Searching and Sorting

- Linear and Binary Search
- Order Statistics - MIN and MAX
- Comparison Based Sorting Algorithms
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
- Lower Bound on Comparison based sorting
- Non-Comparison Based Sorting - Integers Sorting
  - Counting Sort
  - Radix Sort

IMDAD ULLAH KHAN

Sorting

Sorting is to order of numbers in an array. The desired order can be

- Ascending or increasing
- Descending or decreasing

Generally, we sort in ascending order

- Arrangement from smallest value to largest value

Array A is sorted if $A[1] \leq A[2] \leq \cdots A[i] \leq A[i+1] \leq \cdots \leq A[n]$

$A =$

min	2^{nd} min	$\cdots\cdots$	i^{th} min	$\cdots\cdots$	2^{nd} max	max

Selection Sort

Selection sort repeatedly finds the minimum of the *'remaining array'* and brings to its correct position

In i^{th} pass, minimum value in $A[i, \ldots, n]$ is moved to index i

	1	2	3	4	5	6	
$A =$	8	9	3	4	2	7	
Pass 1	2	9	3	4	8	7	$\textsc{FindMin}(A[1-6])$ $\textsc{Swap}(min(A[1-6]), A[1])$
Pass 2	2	3	9	4	8	7	$\textsc{FindMin}(A[2-6])$ $\textsc{Swap}(min(A[2-6]), A[2])$
Pass 3	2	3	4	9	8	7	$\textsc{FindMin}(A[3-6])$ $\textsc{Swap}(min(A[3-6]), A[3])$
Pass 4	2	3	4	7	8	9	$\textsc{FindMin}(A[4-6])$ $\textsc{Swap}(min(A[4-6]), A[4])$
Pass 5	2	3	4	7	8	9	$\textsc{FindMin}(A[5-6])$ $A[5] < A[6]$, $\textsc{No Swap}$

Selection Sort

Selection sort repeatedly finds the minimum of the *'remaining array'* and brings to its correct position

In i^{th} pass, minimum value in $A[i, \ldots, n]$ is moved to index i

---

Algorithm $\textsc{Selection-Sort}(A)$

---

for $i = 1$ to $n - 1$ **do**

$\quad (min, indexofMin) \leftarrow \textsc{findmin}(A[i, \ldots, n])$

$\quad \textsc{swap}(A[i], A[indexofMin])$

Correct by definition!

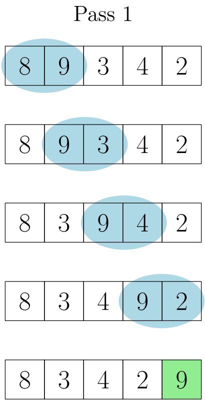
Number of comparisons in successive calls to $\textsc{findmin}$:

$$(n-1) + (n-2) + \cdots + 3 + 2 + 1 \;=\; \frac{n(n-1)}{2} \;=\; O(n^2)$$

Bubble Sort

Bubble sort repeatedly moves the largest element to the end of the *'remaining array'*

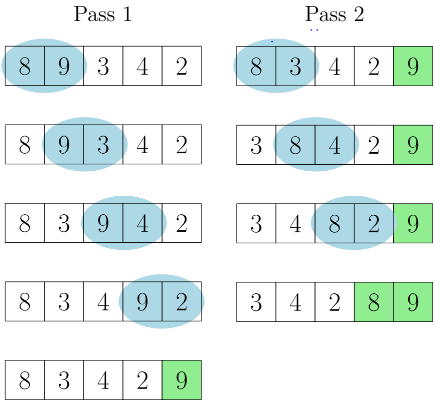
Swaps out-of-order adjacent elements (in a moving bubble)



Bubble Sort

Bubble sort repeatedly moves the largest element to the end of the *'remaining array'*

