COMP20230: Data Structures & Algorithms
Lecture 13: Sorting (1)

Dr Andrew Hines

Office: E3.13 Science East
School of Computer Science
University College Dublin

andrew.hines@ucd.ie

Sorting (1)

Outline

Problem and Applications
Sorting Algorithms: **Bubble, Selection, Insertion**, Quick, Merge, Heap

Take Home Message

Sorting speed depends on the algorithm and the initial data state.

Sorting Problem

Generic Sorting Algorithm

Input: Sequence n of elements in no particular order
Output: Sequence rearranged in increasing order of elements?
values

- Motivation: Fundamental to many real-world applications (e.g. online shopping – sort by price/category/colour)
- Very popular exercise to learn the concepts behind algorithms and data structures (why?)

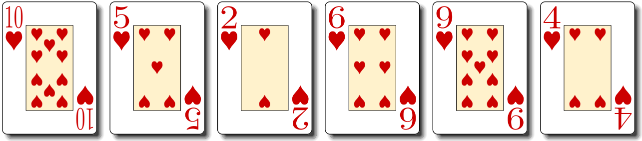
Sorting Algorithms (some)

Hundreds of types and variants, e.g.

bubble_sort
selection_sort
insertion_sort
quick_sort
merge_sort
heap_sort
shell_sort

Bubble Sort Algorithm

1. Get a hand of unsorted cards (same suit)



2. Repeat steps 3 through 5 until nothing happens:
3. For every couple/pair of neighbouring cards (left-right)
4. If the number on the left is bigger than the one on the right
5. Swap the cards
6. Stop

Bubble Sort

- Bubble sort sorts a sequence (ADT) of values
- Based on a structured pattern of **comparison-exchange (CE)** operations
- `comparison_exchange(i)`: Take value in two adjacent slots in the sequence and if the values are out of order (i.e. the larger before the smaller), then swap them around

Pass and Sweep

Sweep

Bubble sort carries out $n-1$ **passes** through the list. For each pass, it carries out a **sweep** of $n-1$ comparison exchanges, left to right:

$$\boxed{10, 5}, 2, 6, 9, 4$$

$$5, \boxed{10, 2}, 6, 9, 4$$

$$5, 2, \boxed{10, 6}, 9, 4$$

$$5, 2, 6, \boxed{10, 9}, 4$$

$$5, 2, 6, 9, \boxed{10, 4}$$

Pass

Passes (for 6 elements, $n-1=5$ passes):

$$10, 5, 2, 6, 9, 4$$

$$5, 2, 6, 9, 4, 10$$

$$2, 5, 6, 4, 9, 10$$

$$2, 5, 4, 6, 9, 10$$

$$2, 4, 5, 6, 9, 10$$

Psuedocode: Bubble Sort

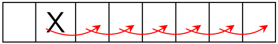
```
Algorithm bubble_sort
Input:  A an array of n elements
Output:  A is sorted
for s = 0 to n-1 do
    for current = 0 to n-2 do
        if A[current] > A[current + 1] then
          swap A[current] and A[current + 1]
        endif
    endfor
endfor
```

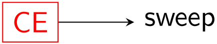
Psuedo-code: Bubble Sort

```
Algorithm bubble_sort
Input:  A an array of n elements
Output:  A is sorted
for s = 0 to n-1 do # Passes
    for current = 0 to n-2 do # Sweeps
        if A[current] > A[current + 1] then
          # Comparison Exchange (CE)
          swap A[current] and A[current + 1]
        endif
    endfor
endfor
```

Observation

- Consider largest value X:
    - No CE can move X leftwards
    - Every CE with X on LHS moves it rightwards
- First sweep pushes X into very last slot in the list (where it belongs)





- CEs of subsequent sweeps leave it there

Bubble Sort: Complexity

```
Algorithm bubble_sort
Input:  A an array of n elements
Output:  A is sorted
for s = 0 to n-1 do
    for current = 0 to n-2 do
        if A[current] > A[current + 1] then
          swap A[current] and A[current + 1]
        endif
    endfor
endfor
```

Complexity: $\mathcal{O}(n^2)$

Pass and Sweep

Sweep

Bubble sort carries out $n-1$ passes through the list. For each pass, it carries out a sweep of $n-1$ comparison exchanges, left to right:

$$\boxed{10, 5}, 2, 6, 9, 4$$

$$5, \boxed{10, 2}, 6, 9, 4$$

$$5, 2, \boxed{10, 6}, 9, 4$$

$$5, 2, 6, \boxed{10, 9}, 4$$

$$5, 2, 6, 9, \boxed{10, 4}$$

Pass

Passes (for 6 elements, $n-1=5$ passes):

$$10, 5, 2, 6, 9, 4$$

$$5, 2, 6, 9, 4, 10$$

$$2, 5, 6, 4, 9, 10$$

$$2, 5, 4, 6, 9, 10$$

$$2, 4, 5, 6, 9, 10$$

Optimising Bubble Sort (1)

Optimisation

When the array is sorted, we can stop. In the example below, we are sorted after pass 4 has completed.

