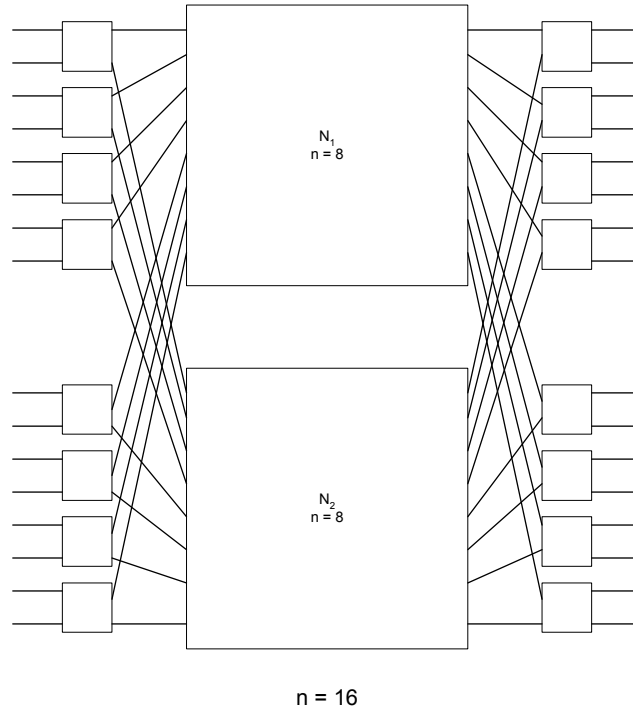


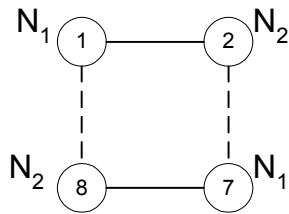
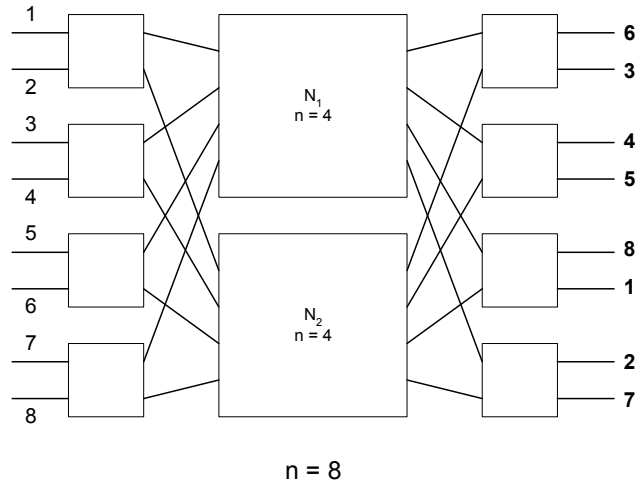
Solution #3:

1. Draw the permutation network for $n = 16$ using subnetworks for $n = 8$ as boxes with 8 inputs and outputs

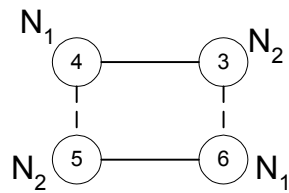


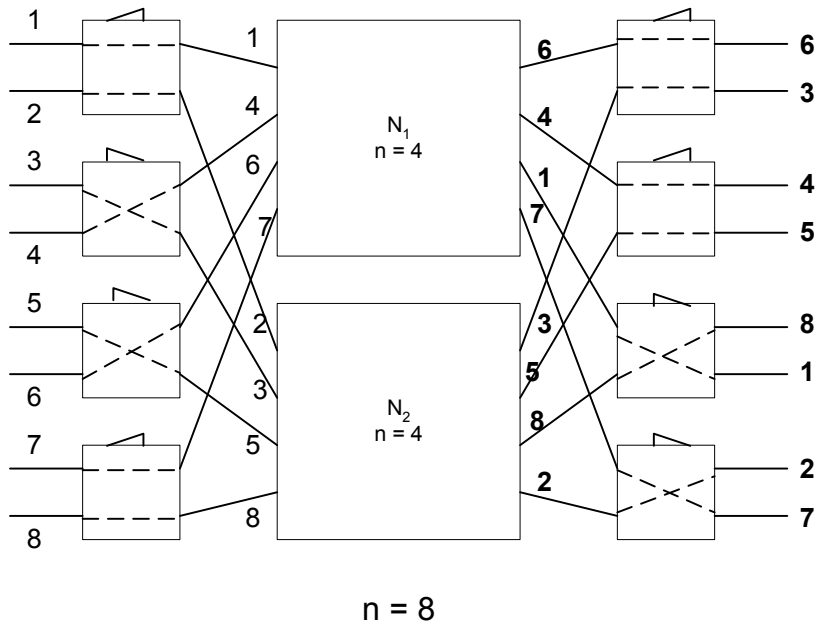
2. Show switch settings for the following using the algorithm described in

class: Show the settings for all switches.



