(this was empty so I thought I’d add my notes, but I fucking suck at computer science so good luck)

Basic definition:

- given set of n items and a knapsack of limited capacity C

- each item has a weight w and a value v

- need to find best way to fill up the knapsack, maximising overall value

- 0/1 knapsack - items cannot be divided, must be whole

- fractional knapsack - items can be divided (e.g. can put half of something in)

- basically figuring out the best way to pack your pick-and-mix cup

Algorithms:

Enumeration:

* compute all possible solutions of packing items
* choose highest value set that is less than or equal to capacity in weight
* simple to implement
* very slow on big inputs
* O(2^n)

Greedy:

* first (preprocessing) sort items in decreasing value/weight ratio
* add items into knapsack in that order, skipping any that break the capacity limit
* choosing ‘best’ items first and not backtracking - committing to any you put in
* sort of like a fat man eating all the best cakes first (chocolate -> toffee -> others)
* O(nlogn) including sorting

Branch-and-bound:

* first (preprocessing) sort items by decreasing value/weight ratio
* compute upper bound of solution
* find upper bound by putting in items like greedy until you find the item that breaks the capacity limit, s then: upper bound = ((capacity left / Weight of s) \* value of s) + values already in
* something something I don’t really get it past here (check facebook?)
* O(2^n)

Dynamic Programming:

* god help me, idk man
* there’s good videos out there, check the facebook page
* decompose problem into smaller problems to solve (divide and conquer)
* recursively define value of an optimal solution in terms of solutions to smaller problems
* O(nC) where C = capacity