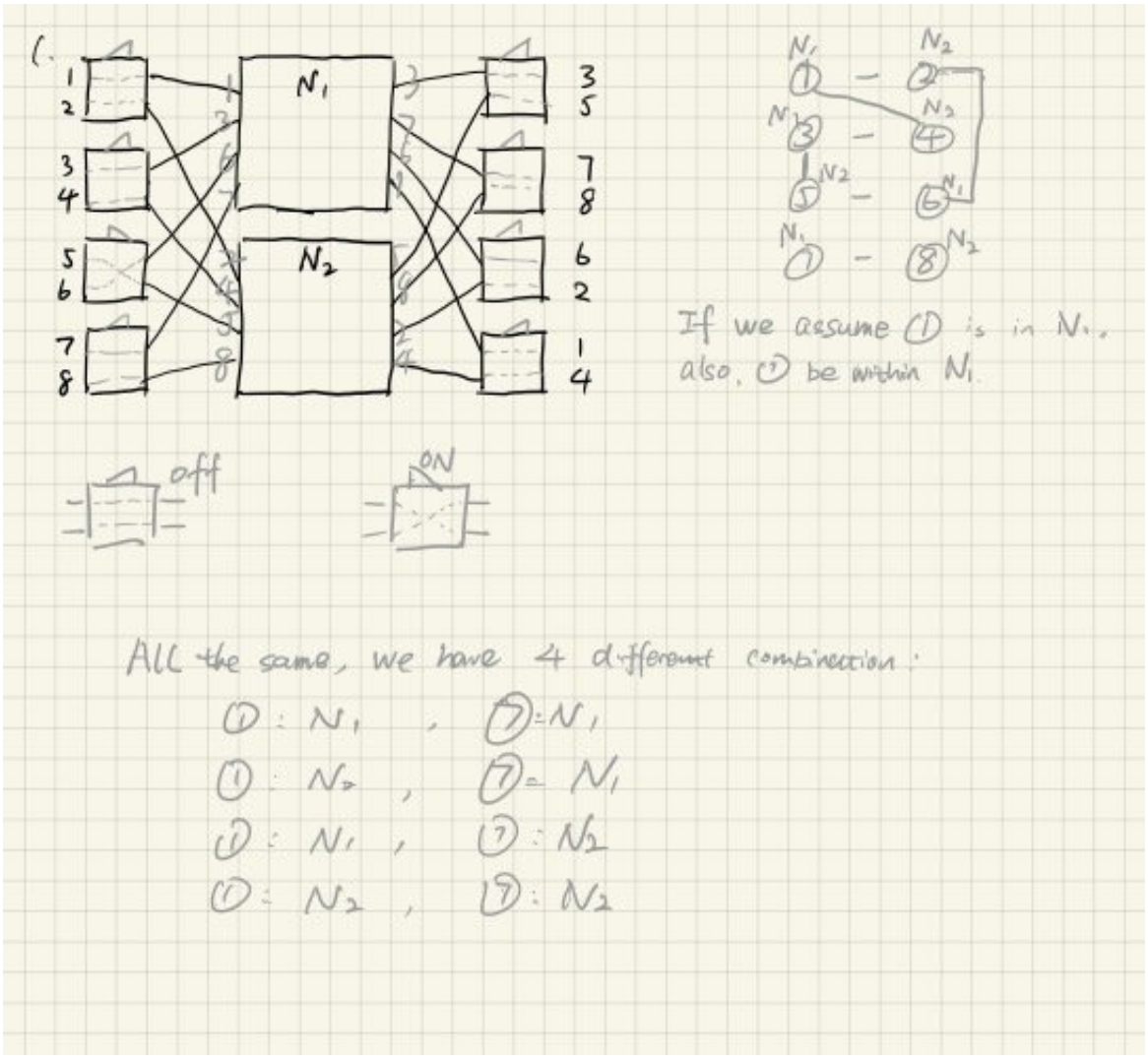
Assignment 3

Jiadao Zou --- jxz172230

Q1

> 1. Show switch settings for the following output permutation in an 8×8 switch using the algorithm described in class: **Show the settings for all switches**. [3,5,7,8,6,2,1,4] The input permutation is [1,2,3,4,5,6,7,8] as usual.