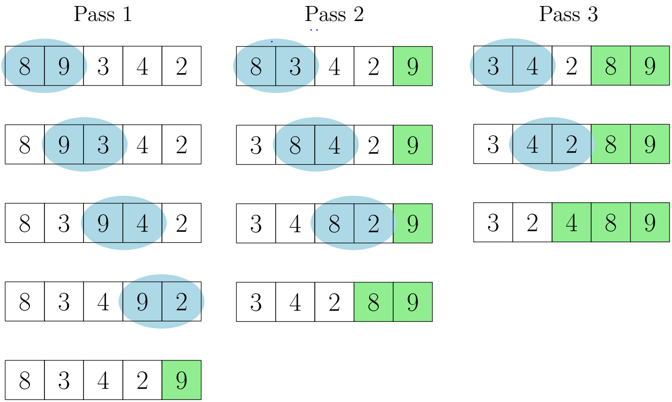
Swaps out-of-order adjacent elements (in a moving bubble)



Bubble Sort

Bubble sort repeatedly moves the largest element to the end of the *'remaining array'*

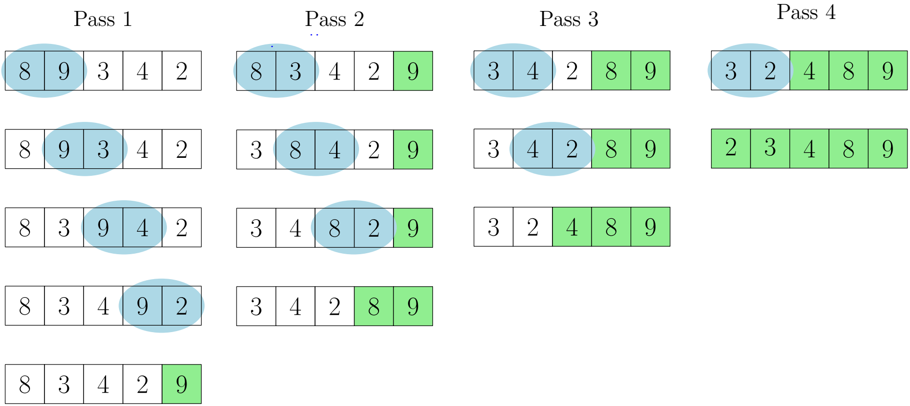
Swaps out-of-order adjacent elements (in a moving bubble)



Bubble Sort

Bubble sort repeatedly moves the largest element to the end of the *'remaining array'*

Swaps out-of-order adjacent elements (in a moving bubble)



Bubble Sort

Bubble sort repeatedly moves the largest element to the end of the *'remaining array'*

Swaps out-of-order adjacent elements (in a moving bubble)

Algorithm $\textsc{Bubble-Sort}(A)$

for $pass = 1$ to $n - 1$ **do**
  for $j = 1$ to $n - pass$ **do**
    if $(A[j] > A[j+1])$ **then**
      $\textsc{Swap}(A[j], A[j+1])$

Worst case number of comparisons is

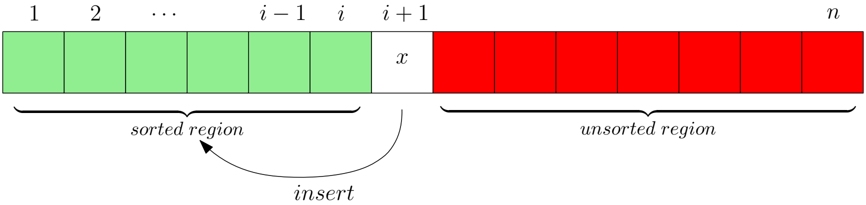
$$(n-1) + (n-2) + \cdots + 3 + 2 + 1 \; = \; O(n^2)$$

Early detect if the array gets sorted (if no swap in a pass)

