Data Structures and Algorithms

XMUT-COMP 103 - 2026 T1

Algorithm Complexity

Felix Yan

School of Engineering and Computer Science

Victoria University of Wellington

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Big-O classes

- Examples:
  - O(1) constant: cost is independent of n : Fixed cost!
    - Retrieve/insert in regular arrays, hashmap operations
  - O(log n) logarithmic: cost grows by 1, when n doubles : *almost constant*
    - Traversing a binary tree, some divide-conquer algorithms
  - O(n) linear: cost grows *linearly* with n :
    - Find a value in array, do something to all elements in an array, adding in the middle of ArrayList
  - O(n log n) log linear: cost grows a bit more than linear: *Slow growth!*
    - Good sorting algorithms (merge, quick, heap sort). Complex divide-conquer algorithms

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

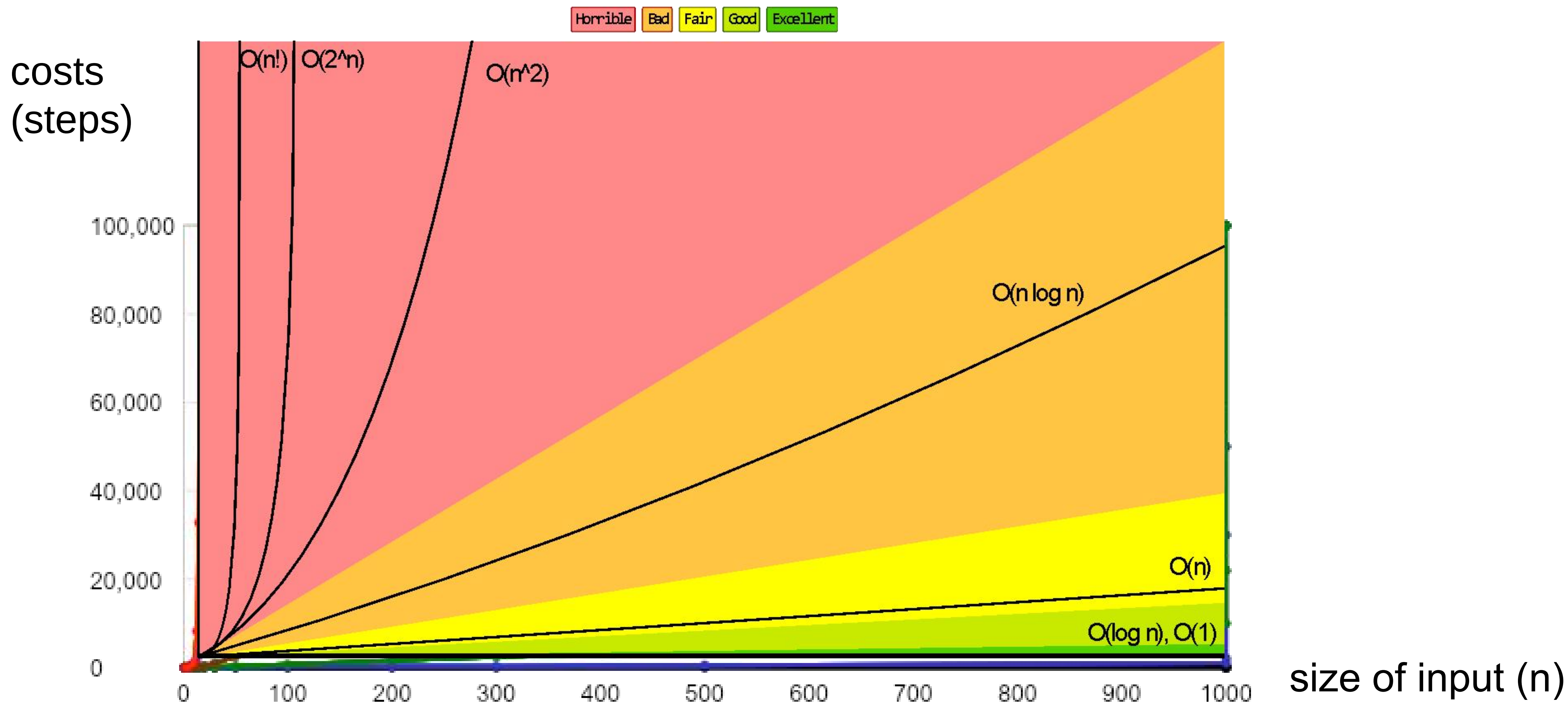
Big-O classes

- Examples continued:
  - $O(n^2)$ quadratic: costs x 4 when n doubles: *limits problem size*
    - Do something to all elements in a 2d array. Nested loops
  - $O(n^c)$, $c>2$ polynomial: *limits problem size even more*
    - Do something to all elements in a 3d array. Many nested loops
  - $O(2^n)$ exponential: costs doubles when n increases by 1: *severely limits problem size*
    - Route finding, e.g. travelling salesman problem
  - Super-exponential: e.g. $O(n!)$ *don't even think about it…*

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

How the different costs grow

- For growing n, the costs grow slower or faster depending on the cost function



© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Manageable problem sizes

- *How large can the data be?*
  - Assume one step takes one microsecond (i.e., 10^{-6} sec) on the computer
  - Then the following problem sizes can be handled by an algorithm in a given Big-O class within a given time unit

Time	1 min	1 h	1 day	1 week	1 year
O(n)	10^7	10^9	10^{11}	10^{12}	10^{13}
O(n log n)	10^6	10^8	10^9	10^{10}	10^{12}
$O(n^2)$	10^4	10^5	10^5	10^6	10^7
