CS 2100: Data Structures & Algorithms 1

Basic Sorts (Part II)

Insertion Sort; Lower-Bound Discussion

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- The University updated the mask policy. As per my Request on Mar 28, 2022 (see Collab), I would greatly appreciate if you would do me a kind favor by **continuing to wear your masks** in CS 2100 (Ridley G008). I know it is a lot to ask, and it is **voluntary**, but I appreciate your understanding.
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

{Reminder} How to Sort?

- Some "straightforward" sorting algorithms
  - Insertion Sort, Selection Sort, Bubble Sort
  - **Each is O(n^2)**

- More efficient sorting algorithms
  - Quicksort, Mergesort, Heapsort
  - **Each is O(n log n)**

Best Sorts are O(n log n)

Sorting using Collections.sort()

- Collections and Arrays classes provide `.sort()` methods!
  - Utilizes `compareTo()` or `Comparator` to ***determine order*** when comparing elements
  - "under the hood", it's a *variant* of something called ***Mergesort***
  - Θ(n log n) worst-case -- as good as we can do
  - We'll discuss how **Mergesort** works soon!

Insertion Sort

Another example of a sorting algorithm

Insertion Sort

- Similar to **bubble sort**, except some slight improvements.
- Most notably, insertion sort will terminate the inner loop *when there is no need to continue* (i.e., this element already in correct position.)

Insertion Sort: Overall Idea & Pseudocode [not complete]

- **Idea**: At any point during sorting, elements 0 through i-1 are sorted and element i onward are not.
- Take element **i**, and slide it down the list until in position, then stop and move onto i+1
- Once i finds its correct spot, *no need to continue* moving down the list.

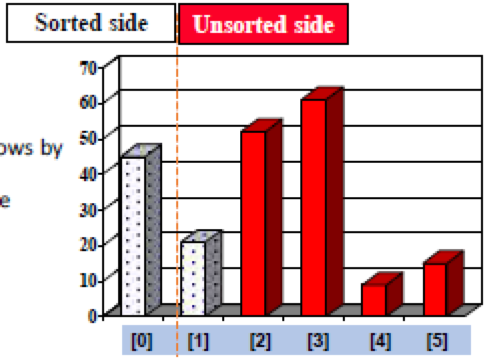
```
insertionSort(List list):
  for each i from 1 to n-1
    val = list[i]
    for each j from i-1 to 0
      if val < list[j]
        list[j+1] = list[j] //slide j up one
    list[j+1] = val //insert val in correct spot
```

However,
if val >= list[j]
you must <u>break</u> the
(inner) loop,
execute last line
(found place for val)
and go back to
outer loop

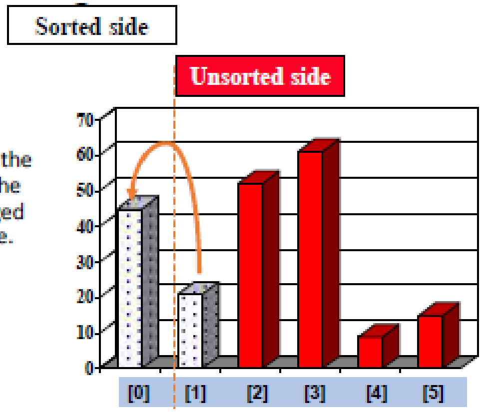
Insertion Sort

- The basic approach to **Insertion Sort** is to make **multiple passes** through the array.
- In each pass, we "**insert**" the first element in the **unsorted** side into its **correct sorted position** in the **sorted** side.
- At the end of each pass, ***all the elements in the "sorted" side are sorted*** in relation to one another, but *may not be in their final sorted position.*
- We keep making passes through the array **until all the elements are in order.**

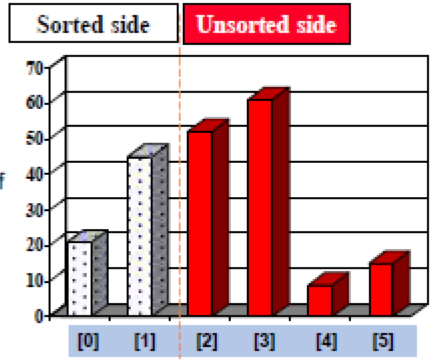
The sorted side grows by taking the front element from the unsorted side...



