Introduction to Sorting Algorithms

The Sorting Problem

- **Input:**
  - A sequence of n numbers $a_1, a_2, \ldots, a_n$
- **Output:**
  - A permutation (reordering) $a_1', a_2', \ldots, a_n'$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$

Reasons to Sort…

General technique in computing:

Preprocess the data to make subsequent operations (not just ADTs) faster

Example: Sort the data so that you can

- Find the k^{th} largest/smallest in constant time for any k
- Perform binary search to find elements in logarithmic time

Sorting's benefits depend on

- How often the data will change
- How much data there is

Real World versus Computer World

Sorting is a very general demand when dealing with data—we want it in some order

- Alphabetical list of people
- List of countries ordered by population

Moreover, we have all sorted in the real world

- Some algorithms mimic these approaches
- Others take advantage of computer abilities

Sorting Algorithms have different asymptotic and constant-factor trade-offs

- No single “best” sort for all scenarios
- Knowing “one way to sort” is not sufficient

A Comparison Sort Algorithm

We have *n* comparable elements in an array, and we want to rearrange them to be in *increasing order*

Input:

- An array A of data records
- A key value in each data record (maybe many fields)
- A comparison function (must be consistent and total): Given keys a and b is a<b, a=b, a>b?

Effect:

- Reorganize the elements of A such that for any i and j such that if i < j then $A[i] \leq A[j]$
- Array A must have all the data it started with

Arrays? Just Arrays?

