Data Structures and Algorithms

XMUT-COMP 103 - 2026 T1

More Examples on Algorithm Complexity

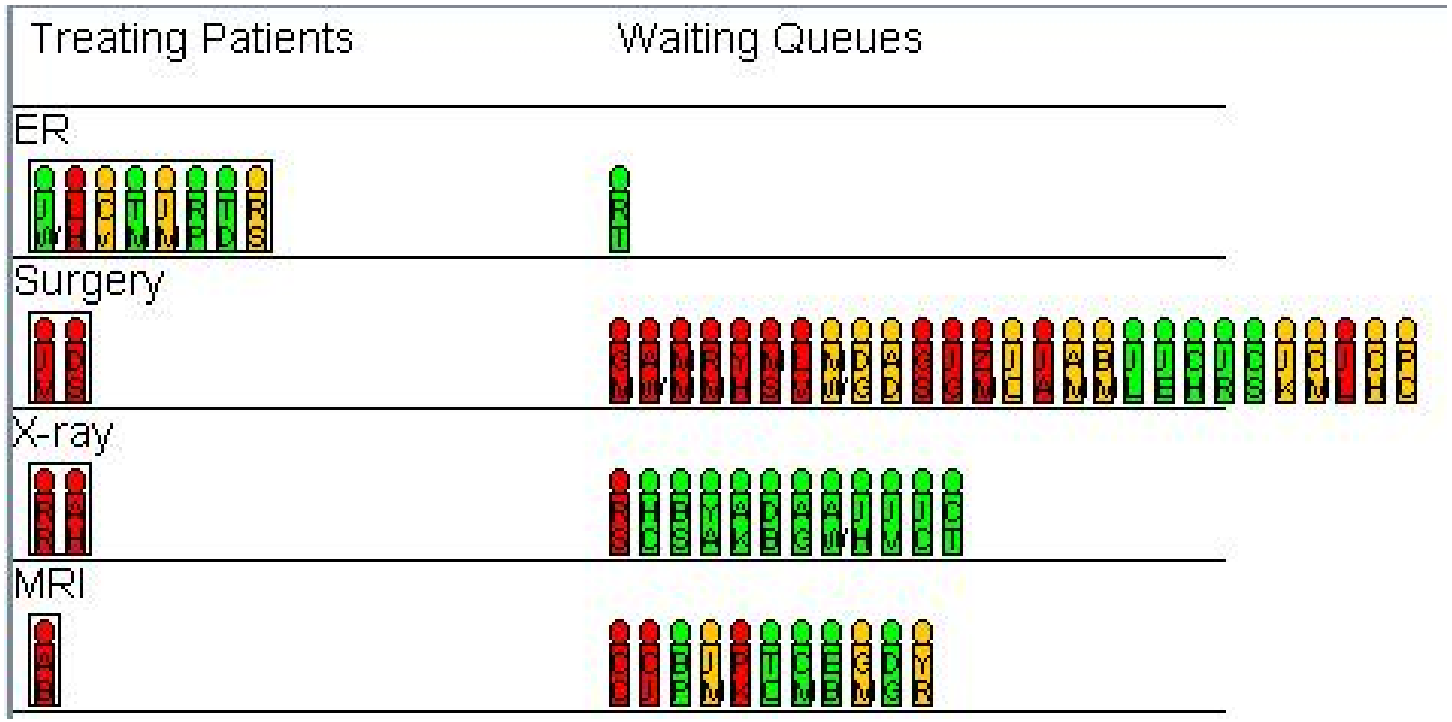
Felix Yan

School of Engineering and Computer Science

Victoria University of Wellington

Aside: Priority Queues

- Why aren't the Patients in priority order when waiting in the queue?



- Note:
  - The front item in the priority queue is always the highest priority.
  - Higher priority items tend to be closer to the front.
  - But they aren't kept in exact order.

- Priority Queues keep the items in a partially ordered tree structure

  ⇒ more efficient to add and remove items [O(log n) instead of O(n)]

  more details later in the course.

