COMP 204

Algorithm design: Selection and Insertion Sort

Mathieu Blanchette
based on material from Yue Li, Christopher J.F. Cameron and Carlos G. Oliver
A **sorting algorithm** is an algorithm that takes
- a list/array as input
- performs specified operations on the list/array
- outputs a sorted list/array

For example:
- \([a, c, d, b]\) could be sorted alphabetically to \([a, b, c, d]\)
- \([1, 3, 2, 0]\) could be sorted:
  - increasing order: \([0, 1, 2, 3]\)
  - or decreasing order: \([3, 2, 1, 0]\)
Why is it useful to sort data?

Sorted data searching can be optimized to a very high level
▶ also used to represent data in more readable formats

Contacts
▶ your mobile phone stores the telephone numbers of contacts by names
▶ names can easily be searched to find a desired number

Dictionary
▶ dictionaries store words in alphabetical order to allow for easy searching of any word

Remember binary search?
Adding more algorithms to your toolbox

In the last lecture, we covered searching algorithms, specifically:

▶ linear search
▶ binary search

Today, we will cover the following sorting algorithms:

▶ selection sort
▶ insertion sort

Images for selection sort are taken from an online tutorial: https://www.tutorialspoint.com/data_structures_algorithms/
Selection sort

Conceptually the most simple of all the sorting algorithms

Start be selecting the smallest item in a list

- then place this item at the start of the list
- repeat for the remaining items in the list
  - move next smallest/largest item to the second position
  - then the next
  - and so on and so on...
  - until the list is sorted

Let’s consider the following unsorted list:

14  33  27  10  35  19  42  44
Selection sort - Iteration #1

Scan the whole list to find the smallest number (10)

Swap 14 (first element) and 10 (smallest element).
Selection sort - Iteration #2

Search for smallest element starting from second element: Find 14.

Swap 33 (second element) with 14 (smallest).
Selection sort - Iterations #3, 4, 5...

The same process is applied to the rest of the items in the list
Selection sort #6

Until the list is sorted
Selection sort (sequence)

Step 1 - find the item with the smallest value in sequence
Step 2 - swap it with the first item in sequence
Step 3 - find the item with the second smallest value in sequence
Step 4 - swap it with the second item in sequence
Step 5 - find the item with the third smallest value in sequence
Step 6 - swap it with the third item in sequence
Step 7 - repeat finding the item with the next smallest value
Step 8 - then swap it with the correct item until sequence is sorted
Algorithm 1 Selection sort

1: procedure SELECTION_SORT(sequence)
2: \[N \leftarrow \text{length of } sequence\]
3: for \(i \leftarrow 0\) to \(N - 1\) do
4: \(\text{min\_index} \leftarrow i\)
5: for \(j \leftarrow i + 1\) to \(N - 1\) do
6: \(\text{if } sequence[j] \leq sequence[\text{min\_index}] \text{ then}\)
7: \(\text{min\_index} \leftarrow j\)
8: end if
9: end for
10: SWAP(sequence[i], sequence[\text{min\_index}])
11: end for
12: end procedure
def selection_sort(sequence):
    N = len(sequence)
    for i in range(0,N):
        min_index = i
        for j in range(i+1,N):
            if sequence[j] <= sequence[min_index]:
                min_index = j
        sequence[i],sequence[min_index] = sequence[min_index],sequence[i]
    return sequence
Insertion sort

Insertion sort works by repeatedly

▶ inserting the next element of the unsorted portion of the list into the sorted portion of the list
▶ resulting in progressively larger sequences of a sorted list

Start with a sorted list of 1 element on the left and \( N-1 \) unsorted items on the right

▶ take the first unsorted item
▶ insert it into the sorted list, moving elements as necessary
▶ now have a sorted list of size 2, and \( N -2 \) unsorted elements
▶ repeat for all items
Insertion sort - Iteration 1

Let’s reuse our unsorted list from before and sort it in ascending order:

```
14  33  27  10  35  19  42  44
```

Iteration 1:
Start by finding out where to insert element at index 1 (33) into sorted portion of list (index 0 to 0):

```
14  33  27  10  35  19  42  44
```

14 > 33? no

```
14    27  10  35  19  42  44
```

put 33 back

```
14  33  27  10  35  19  42  44
```

List[0…1] is now sorted!
Insertion sort - Iteration 2

Insert element at index 2 (27) into sorted portion of list (index 0 to 1):

27

33 > 27? yes

14 33 10 35 19 42 44

14 > 27? no

14 33 10 35 19 42 44

insert 27 at index 1

14 27 33 10 35 19 42 44

List[0...2] is now sorted!
Insertion sort - Iteration 3

Insert element at index 3 (**10**) into sorted portion of list (index 0 to 2):

14 27 33 35 19 42 44

33 > 10? yes

14 27 33 35 19 42 44 10

27 > 10? yes

14 27 33 35 19 42 44

14 > 10? yes

14 27 33 35 19 42 44

List[0...3] is now sorted!
Insertion sort - Iteration 4

Insert element at index 4 (35) into sorted portion of list (index 0 to 3):

10 14 27 33 35 19 42 44

Nothing to do! List[0...4] is now sorted!
Insertion sort - Iteration 5

Insert element at index 5 (19) into sorted portion of list (index 0 to 4):

![Diagram of insertion sort iteration 5]

List[0...5] is now sorted!
Insertion sort - Iteration 6

Insert element at index 6 (42) into sorted portion of list (index 0 to 5):

Nothing to do!

List[0...6] is now sorted!
Insertion sort - Iteration 7 (last one!)

Insert element at index 7 (44) into sorted portion of list (index 0 to 6):

Sorted so far

10 14 19 27 33 35 42 44

42 > 44? no

10 14 19 27 33 35 42 44

put back 42

Nothing to do!

List[0...7] is now sorted!
We’re done!
Algorithm 2 Insertion sort

1: procedure INSERTION_SORT(sequence)
2: for $i \leftarrow 1$ to $N$ do
3:   $key \leftarrow sequence[i]$
4:   // inset key into the sorted sub-list
5:   $j \leftarrow i$
6:   while $j > 0$ and $sequence[j - 1] > key$ do
7:     $sequence[j] \leftarrow sequence[j - 1]$
8:     $j \leftarrow j - 1$
9:   end while
10: $sequence[j] \leftarrow key$
11: end for
12: end procedure
def insertion_sort(sequence):
    N = len(sequence)
    for i in range(1,N):
        key = sequence[i]
        j = i
        while(j > 0 and sequence[j-1] > key):
            sequence[j] = sequence[j-1]
            j -= 1
        sequence[j] = key
    return sequence
SelectionSort and InsertionSort can sort lists of any types of objects (numbers, strings, lists, images...), provided that we can define the comparison operator ">".

SelectionSort and InsertionSort work well on relatively small lists, but...

The amount of work done by these algorithms is proportional to the square of the length of the list.

- Sorting very large lists can take a very long time.

Many other sorting algorithms exist: MergeSort, QuickSort, etc.

- They are a bit more complicated, but
- Work much faster on large lists