List: 27, 13, 44, 15, 12, 99, 63, 57

Pass 1: 13 27 15 12 44 63 57 **99**
Pass 2: 13 15 12 27 44 57 **63 99**
Pass 3: 13 12 15 27 44 **57 63 99**
Pass 4: 12 13 15 27 **44 57 63 99**
Pass 5: 12 13 15 **27 44 57 63 99**
Pass 6: 12 13 **15 27 44 57 63 99**

End: 12, 13, 15, 27, 44, 57, 63, 99

Optimising Bubble Sort (1)

Optimisation

When the array is sorted, we can stop. In the example below, we are sorted after pass 4 has completed.

List: 27, 13, 44, 15, 12, 99, 63, 57

Pass 1: 13 27 15 12 44 63 57 **99**
Pass 2: 13 15 12 27 44 57 **63 99**
Pass 3: 13 12 15 27 44 **57 63 99**
Pass 4: 12 13 15 27 **44 57 63 99**

~~Pass 5: 12 13 15 **27 44 57 63 99**~~
~~Pass 6: 12 13 **15 27 44 57 63 99**~~

End: 12, 13, 15, 27, 44, 57, 63, 99

Optimising Bubble Sort (1)

```
Algorithm bubble_sort
Input:  A an array of n elements
Output:  A is sorted
for s = 0 to n-1 do
    swapped ← False
    for current = 0 to n-2 do
        if A[current] > A[current + 1] then
          swap A[current] and A[current + 1]
          swapped ← True
        endif
    endfor
    if not swapped then
        finish
    endif
endfor
```

Optimising Bubble Sort (2)

Optimisation

After the ith pass the last $(i-1)$ items are sorted: no need to keep evaluating them each pass.

List: 27, 13, 44, 15, 12, 99, 63, 57

Pass 1: 13 27 15 12 44 63 57 **99**
Pass 2: 13 15 12 27 44 57 **63 99**
Pass 3: 13 12 15 27 44 **57 63 99**
Pass 4: 12 13 15 27 **44 57 63 99**
Pass 5: 12 13 15 **27 44 57 63 99**
Pass 6: 12 13 **15 27 44 57 63 99**

End: 12, 13, 15, 27, 44, 57, 63, 99

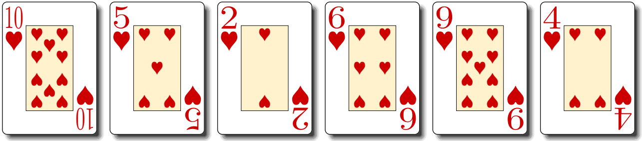
Optimising Bubble Sort (2)

```
Algorithm bubble_sort
Input:  A an array of n elements
Output:  A is sorted
for s = 0 to n-1 do
    swapped ← False
    for current = 0 to n - s - 2 do
        if A[current] > A[current + 1] then
          swap A[current] and A[current + 1]
          swapped ← True
        endif
    endfor
    if not swapped then
        finish
    endif
endfor
```

Selection Sort

Simple Sort (not quite)

1. Get a hand of unsorted cards (same suit)



2. Repeat steps 3 through 5 until nothing happens:
3. Compare all unsorted cards
4. Select the smallest unsorted card
5. Move this card to the sorted hand
6. Stop

Selection Sort

- Iteratively looks for the minimum value in an array
- Then swaps it with the leftmost unsorted item

List: 27, 13, 44, 15, 12, 99, 63, 57

27	13	44	15	12	99	63	57
12	13	44	15	27	99	63	57
12	**13**	44	15	27	99	63	57
12	**13**	**15**	44	27	99	63	57
12	**13**	**15**	**27**	44	99	63	57
12	**13**	**15**	**27**	**44**	99	63	57
12	**13**	**15**	**27**	**44**	**57**	63	99