Shuffle a list

Given a list, put items into a random 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

- For each position, grab a random item and put it in that position
  - add(position, remove(random))

  vs
  - swap [set(position, set(index, get(position))] or Collections.swap(…)

- Use the built-in shuffle!
  - Collections.shuffle(list)

Shuffle a list

n times

- For each position from n-1 to 0,
  - choose a random index ≤ position
  - item = remove(index)
  - add(position, item)

n x O(1)
n x O(n)
n x O(n)

Total: $O(n^2)$

n times

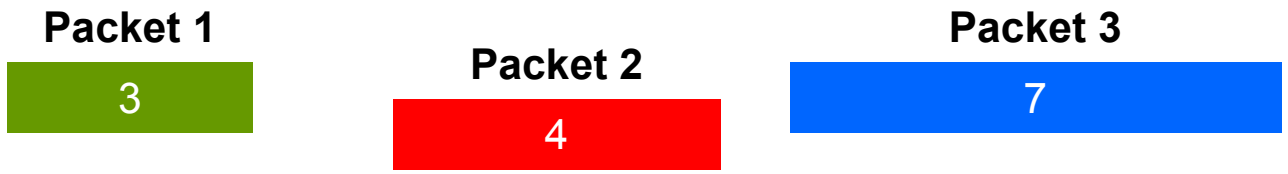
- For each position from n-1 to 0,
  - choose a random index <= position
  - swap(index, position)

n x O(1)
n x O(1)

Total: O(n)

Combinations

- Given a set of n packets of weights w_1 , ..., w_n, and a shipping pallet/container/box that has size z
  - Example:



- Given the target z, what is the largest total weight <= z that can be achieved?
  - Example:
  - z <= 10 ?

  - z <= 6 ?

Combinations – Largest total weight

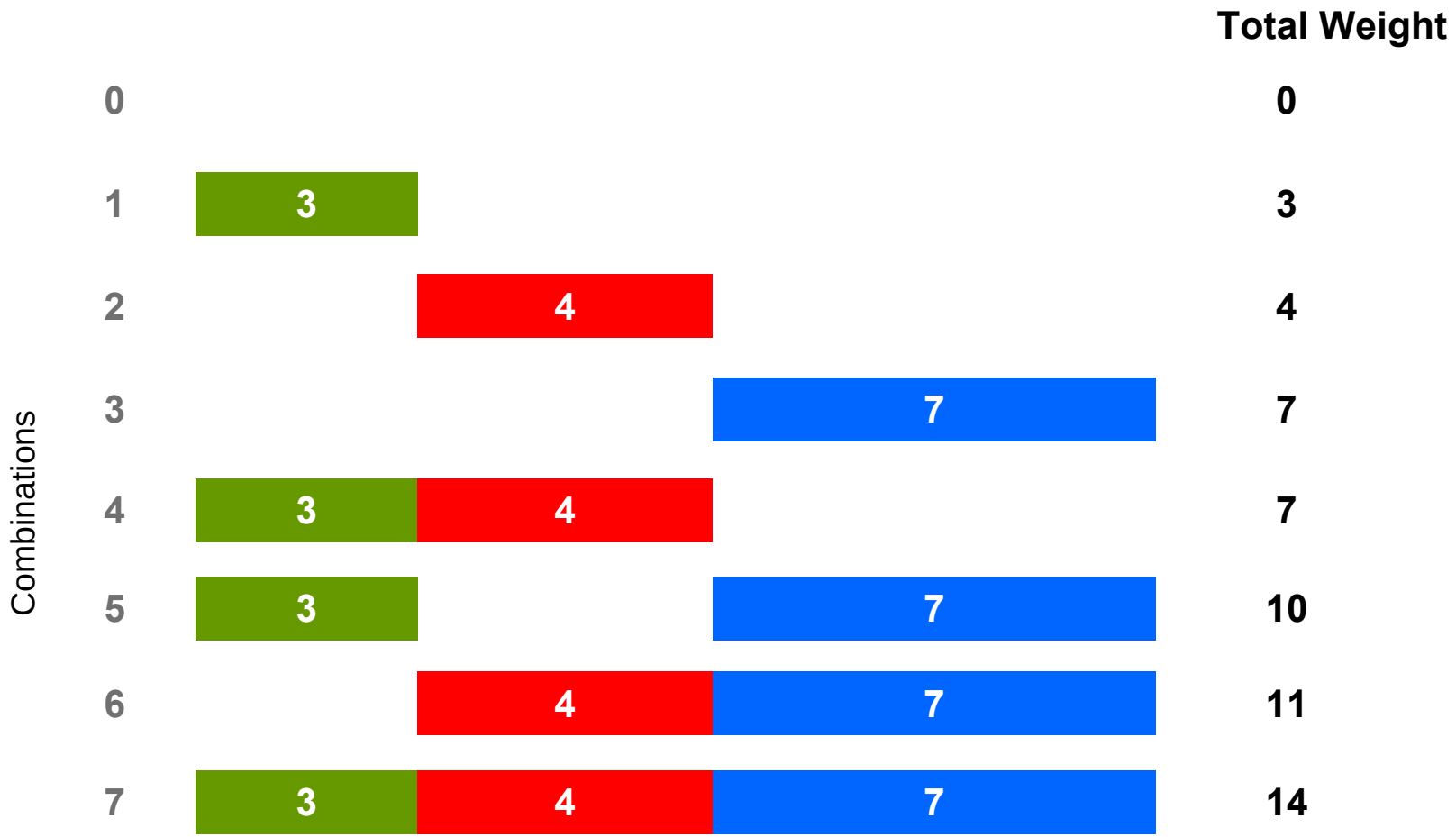
- Given a set of n packets of weights w_1 , ..., w_n
  - Example:

- What is the largest total weight of any combination?
  - Example:
  - The best combination:

  - If all weights are positive, then selecting all packets gives the largest total weight

Combinations – List all

- Can we list all combinations with their respective total weight?



- How many combinations are of n packets are there?
  - 2^n

Combinations – Selecting Packets

- How can we ensure that we did not forget any combination?
  - We just decide for each packet whether it should be selected for the combination or not
  - Yes = “packet selected”, No = “packet not selected”

Combinations	3	4	7	Total Weight
0	No	No	No	0
1	Yes	No	No	3
2	No	Yes	No	4
3	No	No	Yes	7
4	Yes	Yes	No	7
5	Yes	No	Yes	10
6	No	Yes	Yes	11
7	Yes	Yes	Yes	14

How to represent combinations?

- Anything that can be improved?
  - For an algorithm we better use 1 and 0 rather than Yes and No

Combinations	3	4	7	Total Weight
0	0	0	0	0
1	1	0	0	3
2	0	1	0	4
3	0	0	1	7
4	1	1	0	7
5	1	0	1	10
6	0	1	1	11
7	1	1	1	14

- We use a binary representation for combinations:
  - Example: 011 stand for packets 2 and 3

How to represent combinations?

- Does this idea also work for more than 3 packets?
  - Yes, here an example for n = 14:
  - 10001110011010 stands for the packets 1, 5, 6, 7, 10, 11,13

- Step through all numbers from 0 to 111 to try all combinations

  - **for** combn from 0 to 111
    - work out total weight of combination
    - **if** weight <= target and weight > best so far
      - remember weight and combn

Cost of Algorithm with loop

- if n packets, then max combination represented by 2^n
  - **for** combn from 1 to max with n packets, max = 2^n
    - work out total weight of combination O(n)
    - **if** weight <= target and weight > best so far O(1)
      - remember weight and combn O(1)

2^n times

Combinations – Can we do better?

- Given a set of n packets of weights w_1 , ..., w_n, and a target z
  - Example:

- Idea: Consider two options
- First option: if packet 1 has weight <= target z, then select it and we still have n-1 packets to choose from, but target must be reduced by the weight of packet 1

- Second option: do not select packet 1, then we still have n-1 packets to choose from, and target is still the same

Combinations – Can we use recursion?

- Idea: divide the problem (of size n) into two smaller subproblems (of size n-1)
  - So we can use recursion

- First option: if packet 1 has weight <= target z, then select it and we still have n-1 packets to choose from, but target must be reduced by the weight of packet 1

First subproblem of size n-1

- Second option: do not select packet 1, then we still have n-1 packets to choose from, and target is still the same

Second subproblem of size n-1

Combinations

- packet 0 yes no
- packet 1 yes no
- packet 2 yes no
- packet 3 yes no
- packet 4 yes no
- packet 5 yes no
- packet 6 yes no
- packet 7 yes no
- packet 8 yes no
- packet 9 yes no
- packet 10 yes no
- packet 11 yes no

Combinations – Using Recursion

- Start with an empty combination
- initialise bestCombination and bestTotal to 0;
- Find combinations using additional packets from index 0

- To find combinations using additional packets from index i…:

  // first option with first subproblem of size n-1

  - if including packet i would still be <= target
    - add it to the current combination
    - if it beats the current best, then remember total and combination.
    - find combinations using additional packets from index i+1… < RECURSIVE CALL
    - remove it from the current combination

  // second option with second subproblem of size n-1

  - find combinations using additional packets from index i+1… < RECURSIVE CALL

Cost of Algorithm with recursion

- Cost(n) = cost of finding with n remaining packets to try

- Cost(1) = O(1)
- Cost(n) = O(1) + Cost(n-1) + Cost(n-1)
  = 2 Cost(n-1) + O(1)
  = 2(2Cost(n-2) + O(1)) + O(1)

The cost approximately doubles when n increase by 1 => O(2^n)