$O(n^3)$	10^2	10^3	10^3	10^4	10^4
$O(2^n)$	25	31	36	39	44

How much is 1 year ? about half a million sec

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

What is a “step”?

- Any important actions that are at the centre of the algorithm
  - comparing data
  - moving data
  - anything you consider to be “expensive”
  - Doesn’t depend on size of data

```
public  E  remove (int index){
    if (index < 0 || index >= count) throw new ….Exception();
    E ans = data[index];
    for (int i=index+1; i< count; i++)
        data[i-1]=data[i];                ← Key Step
    count--;
    data[count] = null;
    return ans;
}
```

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

What's a step: Pragmatics

- Count the most expensive actions?
  - Adding 2 numbers is cheap
  - Raising to a power is not so cheap
  - Comparing 2 strings *may* be expensive
  - Reading a line from a file *may* be very expensive
  - Waiting for input from a user or another program may take forever…

- Remember the Big (O) picture
- Sometimes we need to know about how the underlying operations are implemented in the computer to choose well (NWEN241/342).

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Costs of Standard Collection classes

- ArrayList: O(1): clear, add, set, remove from end:
  O(n): add, remove, contains, Collections.reverse, .shuffle
  O(n log(n)) Collections.sort,

- ArrayDeque: O(1): clear, push, pop, offer, poll, add/remove First/Last:
  O(n): contains, remove(obj)
- PriorityQueue: O(log(n)): offer, poll

- HashSet: O(1): add, remove, contains
- TreeSet: O(log(n)): add, remove, contains

- HashMap: O(1): clear, containsKey, put, get
  But depends on the cost of hashCode

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Example Algorithms

- Finding the Mode of a set of numbers
- Shuffle a List
- Find combinations of items to fill a pallett

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Finding the Mode of a list

- Mean = total/count
- Median = middle value, separating top 50% from bottom 50%
- Mode = most frequent number.