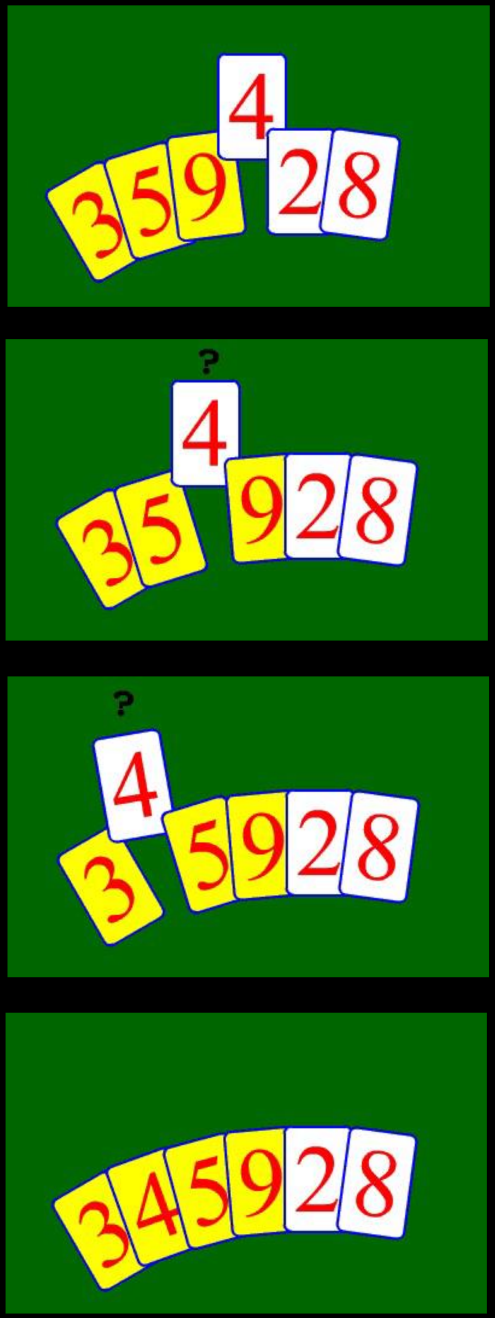
...and inserting it in the place that keeps the sorted side arranged from small to large.



In this example, the new element goes in front of the element that was already in the sorted side.



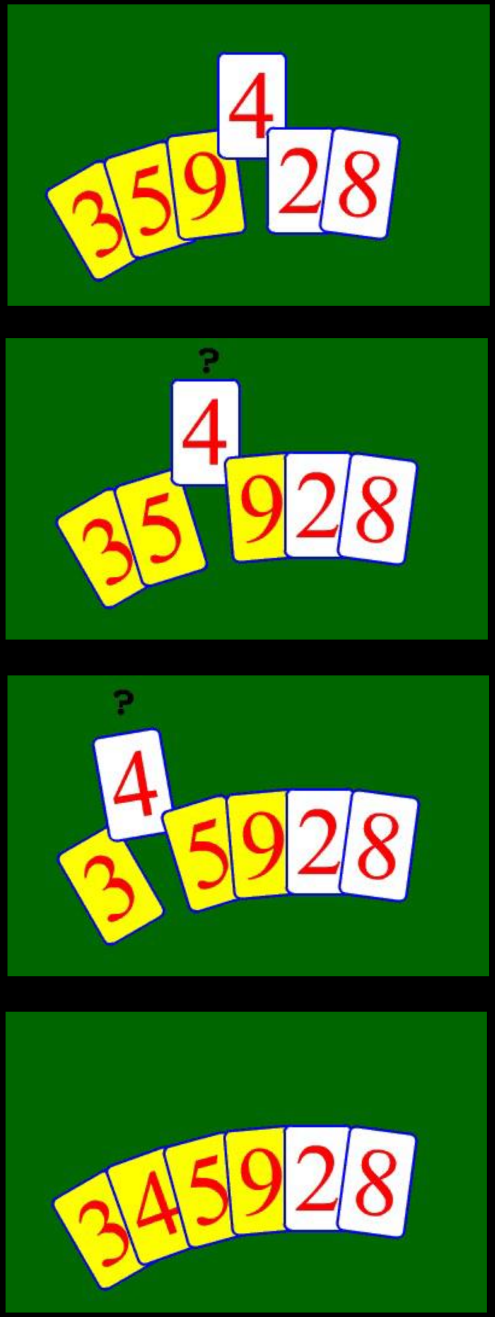
Insertion Sort



Think: sorting a deck of cards!

