University of Chinese Academy of Sciences

Assignment One
Divide-and-Conquer

姓　名：　　张　帅

学　院：网络空间安全学院

学　号：　201828018670119

中国科学院大学

2018 年 10 月

1 PROBLEM ONE

1.1 Description

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values, so there are 2n values total and you may assume that no two values are the same. You'd like to determine the median of this set of 2n values, which we will define here to be the nth smallest value.

However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most O(log n) queries.

1.2 Basic Idea

The idea is similar to median of group medians algorithm[Blum et al,1973]

Since we can only query the kth smallest value from two separate databases, so each database can be seen an ordered digit sequence(Sequence A and B, both size are n).

Each subprogram, we will find the median of sequence A and B , A_{mid}, B_{mid}, then we compare the two values, to determine the next subprogram's query range. If $A_{mid} > B_{mid}$, the median must included in the left half of A or the right half of B; If $A_{mid} < B_{mid}$, the median must included in the right half of A or the left half of B. We only need to recurse this algorithm, return the lower value when the size is 1 in the subprogram.

1.3 Pseudo-Code

PROBLEM 1 Find median from two seprate databases

INPUT: Two seprate databases, A and B; Low bound of each database's query range, low_A, low_B; The size of query range, n.

OUTPUT: Return the median of the two databases.

1: **function** FIND_MEDIAN(low_A, low_B, n)
2:     $mid\_A \leftarrow low\_A + (n-1)/2$
3:     $mid\_B \leftarrow low\_B + (n-1)/2$
4:     **if** $n\%2! = 0$ **then** $mid\_B \leftarrow mid\_B - 1$
5:     **end if**
6:     $A_{mid} \leftarrow$ QUERY(A, mid_A)
7:     $B_{mid} \leftarrow$ QUERY(B, mid_B)
8:     **if** $n == 1$ **then**
9:         **return** MIN(A_{mid}, B_{mid})
10:     **else if** $A_{mid} < B_{mid}$ **then**
11:         **return** FIND_MEDIAN($mid\_A + 1, low\_B, n - (mid\_A - low\_A + 1)$) $T(\frac{n}{2})$
12:     **else**
13:         **return** FIND_MEDIAN($low\_A, mid\_B + 1, mid\_A - low\_A + 1$) $T(\frac{n}{2})$
14:     **end if**
15: **end function**
16:
17: **function** QUERY($database, k$)
18:     **return** the kth smallest value of $database$
19: **end function**

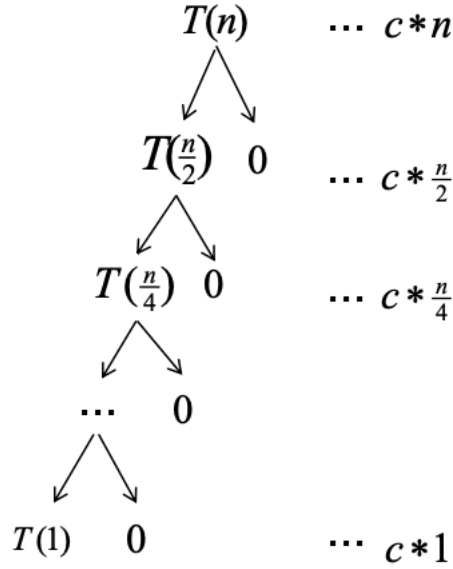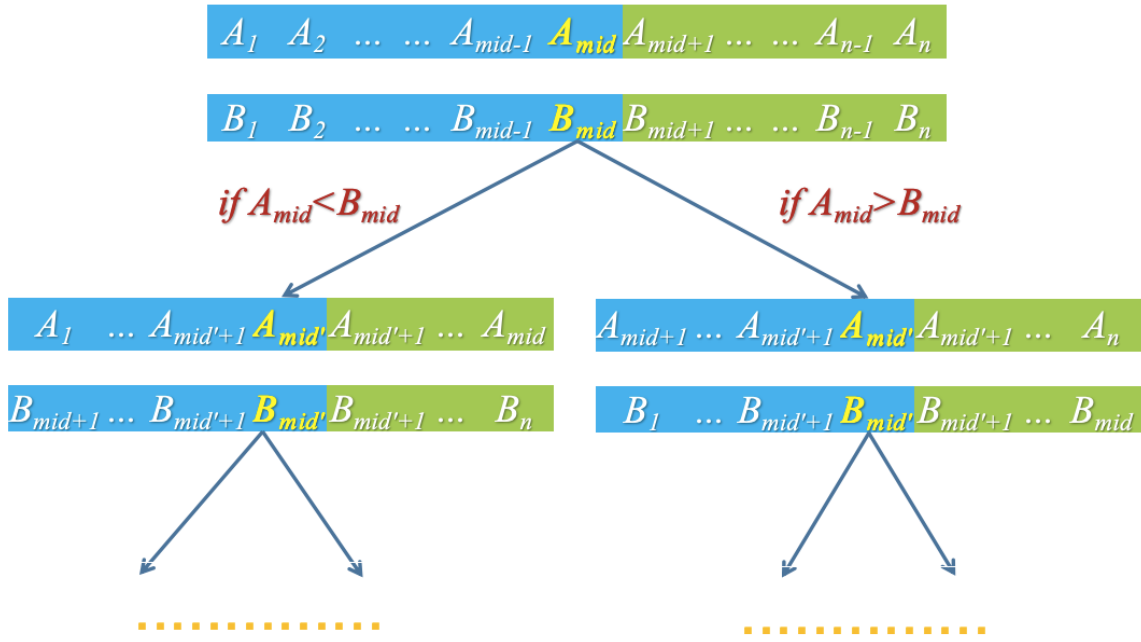## 1.4   Subproblem Reduction Graph