The algorithms we will talk about will assume that the data is an array

- Arrays allow direct index referencing
- Arrays are contiguous in memory

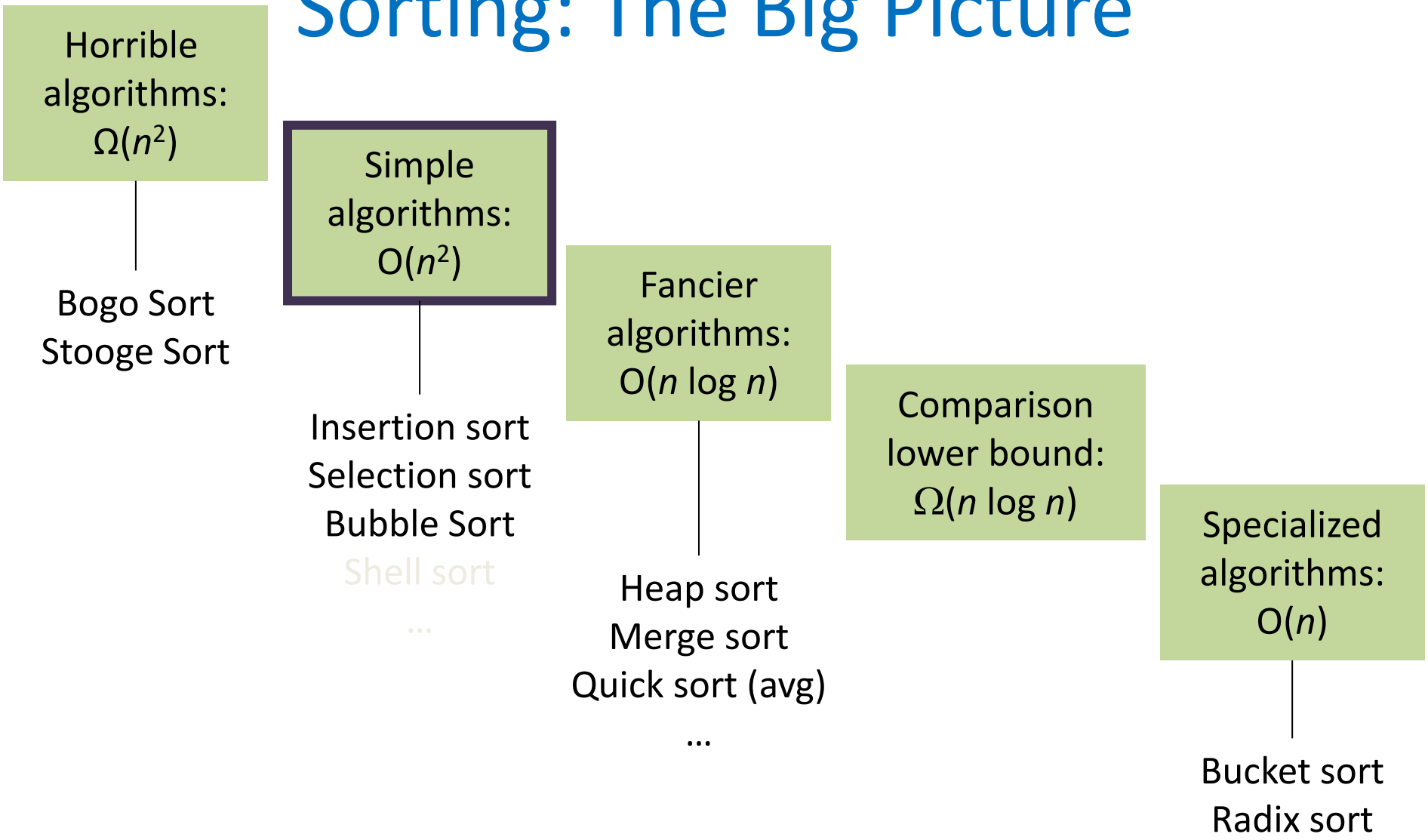
But data may come in a linked list

- Some algorithms can be adjusted to work with linked lists but algorithm performance will likely change (at least in constant factors)

Everyone and their mother's uncle's cousin's barber's daughter's friend has made a sorting algorithm

STANDARD COMPARISON SORT ALGORITHMS

Sorting: The Big Picture



Brute Force Sorting Algorithms - Selection Sort

Selection Sort Scan the array to find its smallest element and swap it with the first element. Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second elements. Generally, on pass i ($0 \le i \le n$-2), find the smallest element in $A[i..n\text{-}1]$ and swap it with $A[i]$:

$$A[0] \le \ldots \le A[i\text{-}1] \mid A[i], \ldots, A[min], \ldots, A[n\text{-}1]$$

in their final positions

An example follows…

Selection sort example

- Initial array:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	22	18	12	-4	27	30	36	50	7	68	91	56	2	85	42	98	25

- After 1st, 2nd, and 3rd passes:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	**-4**	18	12	**22**	27	30	36	50	7	68	91	56	2	85	42	98	25

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	**2**	12	22	27	30	36	50	7	68	91	56	**18**	85	42	98	25

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	**7**	22	27	30	36	50	**12**	68	91	56	18	85	42	98	25

Selection Sort in Java

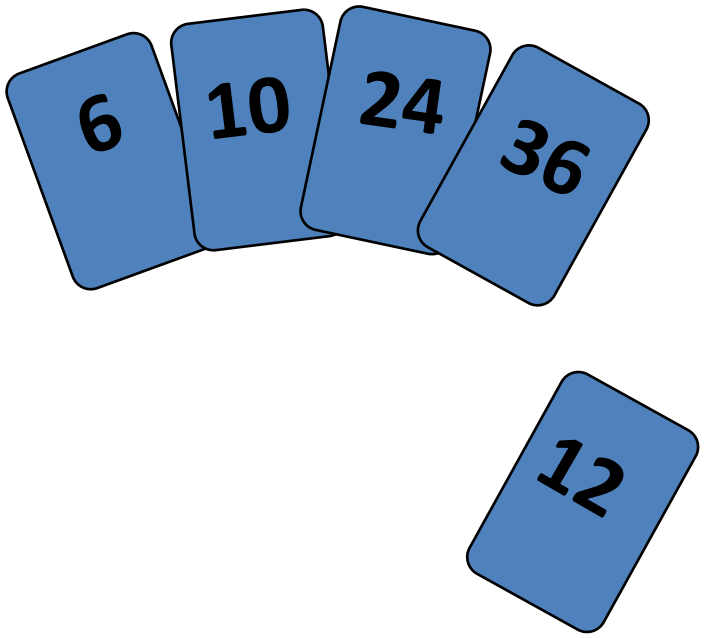
```
public class SelectionSortExample {
   public static void selectionSort(int[] arr){
      for (int i = 0; i < arr.length - 1; i++)
      {
         int min_index = i;
         for (int j = i + 1; j < arr.length; j++){
            if (arr[j] < arr[min_index]){
               min_index = j;//searching for
lowest index
            }
         }
         int smallerNumber = arr[min_index];
         arr[min_index] = arr[i];
         arr[i] = smallerNumber;
      }
   }
```

```
public static void main(String a[]){

      int[] arr1 = {27, 9, 6, 31, 0, -3, 4, 14};
      System.out.println("Before Selection
Sort");

      for(int i:arr1){
         System.out.print(i+" ");
      }
      System.out.println();

      selectionSort(arr1);//sorting array
using selection sort

      System.out.println("After Selection
Sort");

      for(int i:arr1){
         System.out.print(i+" ");

      }
   }
}
```

