Université d'Ottawa | University of Ottawa

CSI2110 Data Structures and Algorithms

- Session 13: Quadratic Sort

Professor: Karim Al Ghoul – Summer 2023

uOttawa

Quadratic Sorting

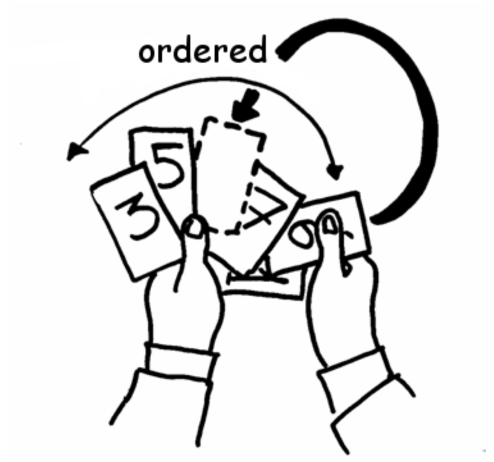
Review

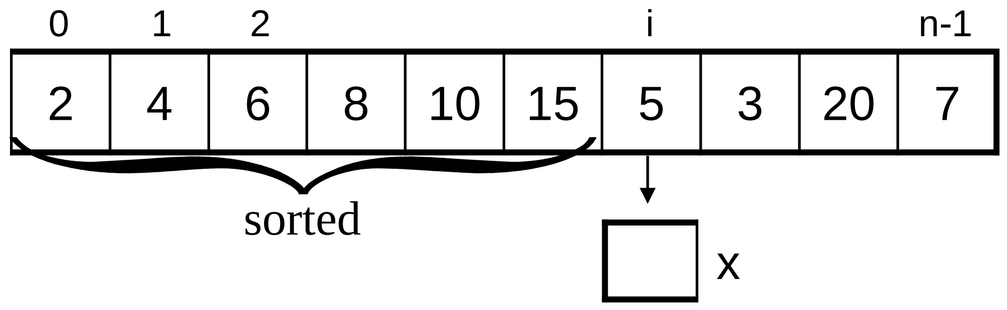
- Insertion Sort with an array

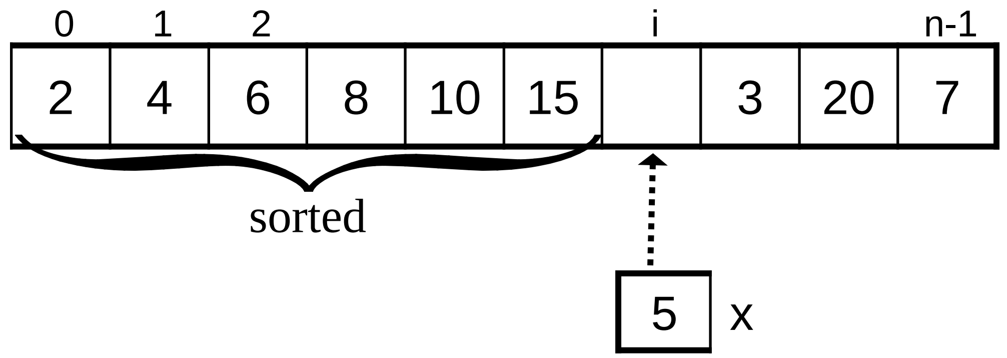
Review

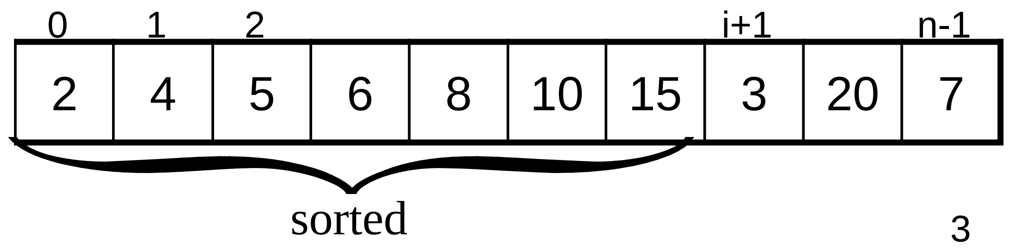
- Selection Sort with an array
- Bubblesort with an array