Q2 Problem.4-5

> Professor Diogenes has n supposedly identical integrated-circuit chips that in principle are capable of testing each other. The professor's test jig accomodates 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 whether the other chip is good or bad, but the professor cannot trust the answer of a bad chip. Thus, the four possible outcomes of a test are as follows:
>
> | Chip A says | Chip B says | Conclusion |
> |---|---|---|
> | B is good | A is good | both are good, or both are bad |
> | B is good | A is bad | at least one is bad |
> | B is bad | A is good | at least one is bad |
> | B is bad | A is bad | at least one is bad |
>
> **a.** Show that if more than $n/2$ chips are bad, the professor cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test. Assume that the bad chips can conspire to fool the professor.
>
> **b.** Consider the problem of finding a single good chip from among n chips, assuming that more than $n/2$ of the chips are good. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.
>
> **c.** Show that the good chips can be identified with $\Theta(n)$ pairwise tests, assuming that more than $n/2$ chips are good. Give and solve the recurrence that describes the number of tests.

a.

- According to the question, we suppose if there are $m \in [0, \frac{n}{2})$ good chips and $n - m$ bad chips.
- Since $m < n - m$, there are always a group of Good chips and a group of Bad chips of the same size.
- If such mirror situation happens, "Bad chips declare bad chips are 'Good' and good chips are 'Bad'. In the mean time, good chips would make a symmetric statement."
- Thus , it is impossible to know the truth from above information.

b.

- Arbitrarily pair up the chips. Only care about the pairs for which both chips says other is good. Since we have at least half of the chips are good, there will be at least one such pair, as well as, at least half of these pairs which both are good are actually good. Thus, dropping a chip in such pairs would never make bad chips be more than good ones.
- So, we just need to arbitrarily pick a chip from each pair (leave the last one alone, if

the total number of remaining chips is odd) and use these chips to make up the sub-instance of the problem.
- If there is only 1 chip left, then it is the good chip. Else if there is no chip remaining, the chip we leave alone at the very beginning is the good one.

c.

- From b, we get a good chip. We could just use it to query every other chip. The recurrence to find it is

$$T(n) \leq T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor$$

- By Master Theorem, we have $T(n) = O(n)$. After finding a good chip, we make pair with the remaining $n-1$ ones, so the total number is $O(n) + (n-1) = \Theta(n)$

Q3 Problem.4-6

> An $m \times n$ array A of real numbers is a **Monge array** if for all i, j, k, and l such that $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, we have
>
> $$A[i,j] + A[k,l] \leq A[i,l] + A[k,j].$$
>
> In other words, whenever we pick two rows and two columns of a Monge array and consider the four elements at the intersections of the rows and columns, the sum of the upper-left and lower-right elements is less than or equal to the sum of the lower-left and upper-right elements. For example, the following array is Monge:
>
> $$\begin{array}{ccccc} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{array}$$

a. Prove that an array is Monge if and only if for all $i = 1, 2, \ldots, m-1$, and $j = 1, 2, \ldots, n-1$ we have

$$A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j].$$

(*Hint:* For the "if" part, use induction seperately on rows and columns.)

b. The following array is not Monge. Change one element in order to make it Monge. (*Hint:* Use part (a).)

$$\begin{array}{cccc} 37 & 23 & 22 & 32 \\ 21 & 6 & 7 & 10 \\ 53 & 34 & 30 & 31 \\ 32 & 13 & 9 & 6 \\ 43 & 21 & 15 & 8 \end{array}$$

c. Let $f(i)$ be the index of the column containing the leftmost minimum element of row i. Prove that $f(1) \leq f(2) \leq \cdots \leq f(m)$ for any $m \times n$ Monge array.

d. Here is a description of a divide-and-conquer algorithm that computes the leftmost minimum element in each row of an $m \times n$ Monge array A:

Construct a submatrix A' of A consisting of the even-numbered rows of A. Recursively determine the leftmost minimum for each row in A'. Then compute the leftmost minimum in the odd-numbered rows of A.

Explain how to compute the leftmost minimum in the odd-numbered rows of A (given that the leftmost minimum of the even-numbered rows is known) in $O(m + n)$ time.

e. Write the recurrence describing the running time of the algorithm described in part (d). Show that its solution is $O(m + n \log m)$.

a.

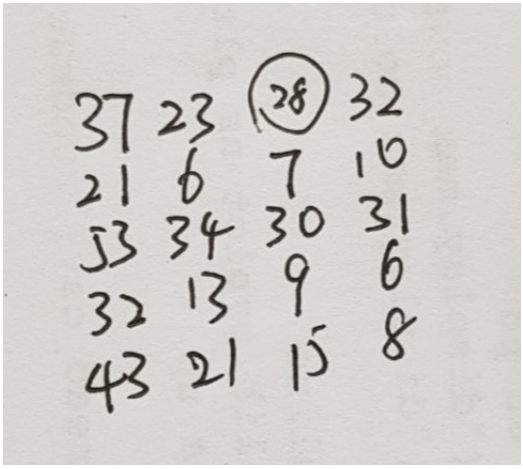
- The "if" part is trivial, because according to the definition: $A[i, j] + A[k, l] \leq A[i, l] + A[k, j]$, we just need to take $k = i + 1 \; and \; l = j + 1$.
- Prove by induction:
  - Base case: $k = i + 1$, assume it holds for $k = i + n$, and we want to show it works for $k + 1 = i + n + 1$.
  - Deduction steps:

$$A[i,j]+A[k,j+1] \le A[i,j+1]+A[k,j]$$
$$A[k,j]+A[k+1,j+1] \le A[k,j+1]+A[k+1,j]$$

$$\text{Put above two equation together}$$

$$A[i,j]+A[k,j+1]+A[k,j]+A[k+1,j+1] \le A[i,j+1]+A[k,j]+ A[k,j+1]+A[k+1,j]$$
$$A[i,j]+A[k+1,j+1] \le A[i,j+1]+A[k+1,j]$$

b.



c.

- Let a_i and b_j be the leftmost minimal elements on row a and b, and we know $a < b$. We try to prove by contrary, assume $i > j$, thus we get]

$$A[a,j]+A[b,i] \le A[a,i]+A[b,j]$$

- But based on our assumption, $a_i,\ b_j$ is the minimal:
  - $A[a,i] \le A[a,j]$ and $A[b,j] \le A[b,i]$
  - $A[a,i]+A[b,j] \le A[a,j]+A[b,i]$
- Thus, we get $A[a,i]+A[b,j] = A[a,j]+A[b,i]$
  - which implies "$A[a,j = A[a,i]$", so instead of $a_i,\ a_j$ is the left most minimal. Obviously, this is opposite to the assumption $i > j$.

d.

- From above, we assume k_i is the index of the i–th row's leftmost minimum, then $k_{i-1} \le k_i \le k_{i+1}$.
- For all positive odd numbers $i = 2h+1, h \ge 0$, k_i is locates within $[k_{i-1}, k_{i+1}]$. Thus we could have the recurrence:

$$
\begin{aligned}
T(m,n) &= \sum_{h=0}^{m/2-1}(k_{2h+2}-k_{2h}+1) \\
&= \sum_{h=0}^{m/2-1} k_{2h+2} - \sum_{h=0}^{m/2-1} k_{2h} + m/2 \\
&= \sum_{h=1}^{m/2} k_{2h} - \sum_{h=0}^{m/2-1} k_{2h} + m/2 \\
&= k_m - k_0 + m/2 \\
&= n - 0 + m/2 \\
&= O(m+n)
\end{aligned}
$$

e.

- The divide time is $O(1)$, the conquer part is $T(m/2)$, and the merge part is $O(m+n)$. Thus we could have the recurrence:

$$
\begin{aligned}
T(m) &= T(m/2) + cn + dm \\
&= \sum_{i=0}^{\lg m-1} cn + \sum_{i=0}^{\lg m-1} \frac{dm}{2^i} + \Theta(1^{\lg m}) \\
&= cn\lg m + dm\left(\frac{1-\frac{1}{2^{\lg m-1}}}{1-\frac{1}{2}}\right) \\
&< cn\lg m + 2dm \\
&= O(n\lg m + m)
\end{aligned}
$$

Q4 9.3-1

> In the algorithm SELECT, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Argue that SELECT does not run in linear time if groups of 3 are used.

- Divided into groups of 7?
  - The median of median group is still less than at least 4 elements from half of the $\lceil n/7 \rceil$ groups (similarly, it also applies to the number less than x):

    $$
    4(\lceil \frac{1}{2} \lceil \frac{n}{7} \rceil \rceil - 2) \geq \frac{2n}{7} - 8
    $$

    and the recurrence becomes

$$
\begin{aligned}
T(n) &\le T(\lceil \frac{n}{7} \rceil) + T(n - (\frac{2n}{7} - 8)) + O(n) \\
&= T(\lceil \frac{n}{7} \rceil) + T(\frac{5n}{7} + 8)) + O(n) \\
&\le cn(1/7 + 5/7) + O(n) \\
&= O(n)
\end{aligned}
$$

- Divided into groups of 3?
  - Following the similar thought, we have

    $$2(\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil - 2) \ge \frac{n}{3} - 4$$

    and the recurrence becomes

    $$T(n) \le T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} + 4)) + O(n)$$

    We assume $T(n) \ge cn \lg n$:

    $$
    \begin{aligned}
    T(n) &\ge c(n/3)\lg(n/3) + c(2n/3)\lg(2n/3) + O(n) \\
    &\ge cn \lg n + cn(2/3 - \lg 3) + O(n) \\
    &= \Omega(n \lg n)
    \end{aligned}
    $$

Q5 9.3-8

> Let $X[1..n]$ and $Y[1..n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\lg n)$-time algorithm to find the median of all $2n$ elements in arrays X and Y.