Analysis of Selection Sort

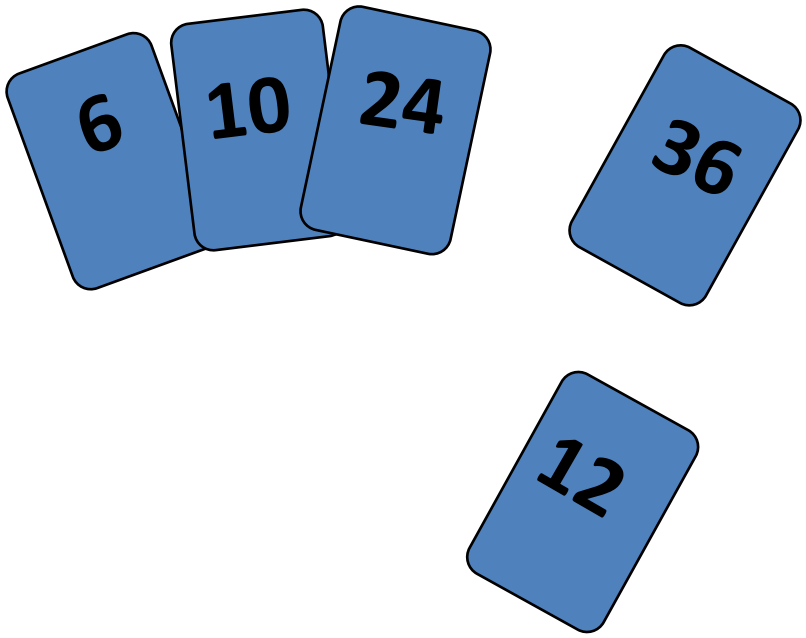
- Comparisons: $\approx n^2/2$
- Exchanges: $\approx n$
- $T(n) = O(n^2)$

Insertion Sort

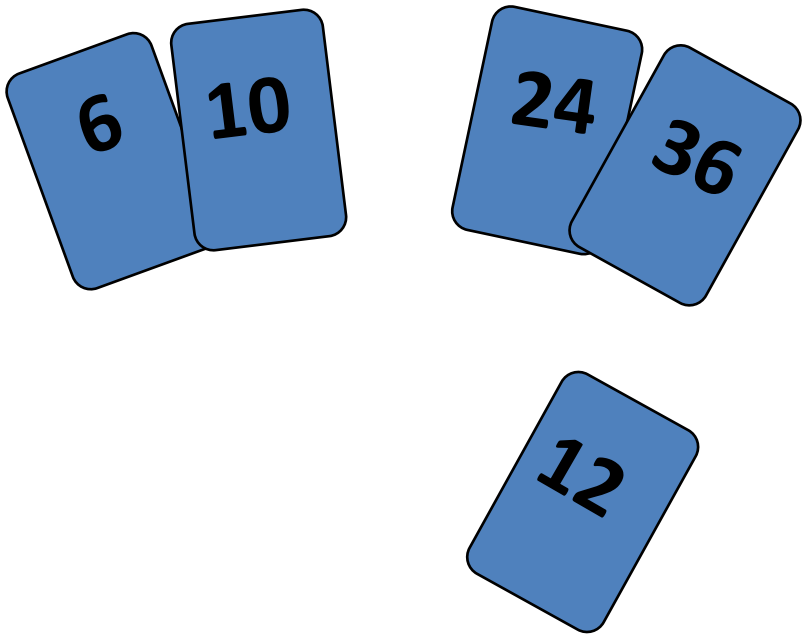


To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



Insertion Sort



Insertion Sort

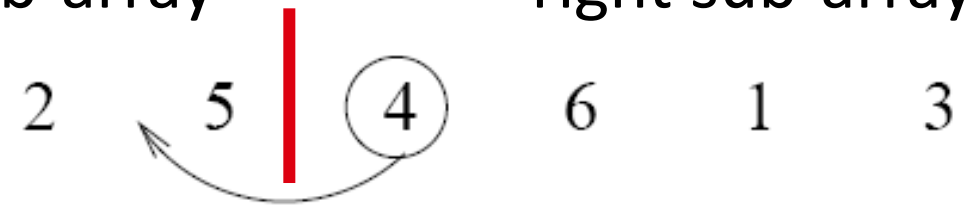
Invented by John Mauchly as early as 1946