End: **12, 13, 15, 27, 44, 57, 63, 99**

Selection Sort: Pseudo-code

```
Algorithm selection_sort
Input:  A an array of n elements
Output:  A is sorted
for j = 0 to n-2 do
    min ← j
    for i = j +1 to n -1 do
        if A[min] > A[i] then
            min ← i
        endif
    endfor
    swap a[min], a[j]
endfor
```

Selection Sort: Pseudo-code

```
Algorithm selection_sort
Input:  A an array of n elements
Output:  A is sorted
for j = 0 to n-2 do # For each element in the array
    min ← j # Find the min and swap
    for i = j +1 to n -1 do
        if A[min] > A[i] then
            min ← i
        endif
    endfor
    swap a[min], a[j]
endfor
```

Selection Sort: Complexity

```
Algorithm selection_sort
Input:  A an array of n elements
Output:  A is sorted
for j = 0 to n-2 do
    min ← j
    for i = j +1 to n -1 do
        if A[min] > A[i] then
            min ← i
        endif
    endfor
    swap a[min], a[j]
endfor
```

Complexity: $\mathcal{O}(n^2)$

Insertion Sort

- Shares with selection sort the idea of increasing the sorted section at the start of the array
- Takes the next item and puts it at the correct position

List: 27, 13, 44, 15, 12, 99, 63, 57

27	13	44	15	12	99	63	57
27	13	44	15	12	99	63	57
13	27	44	15	12	99	63	57
13	27	**44**	15	12	99	63	57
13	**15**	27	44	12	99	63	57
12	13	15	27	44	99	63	57
12	13	15	27	44	**99**	63	57
12	13	15	27	44	**63**	99	57
12	13	15	27	44	**57**	63	99

End: **12, 13, 15, 27, 44, 57, 63, 99**

Insertion Sort: Pseudo-code

```
Algorithm insertion_sort
Input:  A an array of n elements
Output:  A is sorted
for j = 1 to n-1 do
    i ← j
    while i > 0 and A[i-1] > A[i] do
        swap a[i] and a[i-1]
        i ← i - 1
    endwhile
endfor
```

Insertion Sort: Pseudo-code

```
Algorithm insertion_sort
Input:  A an array of n elements
Output:  A is sorted
for j = 1 to n-1 do # For each element in the array
    i ← j # Push right until element inserted
    while i > 0 and A[i-1] > A[i] do
        swap a[i] and a[i-1]
        i ← i - 1
    endwhile
endfor
```

Insertion Sort: Complexity

```
Algorithm insertion_sort
Input:  A an array of n elements
Output:  A is sorted
for j = 1 to n-1 do
    i ← j
    while i > 0 and A[i-1] > A[i] do
        swap a[i] and a[i-1]
        i ← i - 1
    endwhile
endfor
```

Complexity: $\mathcal{O}(n^2)$

Summary

Sorting algorithms are everywhere and there are many different implementations.

Bubble, selection and insertion sort are all the same complexity: $\mathcal{O}(n^2)$

Each could have the fastest runtime depending on the data.

Visual:
`http://algorithm-visualizer.org/brute-force/insertion-sort`

Why learn about sorting?

Get a job!

To look at problems algorithmically

To bring together our learning on complexity, recursion and data structures

Next up: Quick and Merge Sort

Python Implementation Example: Bubble Sort

```python
def bubblesort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
```

Python: Bubble Sort

Adding debug printouts to follow the code:

```
def bubblesort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            print(i, ':  comparing: ', alist[i],alist[i+1])
            if alist[i]>alist[i+1]:
                print('     ', alist[i], '>', alist[i+1], ' => switch')
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
                print("     New list order: ", alist)


def main():
    cards=[10, 5, 2, 6, 9, 4]
    print('before sorting:',cards)
    bubblesort(cards)
    print('after sorting:',cards)

if __name__ == '__main__':
    main()
```

