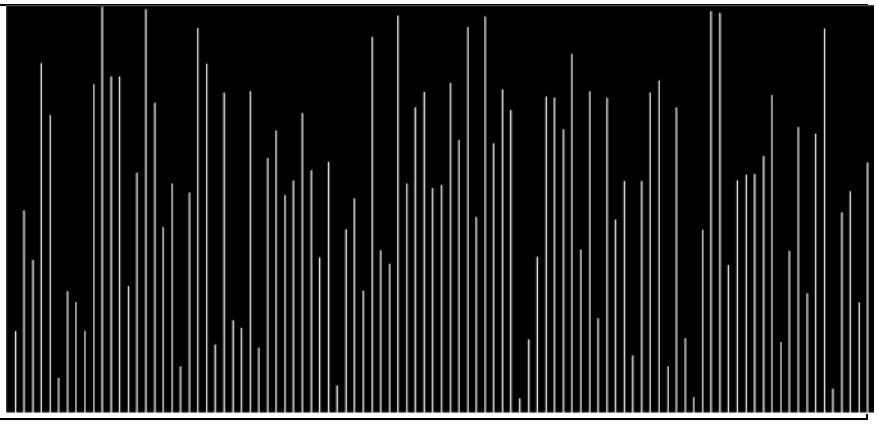
Name(s)_____

Sorting Algorithms – Visualization & Growth (part I)

Sometimes visualizing an algorithm helps us understand how it works and makes it easier to remember the algorithm. It also may help us understand how much more work the algorithm needs to do if we increase the size of the problem.

In this activity you are going to be making some predictions about the "shape" of some sorting algorithms as they are executing on data. You will also make some predictions about how many more comparisons of items are needed if you were to double the input size. Always assume we are sorting in increasing order.



You can assume we start with a set of random data as shown. The height of each white vertical bar represents a random integer between 0 and 500. There are 100 bars shown.

1. Bubble Sort – Start by comparing adjacent items from the end (item(n) and item(n-1), item(n-1) and item(n-2), ... , item3 and item 2, item2 and item1), swapping if the pair is out of order. After going through all the pairs once, we then restart at the end again and only need to go through n-1 items. Then start at the end again and only need to go through n-2 items. etc.

1a. Make a prediction of what the collection will look like after the first 10 iterations by drawing a rough picture in the style of the above.

1b. Make a prediction of what the collection will look like after the first 50 iterations by drawing a rough picture in the style of the above.

1c. Estimate the total number of comparisons necessary to sort 10 random items.

1d. Do you think the total number of comparisons necessary to sort 20 random items will be closer to 2 times your 10 item estimate or closer to 4 times your 10 item estimate?

2. Selection Sort – Locate the smallest item out of all n items by examining each item while keeping track of the location where the smallest item value found thus far is located (minimumPosition). When done looking through items once, swap the first item with the minimumPosition item. Then start at the second item and find the smallest remaining item for the (n-1) items, and then swap it with item 2. etc.

2a. Make a prediction of what the collection will look like after the first 10 iterations by drawing a rough picture in the style of the above.

2b. Make a prediction of what the collection will look like after the first 50 iterations by drawing a rough picture in the style of the above.

2c. Estimate the total number of comparisons necessary to sort 10 random items.

2d. Do you think the total number of comparisons necessary to sort 20 random items will be closer to 2 times your 10 item estimate or closer to 4 times your 10 item estimate?

3. Merge Sort – A divide and conquer (and combine) approach, like the intelligent algorithmic approach to the number guessing game. The first key idea is that a list of length one element is sorted, this will be used to end the “divide” steps. The second key idea is the “combine” step; you can merge two sorted lists of total length n in n comparisons.

Merge Sort divides the list in half, and then first it concentrates (we’ll come back to what this means) on the first half. After that first half is sorted, it concentrates (we’ll come back to what this means) on the second half. After the second half is sorted is uses the “merge” mentioned above to merge the sorted first half with the sorted second half resulting in a list with all the items sorted.

In the previous paragraph when it said “it concentrates” it meant that it calls Merge Sort again on that sublist. Eventually when you call Merge Sort on a list with only one item, it just returns the list (see first key idea in paragraph 1).

3a. Make a prediction of what the collection will look like when it is one quarter done by drawing a rough picture in the style of the above.

3b. Make a prediction of what the collection will look like when it is half way done by drawing a rough picture in the style of the above.

3c. Do you expect the total number of comparisons necessary to sort 10 random items to be higher or lower than the previous sorts? Note all the comparisons are done in the merges.

3d. Do you expect the **growth** in the total number of comparisons necessary to sort 20 random items to be higher or lower than the previous sorts?