Input array

5 2 4 6 1 3

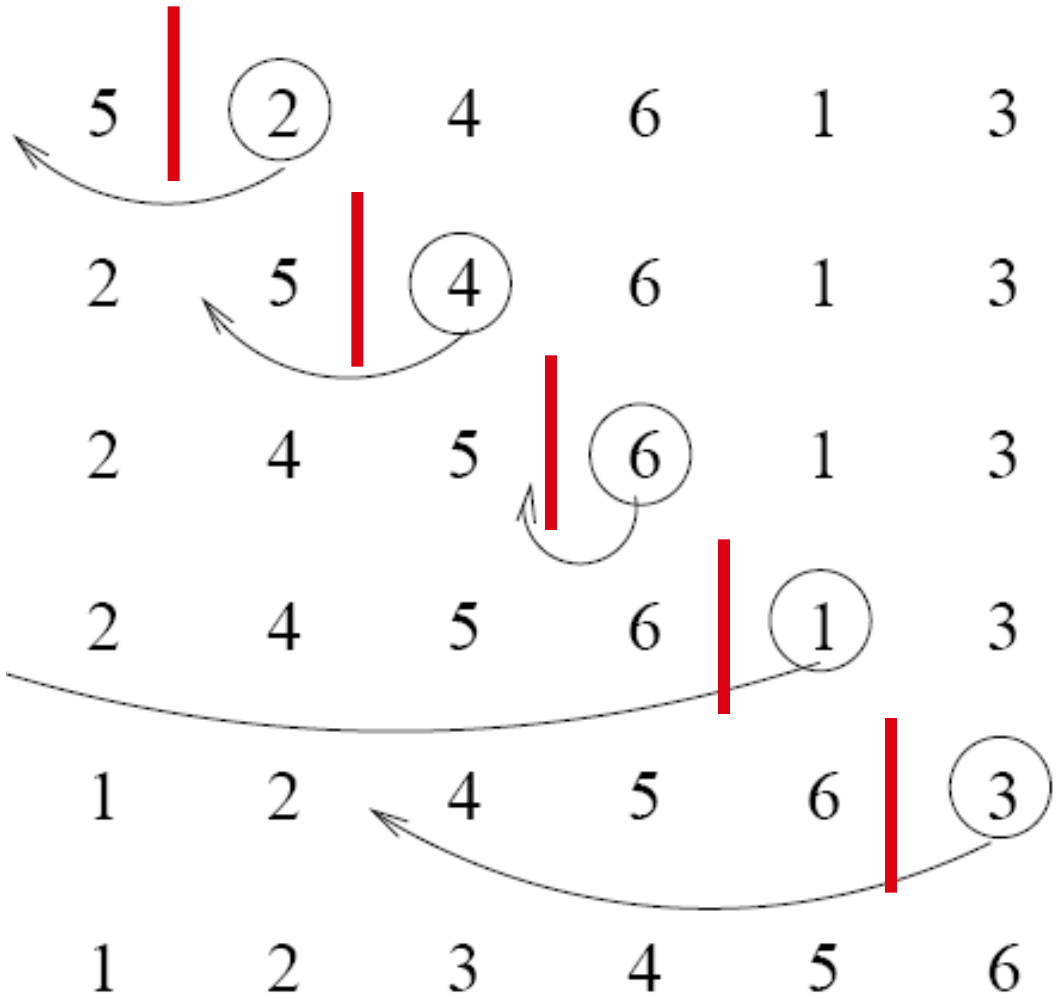
at each iteration, the array is divided in two sub-arrays:

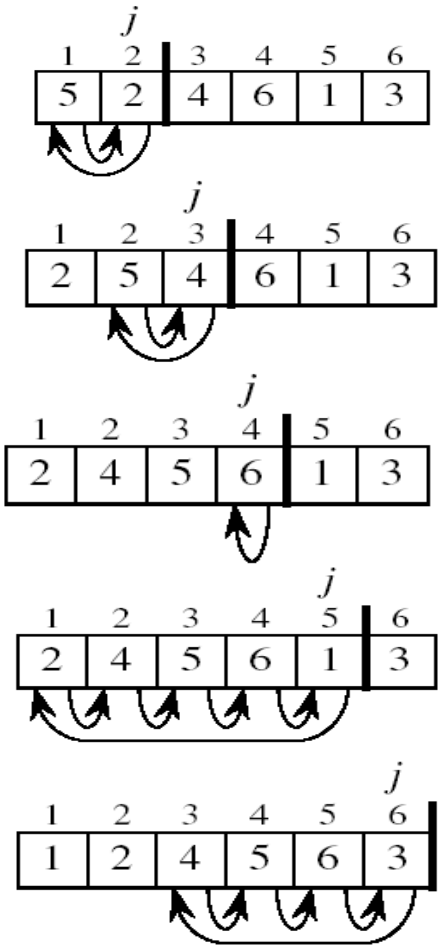
left sub-array | right sub-array



sorted | unsorted

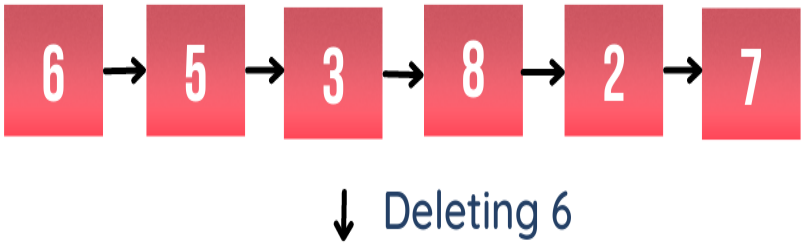
Insertion Sort





Insertion Sort using Linked List

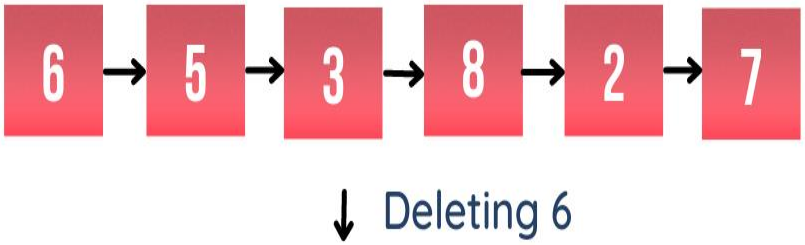
Current List



Sorted Set

Initially Sorted Set is Empty

Current List



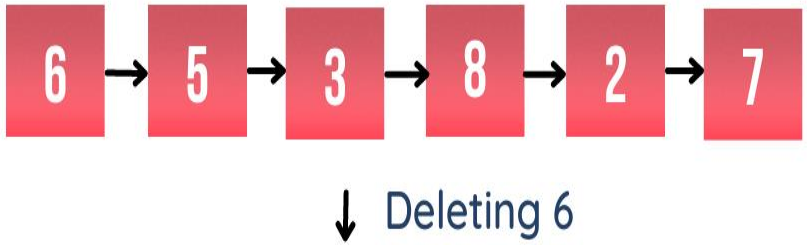
Sorted Set

Initially Sorted Set is Empty

Step 1

We start with the first item 6

Current List

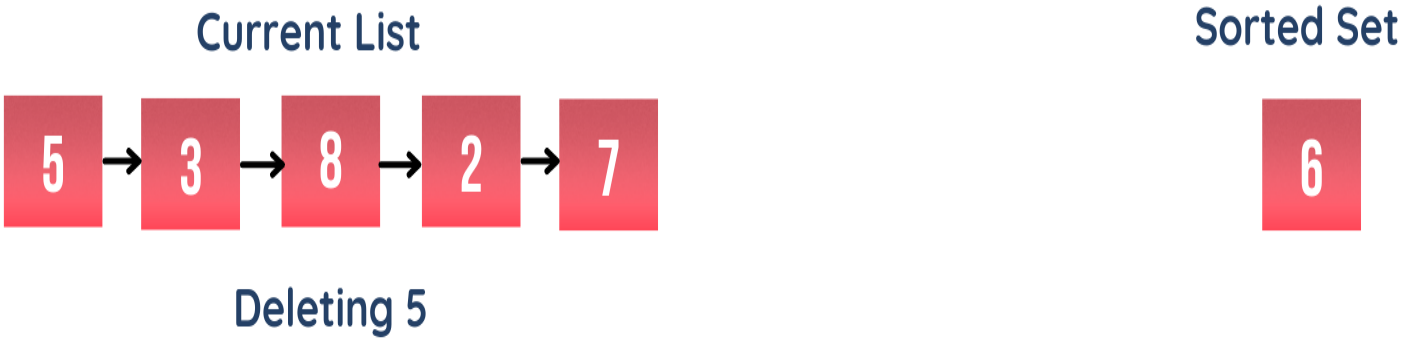


Sorted Set

Initially Sorted Set is Empty

Step 2

Sorted set now has 6 and we remove the next item 5 from the current list



Step 2

Sorted set now has 6 and we remove the next item 5 from the current list

Step 3

As 5<6, 5 is inserted before 6 in the sorted set

Step 4

As 3<5, so place it before 5 in the sorted set

Current List

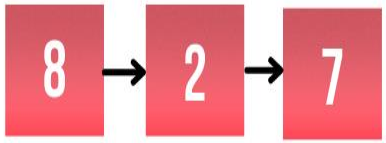
Deleting 8

Sorted Set

Step 4

As 3<5, so place it before 5 in the sorted set

Current List



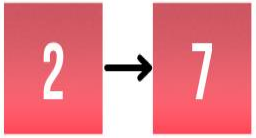
Deleting 8

Sorted Set

Step 5

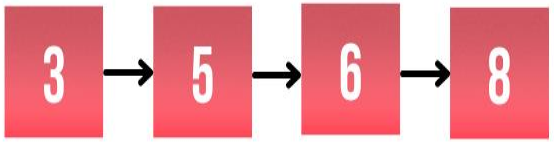
8>6, so it's placed after 6 in the sorted set