Insertion Sort (array)

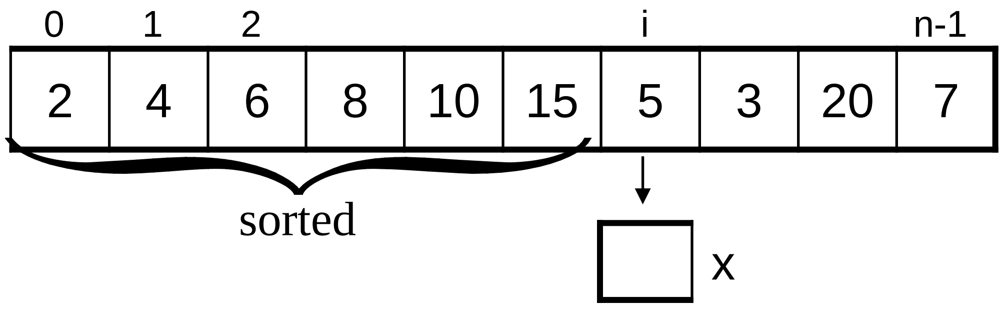








```
for i=1 to n-1
   x ← A[i]
   j ← i-1
   while x.key < A[j].key
            and j >= 0
        A[j+1] ← A[j]
        j ← j-1
   A[j+1] ← x
```



Complexity Insertion sort for arrays

Number of comparisons:

MIN $(C_i)_{min} = 1$ $i=1..n-1$ (already in order) $\longrightarrow$ $C_{min} = n-1 = O(n)$

MAX $(C_i)_{max} = i$ $i=1..n-1$ (in reverse order) $\longrightarrow$ $$C_{max} = \sum_{i=1}^{n-1} i$$

$= n(n-1)/2$

$= O(n^2)$

And with Sequence implemented with linked list ?