23,22,49,25,43,23,5,31,43,27,21,45,43,16,5,21,18,27,39,18,21,7,42,28,21,19

Algorithm:

- set *mode* to the first number and *modeCount* to 1
- **for** each value in the list:
  - step through the list to count how many times value occurs in the list
  - **if** *count* > *modeCount* **then** set *mode* and *modeCount* to *value* and *count*

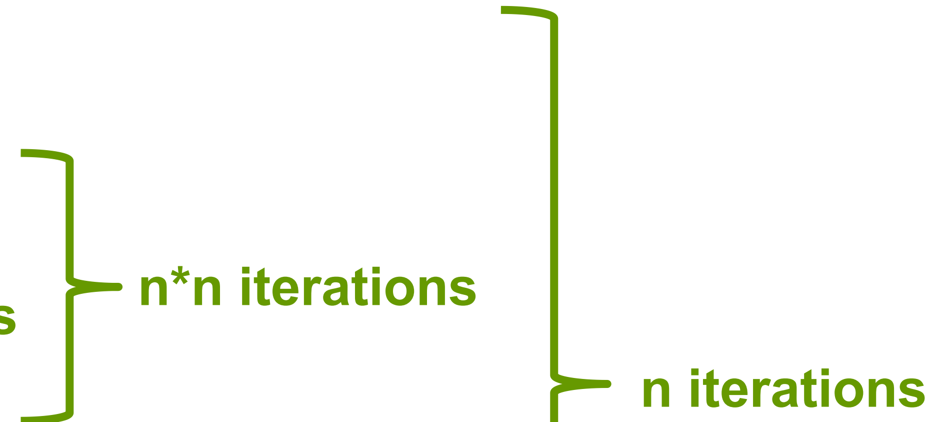
What's the cost if there are *n* numbers?

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Mode: the bad way

Analysis

```
        public int mode(List<Integer>numbers){
O(1)        int mode = numbers.get(0);            1 x O(1)
O(1)        int modeCount = 1;                    1 time
            for (int value : numbers){
O(1)            int count = 0;                    n times
                for (int other : numbers){
O(1)                if (other == value) {         nxn times
O(1)                    count++;                  n … nxn times
                    }
                }
O(1)            if (count > modeCount) {          n times
O(1)                mode = value;                 1 … n times
O(1)                modeCount = count;            1 … n times
                }
            }
O(1)        return mode;                          1 time
        }
```