Current List



Deleting 2

Sorted Set



Step 6

As 2<3, so place it before 3

Current List

Deleting 7

Sorted Set

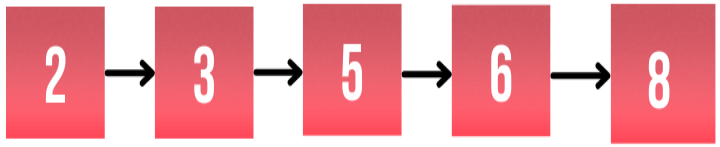
Step 6

As 2<3, so place it before 3

Current List

Deleting 7

Sorted Set



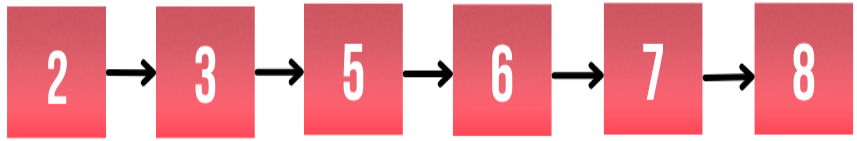
Step 7

As 7>6 and 7<8, so place it after 6. The current list is empty and our sorting is complete

Current List

Empty

Sorted Set



Pseudocode for Insertion Sort on a Singly Linked List

```
function InsertionSort(Node head)
    Node sorted = NULL
    Node curr = head
    while curr != NULL
        Node currNext = curr.next
        SortedInsert(sorted ,curr)
        curr = currNext
    end while
    head = sorted
end function
```

```
function SortedInsert(Node sorted, Node new_node)
    If sorted == NULL or sorted.data > new_node.data
        new_node.next = sorted
        sorted = new_node
    end if
    else
        Node curr = sorted
        while curr.next != NULL and curr.next.data <=
new_node.data
            curr = curr.next
        end while
        new_node.next = curr.next
        curr.next = new_node
    end else
end function
```

Insertion Sort in Java

```
class InsertionSort {

    void sort(int arr[])
    {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }

    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");

        System.out.println();
    }


    public static void main(String args[])
    {
        int arr[] = {27, 9, 6, 31, 0, -3, 4, 14};

        InsertionSort ob = new
InsertionSort();
        ob.sort(arr);

        printArray(arr);
    }
}
```

Insertion Sort - Worst Case Analysis

- The array is in reverse sorted order "**while** i > 0 and A[i] > key"
  - Always $A[i] > key$ in **while** loop test
  - Have to compare key with all elements to the left of the j-th position $\Rightarrow$ compare with $j-1$ elements $\Rightarrow t_j = j$

- $T(n) = O(n^2)$ order of growth in n^2

Insertion Sort vs. Selection Sort

They are different algorithms

They solve the same problem

Have the same worst-case asymptotic complexity

- Insertion-sort has better best-case complexity (when input is “mostly sorted”)

Other algorithms are more efficient for larger arrays that are not already almost sorted