Figure 1: Find median from two seprate databases

1.5   Correctness

In this algorithm, we define median differently between A and B. If the size n is an even number, we define the median of A is $A_{\frac{n}{2}}$ and the median of B is $B_{\frac{n}{2}}$. If the size n is an odd number, we define the median of A as $A_{\lceil \frac{n}{2} \rceil}$, while the median of A is $A_{\lceil \frac{n}{2} \rceil - 1}$. So that we can ensure the total numbers of left half of the two sequence(include each median) is n.

There are three cases of comparision between the median of A and the median of B:

- $\boldsymbol{A_{mid} == B_{mid}}$: Due to every element is distinct, so this case is impossiable.

- $\boldsymbol{A_{mid} > B_{mid}}$: Due to the total number of the left half of A and B is n, if we sort these n numbers, B_{mid} must be in front of A_{mid}, so there is no possible that B_{mid} is the

n^{th} number—the median of $2 * n$ values. And because there are about $(\frac{n}{2} - 1)$ numbers smaller than B_{mid} in sequence B, at least $\frac{n}{2}$ numbers in sequence B that can't be the median value.

It is easy to reach below conclusion:

$$A_k > A_{mid} > B_{mid} \quad k \in [mid + 1, n]$$
$$A_k > A_{mid} > A_{mid-1} > A_{mid-2} > \cdots > A_2 > A_1$$
$$A_k > B_{mid} > B_{mid-1} > B_{mid-2} > \cdots > B_2 > B_1$$

So there are at least n numbers are smaller than $A_k(k \in [mid+1, n])$. Due to we are looking for n^{th} value, so there are about $\frac{n}{2}$ numbers $(A_k(k \in [mid + 1, n]))$ in sequence A can't be the median value.

In summary, we can reach a conclusion that at least n numbers can't be the median value, and we only need to searching the median recursely in the left part of the sequence A and the right part of the sequence B.

- **$A_{mid} < B_{mid}$** : Similar to the above, we can reach a conclusion that at least n numbers can't be the median value, and we only need to searching the median recursely in the right part of the sequence A and the left part of the sequence B.

In summary, the maximum query times occurs when there are only one element left in sequence A and B. As each recursion, the number of sequences is reduced by two times, so the maximum query times is $2 * log_2 n$, And that's $O(log_2 n)$.

1.6   Complexity Analysis

Firstly, we define the time complexity of query as T_{query}.
The search process lasted until the end that the sequence only contain one element.

$$
\begin{aligned}
T(n) &= T(\frac{n}{2}) + 2 * T_{query} + c \\
&= 2 * log_2 n * T_{query} + c * log_2 n \\
&= O(log_2 n) * T_{query} + O(log_2 n)
\end{aligned}
$$

2 PROBLEM TWO

2.1 Description

Given a binary tree, suppose that the distance between two adjacent nodes is 1, please give a solution to find the maximum distance of any two node in the binary tree.

2.2 Basic Idea

DIVIDE: due to the tree is a binary tree, we can divide it into two subtree.
CONQUER: find the maximum distance of any two node in each subtree.
COMBINE: the maximum distance is occur in left tree, or right tree, or a line through the root node. In the third condition, It suffices to consider the maximum distance through the root node consisting of one node from left tree and one node from the right tree and each node's depth must be deepest in its subtree.

