3-sum

- Find all triples of numbers (x_i, x_j, x_k) from the indexed set X where $x_i + x_j + x_k = 0$
- How do we do this? Brute force implementation is going to involve three nested *for* loops, i.e. it will be $O(N^3)$
- Is there an easy way to improve it?
  - Reduction
  - Memoization
  - (Dynamic Programming)
- Example set: -3, -1, 1, 2, 4, 5

Reduction

1. Transform domain of problem A into domain of problem B.
2. Solve problem B.
3. Transform domain of problem B back into domain of problem A.

Improving 3-sum (our problem A)

- Problem B is the following:
  - Given a table of pairs of numbers, indexed by their sum, find, for a value v, every pair p_j such that $v = -p_j$

Sum	**Pairs**	
-4	-3,-1	
-2	-3,1	
-1	-3,2	
0	-1,1	
1	-3,4	-1,2
2	-3,5	
3	-1,4	1,2
4	-1,5	
5	1,4	
6	1,5	2,4
7	2,5	
9	4,5	

Our reduction of A->B

- Build the *sums* table (i.e. memoization)
  - This will take time proportional to N^2
- For every element x_i in the set, get the set of pairs P_i from the table, thus forming a set of tuples: $x_i \rightarrow P_i$ where $P_i = (x_j, x_k)$
  - This will take time proportional to $N \lg s$ where s is the number of sums
- Transform the set P into the set R where
  $R_i = (x_i, x_j, x_k)$
  - This will take time proportional to N (at worst)

Total time for reduction?

- N^2 instead of N^3
- What did it cost us?
  - Memory space: proportional to $N^2 + s$
  - We also need an algorithm to find the relevant index (and pairs) from the table without searching one-by-one (but even that isn't essential)

Merge sort

- This is the example of reduction that I discussed on the blackboard (hopefully):
    - Step 1: transform problem A (sorting an array of length N) into problem B (sorting two arrays each of length $N/2$);
    - Step 2: solve (recursively) each of the parts of problem B (when N gets below a threshold k, we use insertion sort: takes a total of $N^2/2$ time);
    - Step 3: transform the solution to problem B (i.e. two sorted sub-arrays) into the domain of problem A (by merging the two sorted sub-arrays into a sorted version of the original array [this operation takes linear time].
- We can show (later) that the entire operation takes time proportional to $N \log_2 N$.

In general…

- We will use this type of technique (reduction) all the time throughout this course.
- Think of an algorithm that takes time t for N elements where:
  - $t = c N^k$
- We may be able to use reduction to a problem whose solution takes time $t' = c' N^{k'}$ where $c' < c$ or where $k' < k$ or…
- Perhaps we can reduce it to two (or more) problems where each problem involves a subset of N: this is the famous “divide and conquer” technique.

What else?

- Quantum Computing