Insertion Sort

Insertion Sort maintains the *'initial sorted region'* $A[1, \ldots, i]$

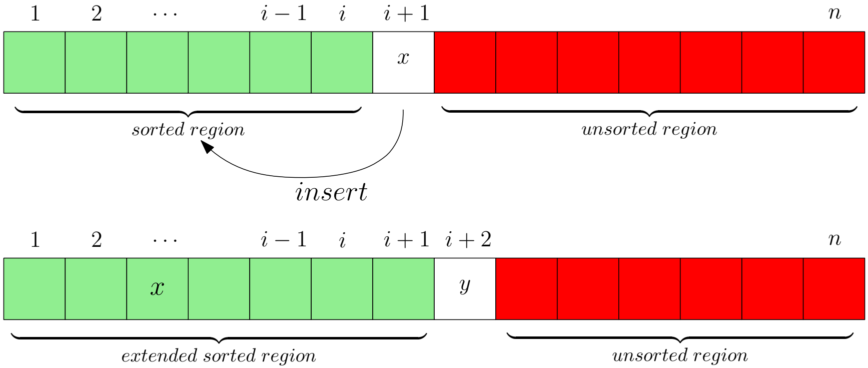
Inserts $A[i+1]$ into the sorted region to extend it



Insertion Sort

Insertion Sort maintains the *'initial sorted region'* $A[1, \ldots, i]$

Inserts $A[i+1]$ into the sorted region to extend it



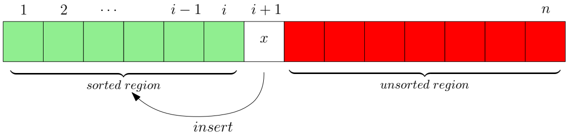
Insertion Sort

Insertion Sort maintains the *'initial sorted region'* $A[1,\ldots,i]$

Inserts $A[i+1]$ into the sorted region to extend it

Algorithm INSERTION-SORT(A)

for $i = 1$ to $n-1$ **do**
  $x \leftarrow A[i+1]$
  $j \leftarrow i$
  while $j > 0$ AND $A[j] > x$ **do**
    $A[j+1] \leftarrow A[j]$
    $j = j - 1$
  $A[j] \leftarrow x$



Insertion Sort: Example

$A =$	8	9	3	4	2	7

Pass 1	8	9	3	4	2	7
Pass 2	8	9	3	4	2	7
Pass 3	3	8	9	4	2	7
Pass 4	3	4	8	9	2	7
Pass 5	2	3	4	8	9	7
Pass 6	2	3	4	7	8	9

9 inserted with 0 swaps

3 inserted with 2 swaps

4 inserted with 2 swaps

2 inserted with 4 swaps

7 inserted with 2 swaps

Insertion Sort: Analysis

Best-Case: When A is already sorted

- No swapping to insert elements at correct position
- 1 comparison at each pass, $n-1$ total comparisons
- No swaps

Worst-Case: When A is reverse sorted

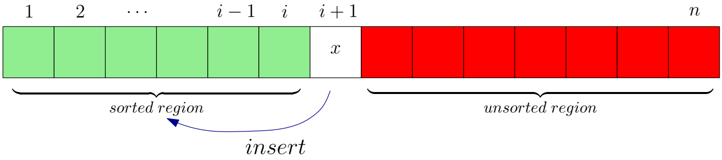
- In each pass all elements in sorted region are compared and swapped
- Number of comparisons: i comparisons in pass i

$$1+2+3+\cdots+(n-2)+(n-1) \;=\; O(n^2)$$

- Number of swaps: i swaps in pass i

$$1+2+3+\cdots+(n-2)+(n-1) \;=\; O(n^2)$$

Insertion Sort with BINARY-SEARCH



- Use EXTENDED BINARY-SEARCH to find position of x in sorted region
- $\log i$ comparisons in the i^{th} pass
- Total comparisons:

$$\log 1 + \log 2 + \log 3 + \cdots + \log n \; = \; \log n! \; \approx \; n \log n$$

▷ Follows from $\log a + \log b = \log(ab)$ and Stirling's approximation

Which sorting algorithm is better?

Selection, insertion, and bubble sort all have worst case runtimes $O(n^2)$

When A is already sorted

- insertion sort benefits
- Bubble sort with early stopping too

InsertionSort with binary search takes $O(n \log n)$ comparisons

If number of comparisons is our only concern (swaps don't count), then this is the best we can do

▷ See lower bound on comparison based sorting