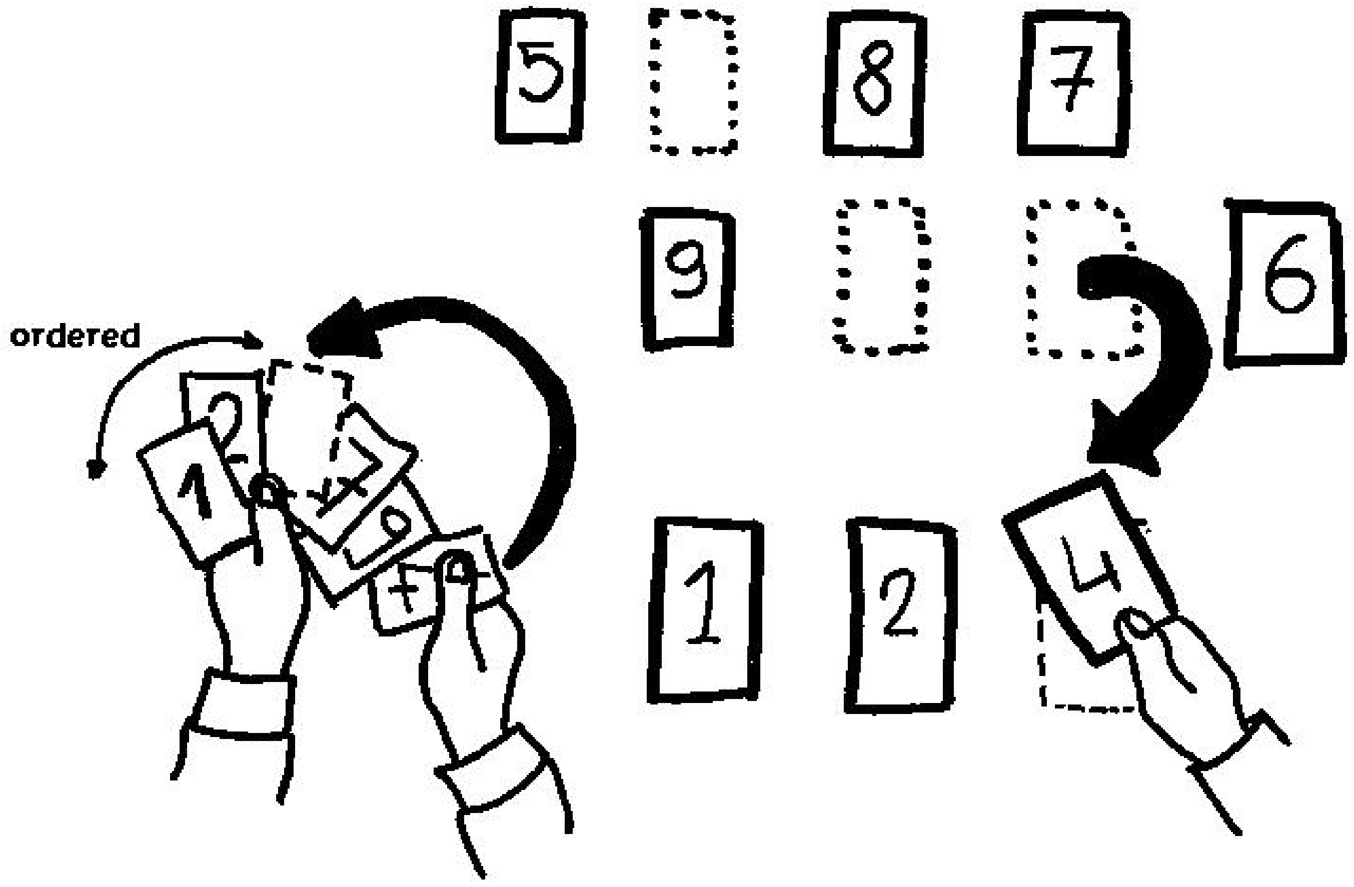
Complexity — Insertion sort for arrays

Number of movements:

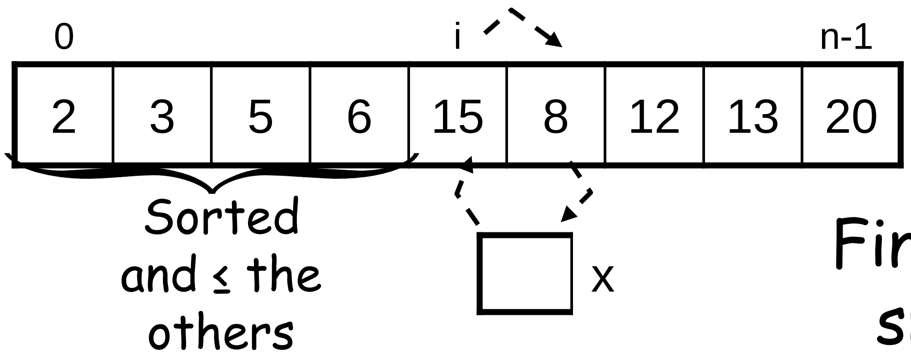
Worst case $O(n^2)$

And with Sequence implemented with linked list ?

Selection Sort (array)



Selection Sort (array)



Finding the smallest element

```
for i=0 to n-2
   k ← i
   x ← A[i]
   for j=i+1 to n-1
      if A[j].key < x.key
            k ← j
            x ← A[j]
   A[k] ← A[i]
   A[i] ← x
```

Complexity of Selection Sort (array)

Comparisons (Does not depend on the initial order of the elements)

$$c = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i) = \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$$

$$= n(n-1) - n(n-1) / 2$$

$$= n(n-1) / 2$$

$$= O(n^2)$$

Insertion vs Selection Sort

- **Insertion sort** divides the array into sorted and unsorted parts, placing each element in the correct position within the sorted part.
- **Selection sort** finds the minimum element in the unsorted part and moves it to its proper place in the sorted part.
- **Insertion sort** has a time complexity of O(n^2), but performs better on partially sorted arrays (best-case: O(n)).
- **Selection sort** requires fewer swaps but more comparisons than insertion sort.
- **Insertion sort** is more efficient when the array is partially sorted, while selection sort is better for highly unsorted arrays.
- Choose the algorithm based on the input data characteristics and specific requirements of the problem.