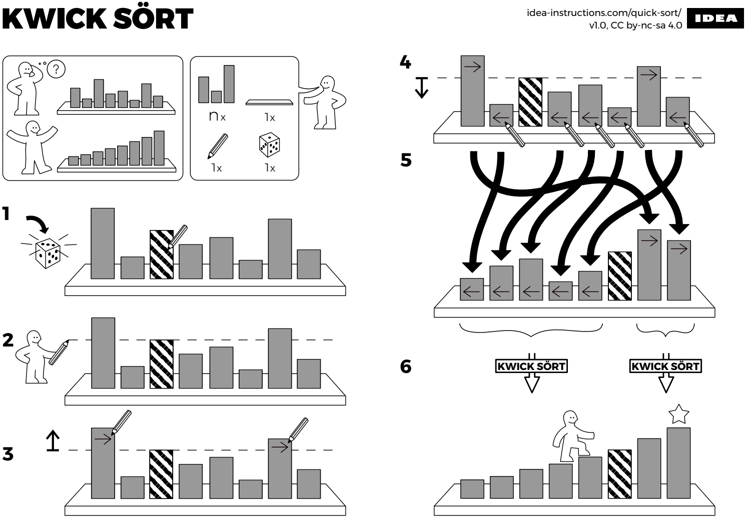
Bubble Sort: Output

Console output for bubble sort with debug to see the sweeps and passes (and while visualisation is easier than debug comments):

```
before sorting:  [10, 5, 2, 6, 9, 4]
0 :  comparing:  10 5
10 > 5 => switch
New list order:  [5, 10, 2, 6, 9, 4]
1 :  comparing:  10 2
10 > 2 => switch
New list order:  [5, 2, 10, 6, 9, 4]
2 :  comparing:  10 6
10 > 6 => switch
New list order:  [5, 2, 6, 10, 9, 4]
3 :  comparing:  10 9
10 > 9 => switch
New list order:  [5, 2, 6, 9, 10, 4]
4 :  comparing:  10 4
10 > 4 => switch
New list order:  [5, 2, 6, 9, 4, 10]
0 :  comparing:  5 2
5 > 2 => switch
New list order:  [2, 5, 6, 9, 4, 10]
1 :  comparing:  5 6
2 :  comparing:  6 9
3 :  comparing:  9 4
9 > 4 => switch
New list order:  [2, 5, 6, 4, 9, 10]
0 :  comparing:  2 5
1 :  comparing:  5 6
```

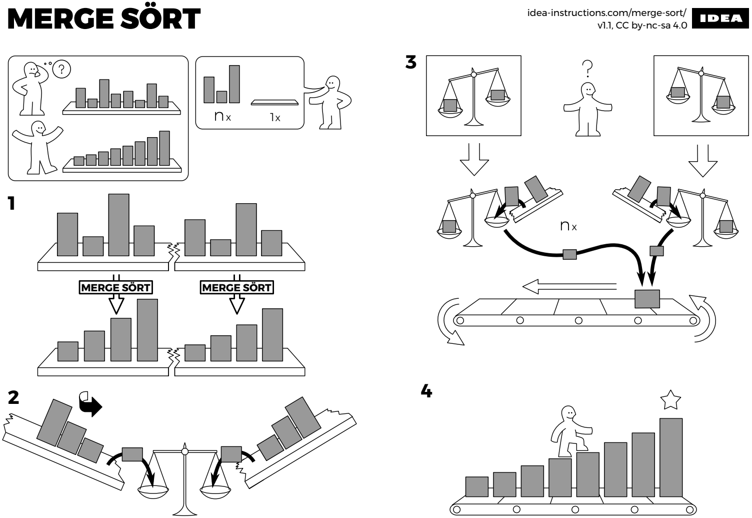
```
2 :  comparing:  6 4
6 > 4 => switch
New list order:  [2, 5, 4, 6, 9, 10]
0 :  comparing:  2 5
1 :  comparing:  5 4
5 > 4 => switch
New list order:  [2, 4, 5, 6, 9, 10]
0 :  comparing:  2 4
after sorting:  [2, 4, 5, 6, 9, 10]
3 :  comparing:  9 4
9 > 4 => switch
New list order:  [2, 5, 6, 4, 9, 10]
0 :  comparing:  2 5
1 :  comparing:  5 6
2 :  comparing:  6 4
6 > 4 => switch
New list order:  [2, 5, 4, 6, 9, 10]
0 :  comparing:  2 5
1 :  comparing:  5 4
5 > 4 => switch
New list order:  [2, 4, 5, 6, 9, 10]
0 :  comparing:  2 4
after sorting:  [2, 4, 5, 6, 9, 10]
```

Quick Sort IKEA Style



https://idea-instructions.com

Quick Sort IKEA Style



https://idea-instructions.com

This afternoon: Project Overview

We will be outlining the project and forming groups.
Groups of 4: You can choose your own groups.
Anyone not in a group will be assigned a group after the lab tomorrow.