

Lecture 8

Simple Sorting
Bubble Sort, Selection Sort, and Insertion Sort

Last time:

We looked at Queue:
- A limited data structure that has two major operations, enqueue and dequeue.
- Enqueue inserts a new element into the back of the queue and dequeue **gets and removes** the first element from the queue.
- Enqueue and dequeue's time complexity is O(1).
- Similar to Stack, it can also be used in applications as an aiding tool for a programmer.
- Java's LinkedList and ArrayDeque provide a few methods that enable programmers to use them as Queue/Deque.
- Example usage of a queue is serving people in line or queue at a coffee shop or printing jobs.

Let's shift gears for a while!

So far, we have seen some data structures: arrays, ArrayList, LinkedList, Stack and Queue.

As we saw in lecture 3, suppose that I put all of your names (first name and last name) and Andrew id or student id in an array to have my roster.

Initially, I put them into an array without any order.
Now, it is time to grade your first homework. Oops! Well, I want to sort you by student id or Andrew id.

The question is "**how would you do it?**"
A computer program is NOT able to see the big picture. Thus, it needs to concentrate on the details and follow preset rules.

Three sorting algorithms in this lecture mainly involve two simple steps executed over and over till the items are sorted:

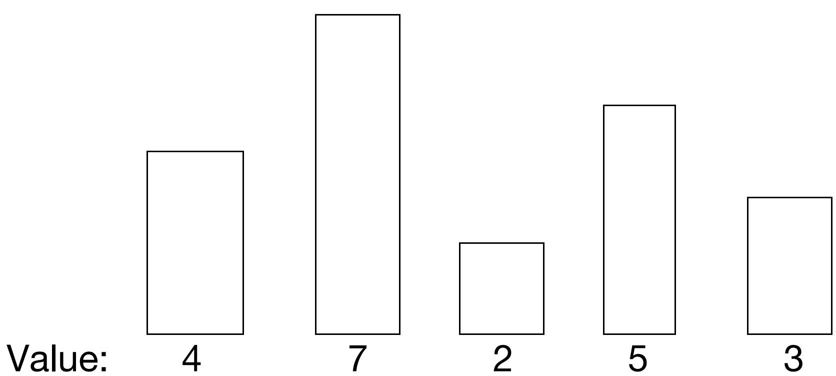
- Compare two items
- Swap the two items, or copy one item or items.

Bubble Sort: Very slow but simple

Step 1: Conceptual View
There are three basic steps in bubble sort.

- Compare two values at a time.
- If the one on the left is larger, swap them to ***BUBBLE UP the larger value to the right***.
- Move one position to the right.

Initial state



After first round

Number of comparisons? ___________
Number of swaps? ___________

After second round

Number of comparisons? ______________
Number of swaps? _____________
Now, do you see a pattern?

After third round

Number of comparisons? ______________
Number of swaps? _____________

After fourth round

Number of comparisons? ______________
Number of swaps? _____________

Step 2: Implementation View

`int[] data = { 4, 7, 2, 5, 3 };`

Swap method

```
// a helper method that swaps two values in an int array
private static void swap(int[] data, int one, int two) {
    int temp = data[one];
    data[one] = data[two];
    data[two] = temp;
}
```

Comparison method for one round

```
// go forward from 0 to the end of the array
for (int in = 0; in < ______________________; in++) {
    if (data[in] > data[in+1]) { // if the left value is bigger
        swap(____________, ___________, ___________); // then swap
    }
}
```

Multiple rounds

```
// go backward from the last index till ________
for (int out = _____________________; out >= _________; out--) {
    for (int in = 0; in < __________________; in++) {
        if (data[in] > data[in+1]) { // if left value is bigger
            swap(__________, ______________, __________);
        }
    }
}
```

Finally, putting all the pieces together

```
public static void bubbleSort(int[] data) {
    for (int out = ________________; out >= ___________; out--) {
        for (int in = 0; in < ___________________; in++) {
            if (data[in] > data[in+1]) {
                swap(____________, ___________, ___________);
            }
        }
    }
}
```

Results (Let's trace!)
First round (out: _____)
4,7,2,5,3 (in: ______, in+1: ______, swap?: ________)
4,2,7,5,3 (in: ______, in+1: ______, swap?: ________)
4,2,5,7,3 (in: ______, in+1: ______, swap?: ________)
4,2,5,3,7 (in: ______, in+1: ______, swap?: ________)

Second round (out: _____)
4,2,5,3,7 (in: ______, in+1: ______, swap?: ________)
2,4,5,3,7 (in: ______, in+1: ______, swap?: ________)
2,4,3,5,7 (in: ______, in+1: ______, swap?: ________)

Third round (out: _______)
2,4,3,5,7 (in: ______, in+1: ______, swap?: ________)
2,3,4,5,7 (in: ______, in+1: ______, swap?: ________)

Fourth round (out:______)
2,3,4,5,7 (in: ______, in+1: ______, swap?: ________)

Invariants
Conditions that remain unchanged as the algorithm proceeds.