- Insertion sort works well with small arrays

Bubble Sort

The term Bubble Sort was coined in 1962, by Iverson

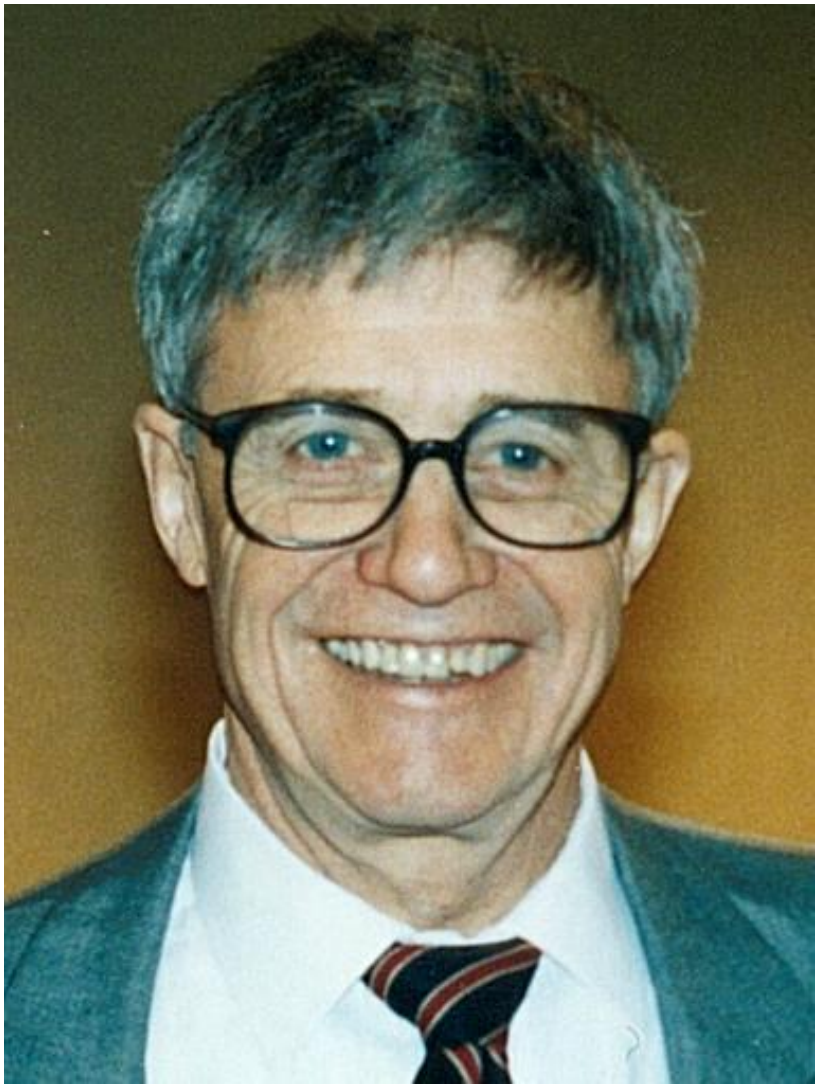
Bubble Sort is not a good algorithm

- Poor asymptotic complexity: $O(n^2)$ average
- Not efficient with respect to constant factors
- If it is good at something, some other algorithm does the same or better

However, Bubble Sort is often taught about

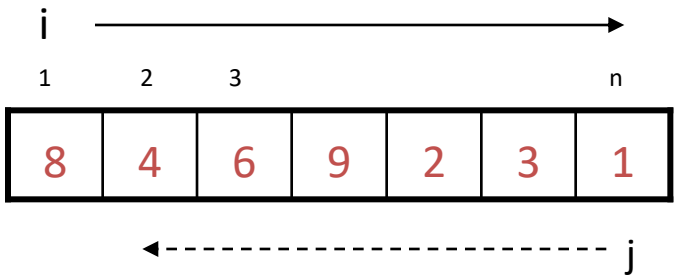
- Some people teach it just because it was taught to them
- Fun article to read:
  Bubble Sort: An Archaeological Algorithmic Analysis,
  Owen Astrachan, SIGCSE 2003

Kenneth E. Iverson (Turning Award Laureate)



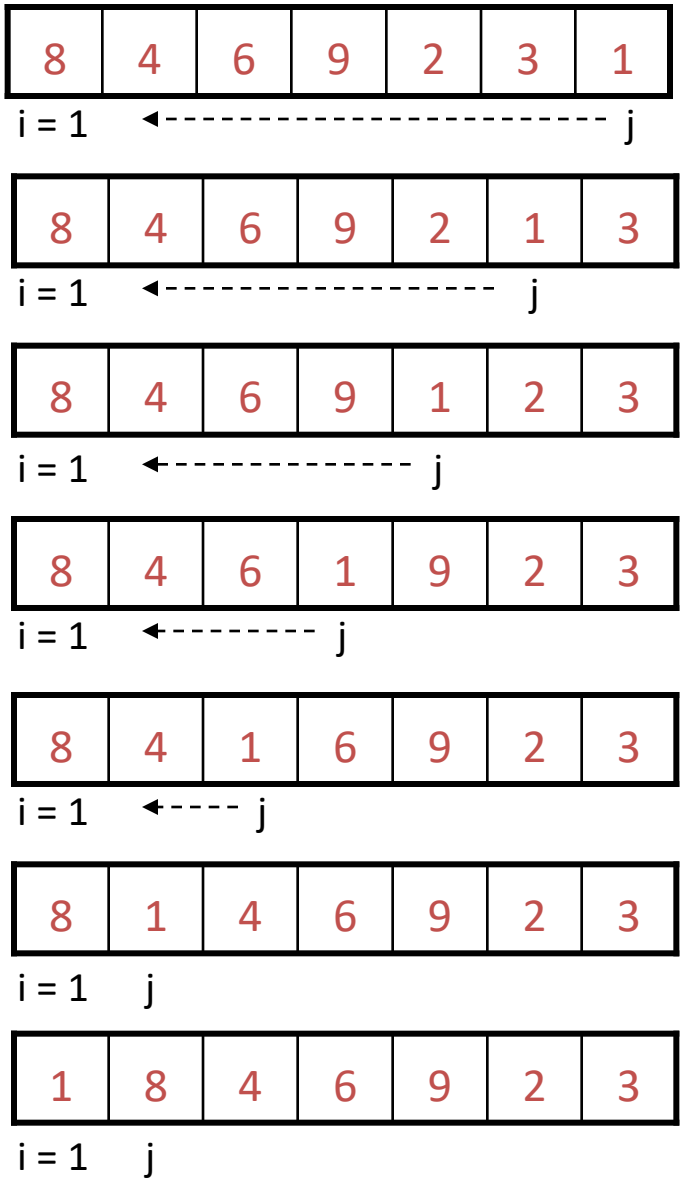
Bubble Sort contd'