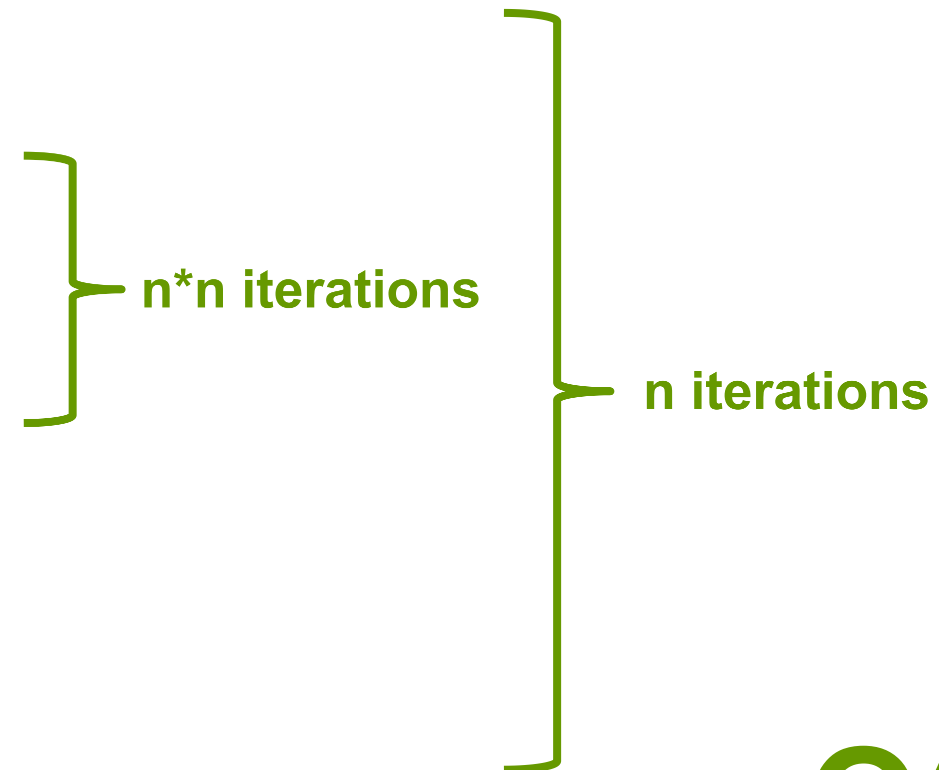
$O(n^2)$

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Mode: the bad way

Analysis

```
public int mode(List<Integer>numbers){
    int mode = numbers.get(0);              1 x O(1)
    int modeCount = 1;                      1 x O(1)
    for (int value : numbers){
        int count = 0;                      n x O(1)
        for (int other : numbers){
            if (other == value) {           nxn x O(1)
                count++;                    nxn x O(1)
            }
        }
        if (count > modeCount) {            n x O(1)
            mode = value;                   1 … n x O(1)
            modeCount = count;              1 … n x O(1)
        }
    }
    return mode;
}                                           1 x O(1)
```

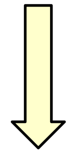


$O(n^2)$

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Finding the Mode of a list faster

- Much easier to see if the list is sorted in order:



23,22,49,25,43,23,5,31,43,27,21,45,43,16,5,21,18,27,39,18,21,7,42,28,21,19

5,5,7,16,18,18,19,21,21,21,21,22,23,23,25,27,27,28,31,39,42,43,43,43,45,49

- Algorithm
  - sort the list
  - set *mode* to first number and *modeCount* to 1
  - set *count* to 1
  - **step** through the list from index 1
    - **if** the number is the same as the previous number, **then** increment *count*
    - **else**
      - **if** *count* > *modeCount*, **then** set *mode* and *modeCount* to previous value and *count*
      - reset *count* to 1
  - **if** *count* > *modeCount*, **then** set *mode* and *modeCount* to previous value and *count*

What's the cost if there are *n* numbers?

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Finding the Mode of a list faster

Analysis

- Algorithm
  - sort the list — 1 x **O(n log(n))**
  - set *mode* to first number and *modeCount* to 1 — **1 time** x **O(1)**
  - set *count* to 1 — **1 time** x **O(1)**
  - **step** through the list from index 1 (**n iterations**)
    - **if** number is same as previous number, **then** — **n times** x **O(1)**
      - increment *count* — **1 … n times** x **O(1)**
    - **else**
      - **if** *count* > *modeCount*, **then** — **n … 1 times** x **O(1)**
        - set *mode* and *modeCount* to previous number and *count* — **n … 1 times O(1)**
      - reset *count* to 1 — **n … 1 times** x **O(1)**
  - **if** *count* > *modeCount*, **then** — **1 time** x **O(1)**
    - set *mode* and *modeCount* to previous value and *count*

Total: O(n log(n))

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Finding the Mode of a list even faster

- Count using a map to count without sorting:

23,22,49,25,43,23,5,31,43,27,21,45,43,16,5,21,18,27,39,18,21,7,42,28,21,19

5-2 7-1 16-1 18-2 19-1 21-4 22-1 23-2 25-1
27-2 28-1 31-1 39-1 42-1 43-3 45-1 49-1

What's the cost if there are *n* numbers?

- Algorithm
  - **for** each value in the list
    - **if** the value is in the map, **then** increment the associated *count*
    - **else** add the value to the map with an associated count of 1.
  - **for** each key in map,
    - **if** associated count > *modeCount*, **then** set *mode* and *modeCount* to key and count

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei

Finding the Mode of a list even faster

Analysis

- Algorithm
  - **for** each value in the list — n times
    - **if** the value is in map, **then** — **n** x **O(1)** — containskey(key)
      - increment the associated *count* — **1…n** x **O(1)** — get(..) & put(..)
    - **else**
      - add value to map with associated count =1. — **n…1** x **O(1)** — put(key, 1)
  - **for** each key in map, — **O(1)** — get all keys — n times
    - **if** associated count > *modeCount*, **then** — **n** x **O(1)** — get(key)
      - set *mode* and *modeCount* to key and count — **1…n** x **O(1)**

Total: O(n)

© Peter Andreae, Karsten Lundqvist, and Mohammad Nekooei