What is the invariant in bubbleSort?
Values after "out" are sorted.

Time complexity
For 5 items, there are 4 comparisons on the first pass, 3 comparisons on the second pass, and so on, which makes 10 comparisons total:
4+3+2+1 = 10
(n-1) + (n-2) + (n-3) + … + 1 = n*(n-1) / 2

We can say that the bubble sort algorithm makes about n^2 / 2 comparisons. Also, keep in mind that there are swaps: n^2 / 4 on average.
When is the case that we need to swap after every comparison on every pass?
Because constants don't count in Big O notation, we can conclude that bubble sort runs in O(n^2) time.

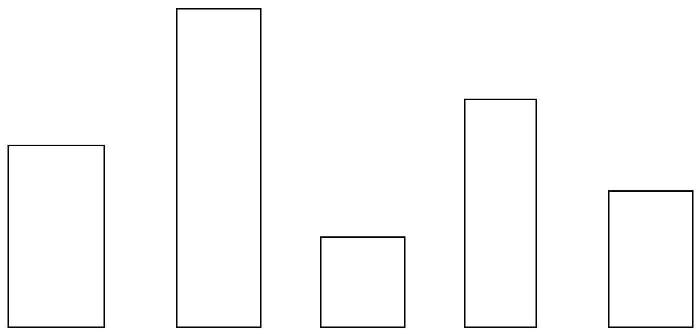
Selection Sort: Faster than Bubble Sort but still not enough

Step 1: Conceptual View

There are two steps in selection sort:

- ***Pick or SELECT*** the minimum value
- Swap it with the element on the left end

Initial State



Value: 4 7 2 5 3

After first round

Number of comparisons? _______
Number of swaps? ________

After second round

Number of comparisons? ________
Number of swaps? ____________
Now, do you see a pattern?

After third round

Number of comparisons? ________
Number of swaps? ____________

After fourth round

Number of comparisons? ________
Number of swaps? ____________

On each pass, where are the items that are sorted?
Where was it in Bubble Sort?

Step 2: Implementation View

```
int[] a = { 4, 7, 2, 5, 3 };
```

Swap method

```
// a helper method that swaps two values in an int array
private static void swap(int[] data, int one, int two) {
    int temp = data[one];
    data[one] = data[two];
    data[two] = temp;
}
```

Select the minimum

```
int min; // min variable
for (int out = 0; out < ________________________; out++) {
    min = ___________; // set initial min value's index

    // Select a new minimum value's index
    for (int in = __________; in < ______________; in++) {
        if (data[in] < data[min]) { // if new min value found
            min = in;  // reset the new min index
        }
    }

    swap(__________, __________, min); // time to put min value.
}
```

Finally, putting all the pieces together

```
public static void selectionSort(int[] data) {
    int min; // min variable
    for (int out = 0; out < ___________________________; out++) {
        min = _____________; // set initial min value's index

        // Select a new minimum value's index
        for (int in = ___________; in < _________________; in++) {
            if (data[in] < data[min]) { // if new min value
                 min = in;  // reset the new min index
            }
        }

        swap(_________, _________, min); // time to put min value.
    }
}
```

Results (Let's trace!)
First round (out: _____, min:______)
4,7,2,5,3 (in: ______, min: ______)
4,7,2,5,3 (in: ______, min: ______)
4,7,2,5,3 (in: ______, min: ______)
4,7,2,5,3 (in: ______, min: ______)
swap(______, ______)
2,7,4,5,3

Second round (out: _____, min: ______)
2,7,4,5,3 (in: ______, min: ______)
2,7,4,5,3 (in: ______, min: ______)
2,7,4,5,3 (in: ______, min: ______)
swap(______, ______)
2,3,4,5,7

Third round (out: _______, min:______)
2,3,4,5,7 (in: ______, min: ______)
2,3,4,5,7 (in: ______, min: ______)
swap(______, ______)
2,3,4,5,7

Fourth round (out:______, min:______)
2,3,4,5,7 (in: ______, min: ______)
swap(______, ______)
2,3,4,5,7

Invariants
The elements less than ____________ variable are sorted.

