Problem 1: Alice wants to throw a party and is deciding whom to call. She has \( n \) people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know and five other people whom they do not know. Give an efficient algorithm that takes as input the list of \( n \) people and the list of pairs who know each other and outputs the best choice of party invitees.
**Problem 2:** Suppose you are choosing between the following three algorithms:

- **Algorithm A** solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- **Algorithm B** solves problems of size $n$ by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
- **Algorithm C** solves problems of size $n$ by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-$O$ notation), and which would you choose?
Problem 3: Given a sorted array of distinct integers $A[1, \ldots, n]$, you want to find out whether there is an index $i$ for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$. 
Problem 4: $A[1, \ldots, n]$ is said to have a **majority element** if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form “is $A[i] > A[j]$?”. (Think of the array elements as GIF files, say.) However you can answer questions of the form: “is $A[i] = A[j]$?” in constant time.

1. Show how to solve this problem in $O(n \log n)$ time.
2. Can you give a linear-time algorithm?