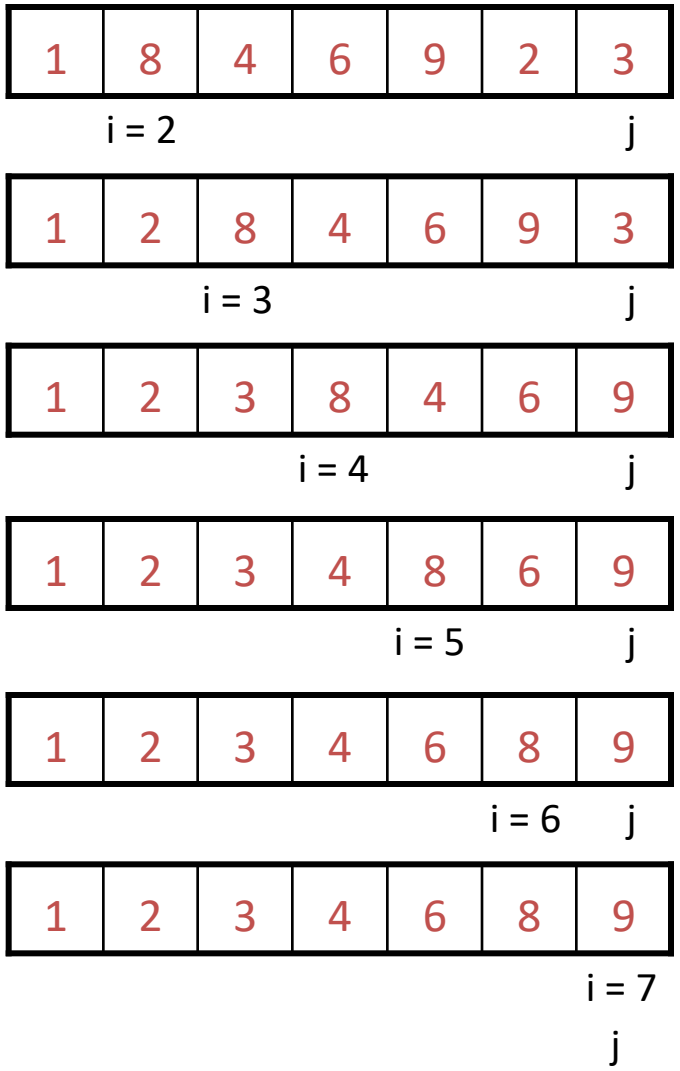
- Idea:
  - Repeatedly pass through the array
  - Swaps adjacent elements that are out of order



- Easier to implement, but slower than Insertion sort

Example – Bubble Sort





Bubble Sort in Java

```
import java.util.Arrays;
class Main {

  static void bubbleSort(int array[]) {
    int size = array.length;

    for (int i = 0; i < (size-1); i++) {

      boolean swapped = false;
      for (int j = 0; j < (size-i-1); j++) {

        if (array[j] > array[j + 1]) {
          int temp = array[j];
          array[j] = array[j + 1];
          array[j + 1] = temp;

          swapped = true;
      }
    }
if (!swapped)
      break;

    }
  }

  public static void main(String args[]) {

    int[] data = {27, 9, 6, 31, 0, -3, 4, 14};


    Main.bubbleSort(data);

    System.out.println("Sorted Array in
Ascending Order:");

System.out.println(Arrays.toString(data));
  }

}
```

Analysis of Bubble Sort

- Comparisons: $\approx n^2/2$
- Exchanges: $\approx n^2/2$
- $T(n) = O(n^2)$

Our Sorting Tool

https://algorithmvisualizer.github.io/AlgoVis/

Best way to sort?

It depends!

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press.
- Aho Alfred, V., et al. *Data structures and algorithms*. USA: Addison-Wesley, 1983.
- Weiss, M. A. (2012). *Data structures and algorithm analysis in Java*. Pearson Education, Inc.
- https://www.cse.unr.edu/~bebis/CS477/Lect/InsertionSortBubbleSortSelectionSort.ppt

Thank you