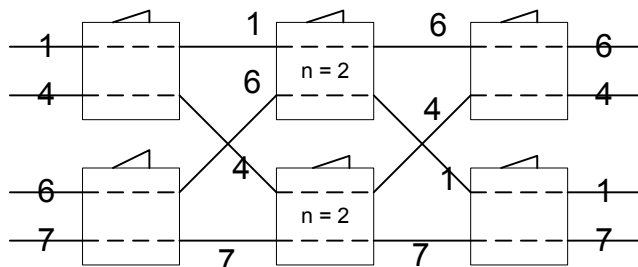
Forbidden Pairs Diagram



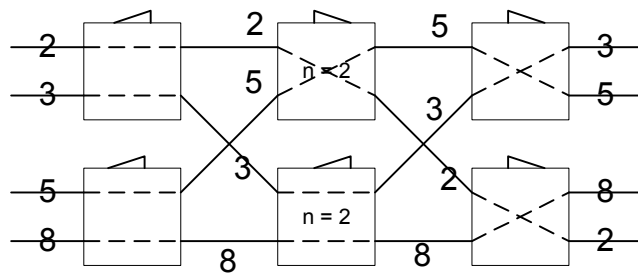
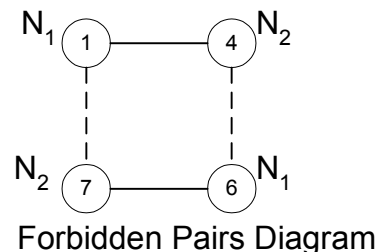


This gives us two smaller problems to do one for each block above. These

are shown below:

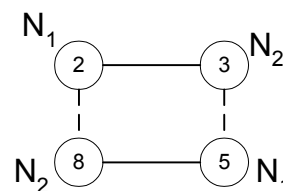


N_1



N_2

Forbidden Pairs Diagram



3. We give below a divide-and-conquer algorithm whose worst case complexity is $O(n \lg n)$. It is possible to give one whose complexity is $O(n)$.

We will divide the data set into two roughly equal size sets each time and call the main subroutine on the smaller problems until the size becomes very small. In this process we will have to solve problems on arrays of the type $A[i, i + 1, \dots, j]$ in general. If $|j - i| \leq 3$, then we can solve the problem in constant time by "brute-force" enumeration. If $|j - i| > 3$, then we subdivide into two arrays $A[i, i + 1, \dots, i + \lfloor \frac{j-i}{2} \rfloor]$, and $A[\lfloor \frac{j-i}{2} \rfloor + 1, \dots, j]$ and solve the "main" problem on these two subarrays by recursively calling the main algorithm. This gives us the maximum value subarrays in each part. But the maximum sum subarray in the array $A[i, \dots, j]$ might "straddle" this partition in the sense that it contains elements from both sides. To check this and find such a subarray if this is the case, we have to do additional work as in the "merge" part of merge-sort. For this we want an algorithm that finds the $\max_{p: i \leq p \leq k+1} [\sum_{l=p}^k A[l]]$ (where we interpret $\sum_{l=k+1}^k A[l]$ as 0) in an array of the form $A[i, i + 1, \dots, k]$. This can be easily done in $O(n)$ time. We do this with $k = i + \lfloor \frac{j-i}{2} \rfloor$ for the "left" half of our array. Similarly we can find $\max_{p: k+1 \leq p \leq j} [\sum_{l=k+1}^p A[l]]$ in the

array $A[k + 1, \dots, j]$ with $k = i + \lfloor \frac{j-i}{2} \rfloor$. These two together give us the straddling subarray that is the maximum value and we can compare this the results in each part to get the overall maximum. So the "merge" part takes $O(n)$ time. Hence our overall recurrence relation is:

$$t(n) = t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + t\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$$

whose solution by master theorem is $t(n) = \Theta(n \lg n)$.