2.3 Pseudo-Code

PROBLEM 2 Find the maximum distance of any two node in the binary tree
---
INPUT: The binary tree's root node, $root$.
OUTPUT: The maximum distance of any two node in the binary tree.
 1: **function** MAX_DISTANCE($root$)
 2:     /* ($MDis, MDep$) represent the maximum distance and depth in the binary tree that its root node is $root$. */
 3:     **if** $root == NULL$ **then**
 4:        **return** $(0, 0)$
 5:     **else if** $root \rightarrow left == NULL$ **and** $root \rightarrow right == NULL$ **then**
 6:        **return** $(0, 1)$
 7:     **end if**
 8:     $(L\_MDis, L\_MDep) \leftarrow$ MAX_DISTANCE($root \rightarrow left$)
 9:     $(R\_MDis, R\_MDep) \leftarrow$ MAX_DISTANCE($root \rightarrow right$)
10:     $LR\_MDis \leftarrow L\_Dep + R\_Dep$
11:     $MDis \leftarrow$ MAX(L_MDis, R_MDis, LR_MDis)
12:     $MDep \leftarrow$ MAX(L_MDep, R_MDep) $+1$
13:     **return** $(MDis, MDep)$
14: **end function**

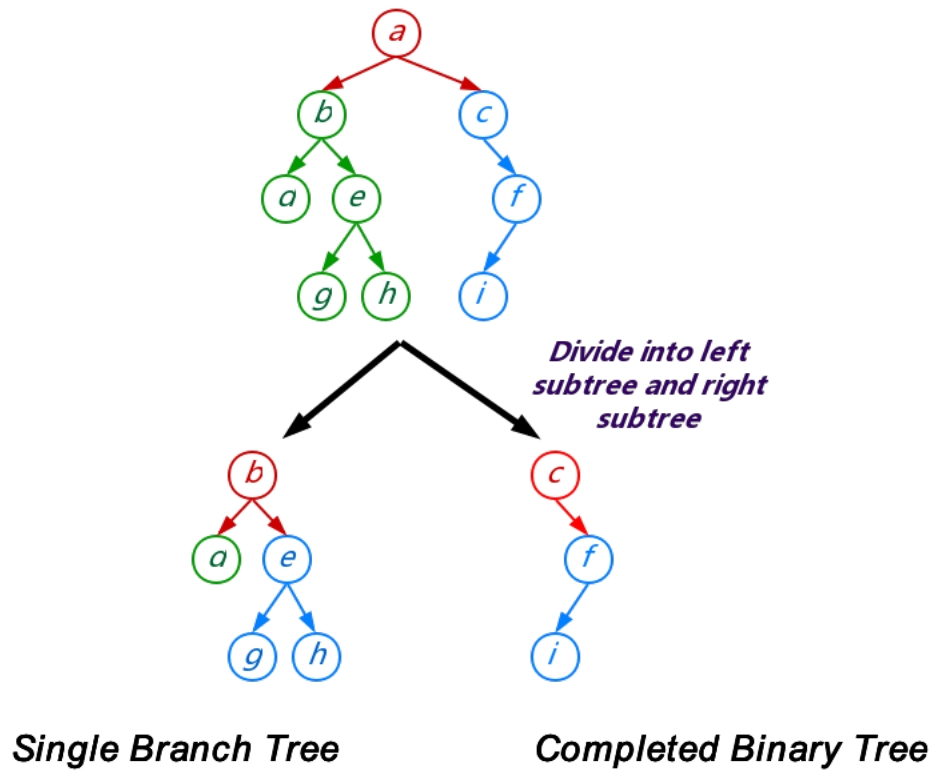## 2.4 Subproblem Reduction Graph

Figure 2: Find Maximum Distance

2.5 Correctness

The location of the largest distance line must be one of the following three situations.

- Completely located in the left subtree.
- Completely located in the right subtree.

- Through the root node.

In our algorithm, we use the maximum depth of the subtree to calculate the result of the third cases. Next, we will prove the correctness of the third cases in this algorithm. First, we define the two nodes as (u, v).

- If the tree doesn't has one node, there is no doubt that the maximum distance is zero. If the tree has only one node—root, the maximum distance is zero obviously. In our algorithm, due to both left and right subtree's depth are zero, we can calculate the result is zero according to the formula in the tenth line ($LR\_MDis \leftarrow L\_Dep + R\_Dep$).

- If the tree has many nodes, we need to calculate the depth of two subtrees. But why does the maximum distance contain the largest depth of nodes? Suppose the right is NULL and its depth is zero obviously, so one of the two nodes that constitute the maximum distance must be root node and the other node is locate in the left subtree, we regard the root node as node u. Give two node N_i, N_j, and $Dep_{N_i} < Dep_{N_j}$, the two nodes are locate in left subtree. Suppose N_i is node v, so the distance (u, v) is Dep_{N_i}. But there is another node N_j that makes the distance larger. Thus the deeper the depth of the node v, the greater the distance (u, v). In other words, Only the nodes with the deepest depth can form the maximum distance. This proof applies to another case that both left and right subtrees are not NULL.