Bubble Sort

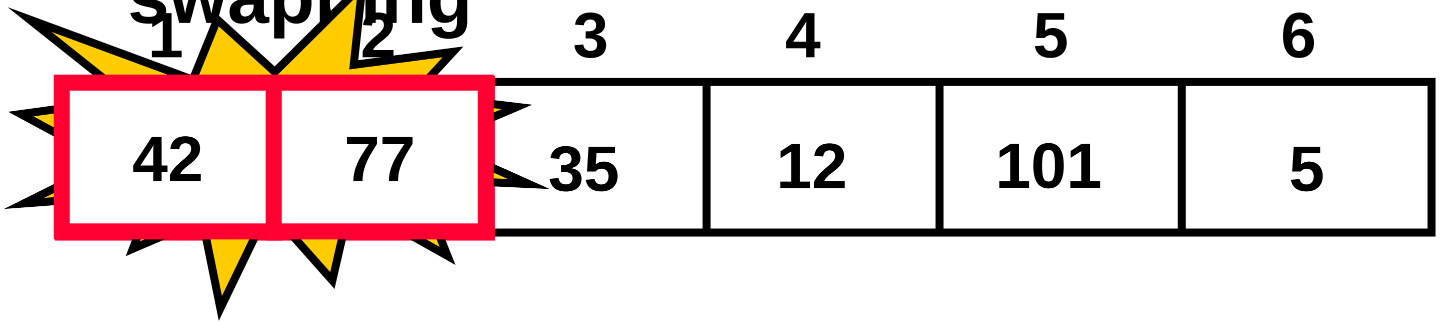
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

1	2	3	4	5	6
77	42	35	12	101	5

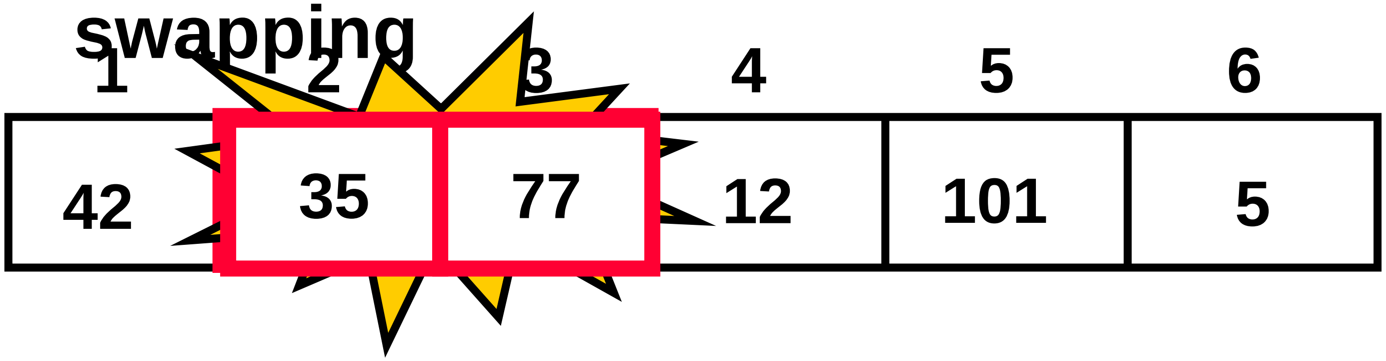
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**



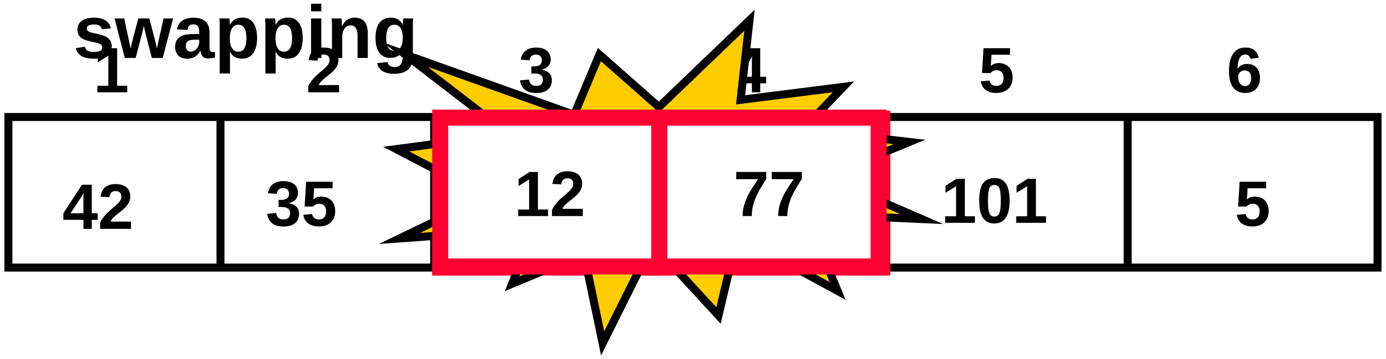
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**