4. Professor Diogenes has n supposedly identical VLSI chips that in principle are capable of testing each other. The professor's test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately the condition of the other chip; but the answer from a bad chip is not to be trusted. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	either both are good or both are bad
B is good	A is bad	at least one is bad; actually we can show that A is bad
B is bad	A is good	at least one is bad; actually we can show that B is bad
B is bad	A is bad	at least one is bad

Assume that more than $\frac{n}{2}$ chips are good. (this is the same as saying that the number of good chips is strictly greater than that of bad chips).

(i) Show that $\frac{n}{2}$ pair-wise tests are sufficient to reduce the problem size to one that is no more than half the size of the original problem.

(ii) Using the above show how to identify at least one good chip. Write down the recurrence relation that arises out of this and using the master theorem solve this recurrence relation. Once one good chip is identified, can we identify which of the remaining chips are good and which are bad?

Solution:

(i) Here is the complete solution:

Suppose we pair chips and test each pair: $\{1, 2\}, \{3, 4\}, \dots, \{2 \lfloor \frac{n}{2} \rfloor - 1, 2 \lfloor \frac{n}{2} \rfloor\}$. Let T_1 be the set of pairs where exactly one of the pair says that the other is bad. If there is a pair $A - B$ in T_1 in which A says that B is good and B says that A is bad, then we know that A is truly bad (**we can not conclude any thing about B**). We **can and do** remove **both chips in all such pairs from further consideration** and maintain the property that there are more good chips than bad ones. (But a clever adversary may produce answers that contain no such pairs). Let T_2 be the set of pairs where each chip says that the other is bad. We know that in each pair in T_2 there is **at least one bad chip**. We can **not** eliminate **one** of these and retain the other – because the chip that eliminated might be a good one and this might reduce the count of good ones. So we **must**

and do eliminate both such chips out of these pairs in T_2 . This still preserves the property that there are more good than bad chips in the set that remains. When we eliminate the set $T_1 \cup T_2$ from further consideration, the remaining number may not be a power of 2 even if we started with a power of 2. So the assumption that the number of chips is a power of 2 is useless (and so I have deleted it in this version).

Let S be the set of pairs in the category where each chip says the other is good. [Each pair is in $T_1 \cup T_2 \cup S$ and these sets are not overlapping.] For each pair in S , either both are good or both are bad. Eliminating both might reduce the number of good chips without concurrently reducing the number of bad chips and so we might end up with more bad chips in the remaining set of chips. Keeping both might **not** reduce the set under consideration down to half the size of the original set. **So we can not eliminate both; we can not keep both.** So we must eliminate **one** from each pair **without knowing whether the pair is truly good or truly bad.** But we have to be careful. **In case n is odd, last chip is not tested in this round. We can either keep it for further consideration or not.** But whatever we do, we need to **make sure that in the set that remains there are strictly more good than bad chips.** For this purpose, **we keep the last chip when n is odd if the number of pairs in S is even and do not keep it if this number is odd.** If the number of pairs in S is even, then we could have the possibility of an equal number of truly good pairs and bad pairs. If we retain one from each pair, then the number of good and bad chips might be equal if we do not include the last chip when n is odd. Fortunately in this case, the last chip is truly good and so retaining it for further tests preserves the property that there are more good than bad. If the number of pairs is odd, the fact that there are more good than bad before elimination, tells us that the number of truly good pairs exceeds the number of truly bad pairs. So in this case we do not retain the last chip when n is odd. This is the basis of our divide and conquer strategy. This yields a recurrence relation:

$$\begin{aligned} t(n) &= t\left(\left\lceil \frac{n}{2} \right\rceil\right) + \left\lfloor \frac{n}{2} \right\rfloor \\ t(1) &= t(2) = 0; t(3) = 1 \end{aligned}$$

Since we have eliminated at least roughly half the chips in this process by doing $\lfloor \frac{n}{2} \rfloor$ tests. This recurrence has a solution $t(n) = O(n)$. Actually you can show that this recurrence relation has a solution $T(n) \leq n - 2$ by induction.

Once a good chip is found (when we reach $n = 1$ or 2 as a boundary condition), then we can use this to check all others in $(n - 1)$ tests giving a total of $(2n - 3)$ tests.