Time Complexity
First of all, do you see any improvement?

Is the number of comparisons in the selection sort the same as the bubble sort?

How about number of swaps?
What is the time complexity?

Insertion Sort: Best among the three!

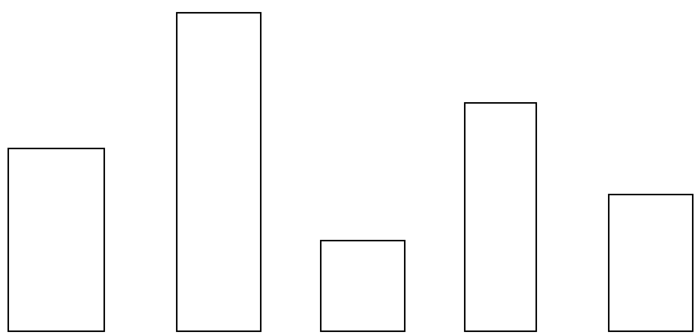
Step 1: Conceptual View

To me, this is the *most intuitive* sorting algorithm.

Most important thing in the insertion sort is that there is ***an imaginary dividing line.***

- Left hand side of the line is sorted among themselves
- The first element of the right hand side of the line should be inserted into the left hand side in a proper position
    - First, we keep the value of the first element into a temp place
    - Shift the items of the left hand side to the right so that there can be a space for the value that is stored in the temp place
    - When the position is found***, INSERT the value into*** that position

Initial State



Value: 4 7 2 5 3

After first round

Where is the dividing line? ________

Which value should be kept in the temporary place? __________

After second round

Where is the dividing line? ________
Which value should be kept in the temporary place? __________

After third round

Where is the dividing line? ________
Which value should be kept in the temporary place? __________

After fourth round

Where is the dividing line? ________
Which value should be kept in the temporary place? __________

Step 2: Implementation View

```
int[] a = { 4, 7, 2, 5, 3 };
```

```
public static void insertionSort(int[] data) {
    // set and increase the dividing line
    for (int out = _________; out < ___________________; out++) {
        int temp = data[out]; // store the value into temp
        int in = out;

        // go backward in the left side of the imaginary line
        // to find a place to insert temp value
        while (___________ && ________________________________) {
            data[in] = data[in-1];
            in--;
        }

        data[_____________] = temp; // INSERT the temp value
    }
}
```

Results (Let's trace!)
First round (out: _______, tmp: ______)
4, ,2,5,3 (in: ______, in-1:______, shift?:_______)
insert
4,7,2,5,3

Second round (out: _______, tmp: ______)
4,7, ,5,3 (in: ______, in-1:______, shift?: ______)
4, ,7,5,3 (in: ______, in-1:______, shift?: ______)
 ,4,7,5,3
insert
2,4,7,5,3

Third round (out: _______, tmp:______)
2,4,7, ,3 (in: ______, in-1: ______, shift?: ______)
2,4, ,7,3 (in: ______, in-1: ______, shift?: ______)
insert
2,4,5,7,3

Fourth round (out:_______, tmp:_______)
2,4,5,7, (in: ______, in-1: ______, shift?: _______)
2,4,5, ,7 (in: ______, in-1: ______, shift?: _______)
2,4, ,5,7 (in: ______, in-1: ______, shift?: _______)
2, ,4,5,7 (in: ______, in-1: ______, shift?: _______)
insert
2,3,4,5,7

Invariants
At the end of each round, the elements less than __________ variable are sorted ***AMONG THEM***.

Time Complexity
First of all, do you see any improvement?

Is the number of comparisons in the insertion sort the same as the bubble sort? (Hint: how far do you go back to actually insert each round?)

Do you notice anything different in terms of swapping?

What is the running time complexity?

There is a time that insertion sort can run even in O(n) time. Can you think of it?

Now, how about the opposite case of the previous situation? Would it be faster than the bubble sort?

These three internal sorting algorithms, bubble sort, selection sort and insertion sort, all run in O(n^2) time in the worst case.

However, often, insertion sort performs better than the other two because it may require less number of comparisons depending on the input values and uses copying instead of swapping.