"Bubbling Up" the Largest Element

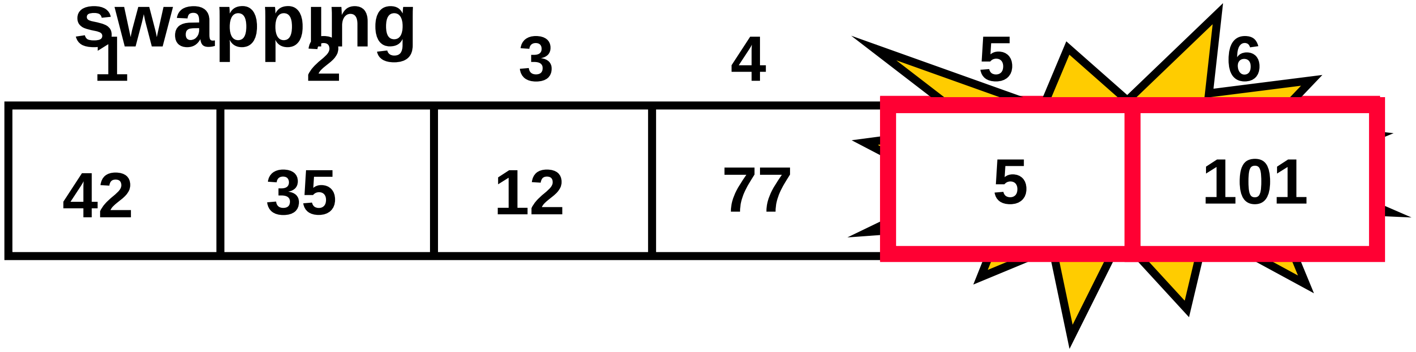
- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

1	2	3	4	5	6
42	35	12	77	101	5

No need to swap

"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**



"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

0	1	2	3	4	5
42	35	12	77	5	101

Largest value correctly placed

Items of Interest

- **Notice that only the largest value is correctly placed**
- **All other values are still out of order**
- **So we need to repeat this process**

0	1	2	3	4	5
42	35	12	77	5	101

Largest value correctly placed

Repeat “Bubble Up” How Many Times?

- **If we have N elements…**
- **And if each time we bubble an element, we place it in its correct location…**
- **Then we repeat the “bubble up” process N – 1 times.**
- **This guarantees we’ll correctly place all N elements.**

“Bubbling” All the Elements

N - 1

0	1	2	3	4	5
42	35	12	77	5	101

0	1	2	3	4	5
35	12	42	5	77	101

0	1	2	3	4	5
12	35	5	42	77	101

0	1	2	3	4	5
12	5	35	42	77	101

0	1	2	3	4	5
5	12	35	42	77	101

Complexity of Bubblesort (arrays)

And with Sequence implemented with linked list ?

Comparisons

$$c = \sum_{i=1}^{n-1} (n-i) = n - \left\{ \frac{n}{2}(n-1) \right\} = O\left(n^2\right)$$

Movements

$D_{min} = 0$ (already in order)

$D_{max} = O(n^2)$ (in reverse order)

Already Sorted Collections?

- **What if the collection was already sorted?**
- **What if only a few elements were out of place and after a couple of “bubble ups,” the collection was sorted?**
- **We want to be able to detect this and “stop early”!**

0	1	2	3	4	5
5	12	35	42	77	101

Using a Boolean “Flag”

- **We can use a boolean variable to determine if any swapping occurred during the “bubble up.”**
- **If no swapping occurred, then we know that the collection is already sorted!**
- **This boolean “flag” needs to be reset after each “bubble up.”**