2.6 Complexity Analysis

First of all, we consider two extreme situations:

- **Single Branch Tree:** The binary tree degenerate into single branch tree, the maximum depth of the tree is n, $T(n) = T(n-1) + O(1)$, it cost $O(n)$ time.

- **Comleted Binary Tree:** The binary tree is a completed tree, the maximum depth of the tree is $log(n)$.

$$
\begin{aligned}
T(n) &= 2 * T(\frac{n}{2}) + c \\
&= c * \frac{1 - 2^{log_2 n}}{1 - 2} \\
&= cn - c \\
&= O(n)
\end{aligned}
$$

Obviously, the time comlexity of this situation is same as single branch tree—$O(n)$.

- **Most Cases:** Instead of encounter a completed binary tree or a single branch tree, we may encounter a tree that similar to the above situations. We claim that the expected running time is still $T(n) = O(n)$.

3 PROBLEM THREE

3.1 Description

Suppose now that you're given an $n * n$ grid graph G. (An $n * n$ grid graph is just the adjacency graph of an$n * n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers(i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$.)

We use some of the terminology of problem 3. Again, each node v is labeled by a real number x_v ; you may assume that all these labels are distinct. Show how to find a local minimum of G using only $O(n)$ probes to the nodes of G. (Note that G has n^2 nodes.)

3.2 Basic Idea

DIVIDE: Divide the graph into four quadrants.

CONQUER: Find global minimum value and its coordinate in center row/column. If it's a nadir, return the value.

COMBINE: The subprogram's result is one of this program's results, it also is this problem's result.

In the beginning, we fill some values(INT_MAX) around the grid graph G. Set the minimum value v is $INT_M AX$ and it's coordinate V is $(0, 0)$.

At each recursion, we keep track on the current minimum (v, V). Then we calculate the global miminum of center row and center column of the new graph G' and compare it to v. If the letter is bigger, update current minimum. Then check whether the current minimum is actually a local minimum. And if so, return it; Otherwise, comparing v with it's neighbors which is not included in center row/column or boundary to determine which quadrant it should enter. Then recurse as usual.

If the quadrant's row-size or column-size less than 4, we calculate directly the global minimum of the quadrant and return it. Otherwise, reuse the above approach.

3.3 Pseudo-Code

PROBLEM 3 find a local minimum of $G(n * n$ grid graph $)$

INPUT: An mxn grid graph, G; The graph size,m, n; The $left - top$ corner's coordinate of G,S; The global minimum value and its coodinate of center row and column,min, M

OUTPUT: A local minimum value and its coordinate, (v, V)

1: **function** FIND_LOCALMINIMUM(G, m, n, S, v, V)
2:    **if** $m <= 3$ **or** $n <= 3$ **then**
3:        find global min value v and its coordinate V within the $n * n$ window
4:        **return** (v, V)
5:    **end if**
6:    /* Find global minimum within the center row and column */
7:    **for** $i = 0 \to n - 1$ **do**
8:        **if** $v > G[S_x + (m - 1)/2][S_y + i]$ **then** **Center Row**
9:            $v \leftarrow G[S_x + (m - 1)/2][S_y + i]; V \leftarrow (S_x + (m - 1)/2, S_y + i)$
10:        **end if**
11:    **end for**
12:    **for** $i = 0 \to m - 1$ **do**
13:        **if** $v > G[S_x + i][S_y + (n - 1)/2]$ **then** **Center Column**

14:          $v \leftarrow G[S_x + i][S_y + (n-1)/2]; V \leftarrow (S_x + i, S_y + (n-1)/2)$

15:      **end if**

16:     **end for**

17:     **if** Coordinate V is lower than all its neighbor **then**

18:      **return** (v, V)

19:     **end if**

20:     /* Recurse in quadrant */

21:     **if** $(V_x == S_x$ **and** $V_y < S_y + (n-1)/2)$ **Or** $(V_x == S_x + (n-1)/2$ **and** $V_y < S_y + (n-1)/2$ **and** $G[V_x - 1][V_y] < G[V_x + 1][V_y])$ **Or** $(V_y == S_y$ **and** $V_x < S_x + (m-1)/2)$ **Or** $(V_y == S_y + (n-1)/2$ **and** $V_x < S_x + (m-1)/2$ **and** $G[V_x][V_y - 1] < G[V_x][V_y + 1])$ **then**