- if n is a power of 2

```
--------------
Median(X,Y)
--------------
n = X.length
if n == 1:
    return min(X[1], Y[1])
elif X[n/2] < Y[n/2]:
    return Median(X[n/2+1...n], Y[1...n/2], n/2)
else:
    return Median(X[1...n/2], Y[n/2+1...n], n/2)
```

- n is not a power of 2
  - Since we have 2n number of elements. We define the lower median value as m.

- Suppose m is in $X[k-1]$, then there are k elements less or equal to m and $m-k$ elements equal or larger than m. Since we know for the combined array, half numbers must be less or equal to m, and the second half should be equal or larger than m. Thus, there must be $n-k$ elements in Y that are less or equal to m, while k elements of Y should be equal or larger than m.
- Thus we could check the kth element $X[k-1]$ is the m by checking whether $Y[n-k-1] \leq X[k-1] \leq Y[n-k]$, if $k=n$, check $X[n-1] \leq Y[0]$.
- Since we assume median in X, if it is in Y, we gonna replace X, Y's order.
- Then we just apply binary search to look up the median.

```
---------------------
MedianOFTwoArray(X, Y):
---------------------
    n = X.length
    median = FindMedian(X, Y, n, 0, n-1)
    if median == None:
        median = FindMedian(Y, X, n, 0, n-1)
    return median

--------------------------------
FindMedian(X, Y, n, low, high):
--------------------------------
    if low > high:
        return None
    else:
        k = floor((low + high) / 2)
        if k == n-1 and X[n-1] <= Y[0]:
            return X[n-1]
        elif k < n-1 and Y[n-k-1] <= X[k] <= B[n-k]:
            return X[k]
        elif X[k] > Y[n-k]:
            return FindMedian(X, Y, n, low, k-1)
        else:
            return FindMedian(X, Y, n, k+1, high)
```

Q6

The following problem is taken from the book by Tardos and Kleinberg: Given a complete binary tree with $n = 2^d - 1$ nodes for some positive integer d. Each node u is associated with a number x_u. A node v in the tree is a local minimum if x_v is less than the value associated with its neighbors (children or parent). You may assume that numbers associated with nodes are distinct (i.e. no two are equal). We want to determine a local minimum. The only operation allowed, in addition tocomparisons, is to query a node for its value and this is counted as one operation. Design a divide-and-conquer algorithm, set up the corresponding recurrence relation and solve it to determine the complexity of your algorithm. The better the complexity the higher your score for this problem.

- Since it has $2^d - 1$ nodes, it is a full binary tree. According to definition, a local minimum should be less than the value of its parent, two children.
- So the method is
    1. Check the root, if it is smaller than both its children, then we got one local minimum. If not, we pick the smaller child and recursively apply this check.
    2. Termination: either a) find a node than is smaller than both its children, or b) we reach the leaf
        - a) it is a local minimum because it is smaller than its parent and children.
        - b) This node's children are both leaves. If one of its child is larger than it, then we only make the smaller child as local minimum, otherwise if both children are smaller than this node, local minimize the children.
    3. Complexity Analysis: At most 2 new nodes per level are being looked at, and at most 3 comparison are made, and the depth $d = \lg n + 1$, so we have $T(n) = O(\lg n)$
    4. Algorithms:

```
 -------------------------------
FindLocalMin(node, LocalMinArray):
-------------------------------
    if not(node.leftchild):
        # full tree, either 2 children or none child
        return none
    elif node.val < node.leftchild.val and node.val < node.rightchild.val:
        LocalMinArray.extend(node.val)
        return FindLocalMin(node.leftchild)
        return FindLocalMin(node.rightchild)
    else:
        if  node.leftchild.val < node.val:
            return FindLocalMin(node.leftchild)
```

```
14          else:
15              return FindLocalMin(node.rightchild)
```

Challenge problem 1:

- If the graph is rectangular grid graph:
  1. pick any vertex to be the initial current vertex
  2. probe the value of current vertex and all of its neighbors'.
  3. If the current one has the lowest label, halt and report it as local minimum.
  4. Else, select the neighbor with the lowest label as the new current vertex and return to Step2.
- We don't do backtrack and we could see this graph as triple tree, which a node has a parent and three children. The complexity may be, I guess, as $O(\log_3 n)$