- Yellow: sorted
- White: unsorted
- Insert 4 in sorted position:
  - 4 is smaller than 9; shift 9 to right
  - 4 is smaller than 5, shift 5 to right
  - 4 is larger than 3, INSERT 4 into correct slot

Insertion Sort



Insertion Sort: Analysis

- Worst-Case is the same as bubble sort: **$\Theta(n^2)$**
- BUT, if the inner loop only has to shift a few elements out of the way each time, then it terminates early, which makes insertion sort **very fast** in some situations.
  - List is "***almost sorted***"
  - List is ***very small***

Cool Sorting Algorithms Visualizations

https://www.toptal.com/developers/sorting-algorithms

Best of a Breed?

Lower Bounds Proof

- We attempt to: Prove a whole <u>class of algorithms</u> have the same best-case run-time

- **Question**: **bubble** and **insertion** sort are **adjacent sorts**. Is it possible to develop an adjacent sort algorithm that is (Little-Oh) **$o(n^2)$** [*strictly faster than **n^2***]?
- **Claim**: Sorting a list *by only swapping adjacent elements* is (**Big-Omega;** LOWER BOUND) $\Omega(n^2)$
- Proof: Coming up!

... Spoiler

- Answer: **NO!**

  - It is not possible for another adjacent sort algorithm to run more efficiently than $\mathbf{n^2}$ time
  - That is, quadratic ($\mathbf{n^2}$) runtime is the **best** runtime you can possibly achieve with any adjacent sorting algorithm → It could be $\mathbf{n^2}$, or worse *(your approach might be really inefficient, doing worse than* $\mathbf{n^2}$*)*
    - Therefore: it is Big-Omega($\mathbf{n^2}$), or $\Omega(n^2)$

[Aside] Printing n elements in a list / Counting Argument

- At **minimum**, you look at all n items at least once, and print each one
  - print(arr[0])
  - print(arr[1])
  - print(arr[2])
  - …
  - print(arr[n])
- Therefore: (**Big-Omega;** LOWER BOUND) $\Omega(n)$
- There exists **<u>no</u>** clever way to **print n items** in less than BigTheta(n), $\Theta(n)$, time (doing n operations, or *more* if very *inefficient* solution)

Lower Bounds Proof

- **Overall Approach:** Count the *minimum amount of work* necessary to sort the list and divide by how much of the list can be fixed in a single operation.
- **Inversion:** An inversion is a pair of elements in the list x and y that are *not in relatively correct sorted order.*
  - **Inversion count:** count of amount of work left to do to sort the list
  - *What we want…*
  - A sorted list, and have a **count** to show there is no work left to do
  - An unsorted list, and have a **count** representing how much work is left to do
- **Observation:** A sorted list has 0 inversions. Thus "sorting" a list means *removing all inversions.*

Inversions

- **How many inversions in worst case as a function of n?**

- Every element in the list is out of order with every other element (NOT including itself)
  - n(n-1) inversions

- Final formula for Inversions: *Big-Theta*(n^2)

$$\frac{n(n-1)}{2} = \frac{n^2 - n}{2} = \theta(n^2)$$

- *We divide by 2, so that we don't count pairs twice!*
  - *"7" and "4" are out of relative order AND "4" and "7" are out of relative order (same thing!)*

How many Inversions Can I Fix Per Swap??

- How many inversion fixes can we do per operations?
- **n^2** inversions to fix (and we want to reduce this number of inversions to zero…)

- Q: How many inversions can I lower per operations?
- A: swap → fixes only one (1) inversion!

Lower Bound Proof

- **Min Inversions:** 0 (a sorted list)
- **Max Inversions:** ((n)(n-1)) / 2 = $\Theta(n^2)$ (i.e., a reverse sorted list)
- Fixes per swap: Swapping adjacent items can fix at most one (1) inversion.
- **Conclusion:** Worse case is always (**Big-Omega;** LOWER BOUND) $\Omega(n^2)$

- *Only way to achieve less than quadratic run-time, is if you can fix > 1 inversion per swap*
  - *Better algorithms fix > 1 thing per operation!*