22:      **return** Find_LocalMinimum$(G, m/2, n/2, S, v, V)$ **Left-Top Quadrant**

23:     **else if** $(V_x == S_x + (m-1)/2$ **and** $V_y < S_y + (n-1)/2$ **and** $G[V_x - 1][V_y] > G[V_x + 1][V_y])$ **Or** $(V_x == S_x + m - 1$ **and** $V_y < S_y + (n-1)/2)$ **Or** $(V_y == S_y$ **and** $V_x > S_x + (m-1)/2)$ **Or** $(V_y == S_y + (n-1)/2$ **and** $V_x > S_x + (m-1)/2$ **and** $G[V_x][V_y - 1] < G[V_x][V_y + 1])$ **then**

24:      **return** Find_LocalMinimum$(G, m - m/2, n/2, (S_x + (m-1)/2, S_y), v, V)$ **Left-Bottom Quadrant**

25:     **else if** $(V_x == S_x$ **and** $V_y > S_y + (n-1)/2)$ **Or** $(V_x == S_x + (m-1)/2$ **and** $V_y > S_y + (n-1)/2$ **and** $G[V_x - 1][V_y] < G[V_x + 1][V_y])$ **Or** $(V_y == S_y + (n-1)/2$ **and** $V_x < S_x + (m-1)/2$ **and** $G[V_x][V_y - 1] > G[V_x][V_y + 1])$ **Or** $(V_y == S_y + n - 1$ **and** $V_x < S_x + (m-1)/2)$ **then**

26:      **return** Find_LocalMinimum$(G, m/2, n - n/2, (S_x, S_y + (n-1)/2), v, V)$ **Right-Top Quadrant**

27:     **else if** $(V_x == S_x + (m-1)/2$ **and** $V_y > S_y + (n-1)/2$ **and** $G[V_x - 1][V_y] > G[V_x + 1][V_y])$ **Or** $(V_x == S_x + m - 1$ **and** $V_y > S_y + (n-1)/2)$ **Or** $(V_y == S_y + (n-1)/2$ **and** $V_x > S_x + (m-1)/2$ **and** $G[V_x][V_y - 1] > G[V_x][V_y + 1])$ **Or** $(V_y == S_y + n - 1$ **and** $V_x > S_x + (m-1)/2)$ **then**

28:      **return** Find_LocalMinimum$(G, m - m/2, n - n/2, (S_x + (m-1)/2, S_y + (n-1)/2), v, V)$ **Right-Bottom Quadrant**

29:     **end if**
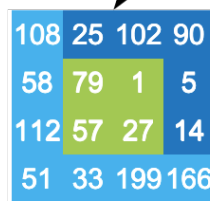
30: **end function**

3.4 Subproblem Reduction Graph
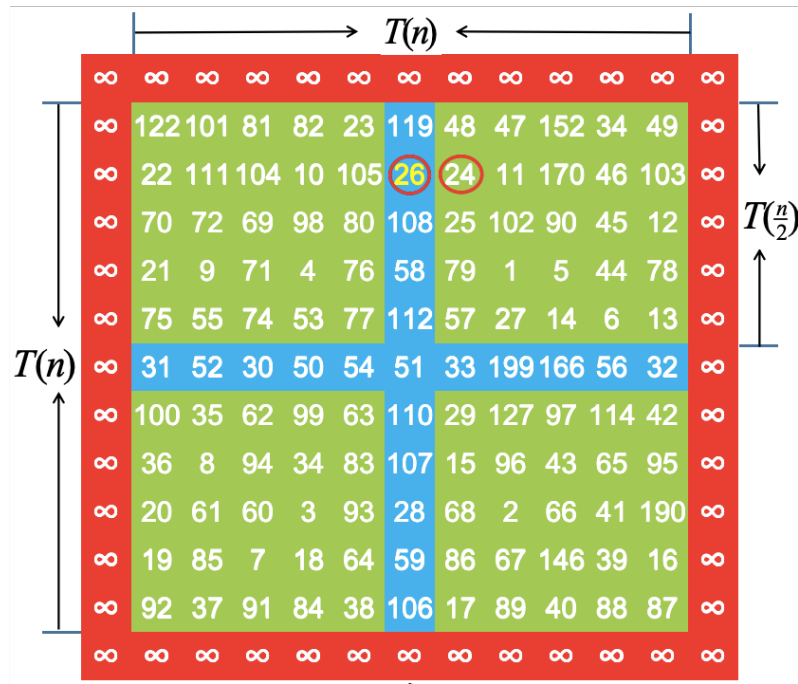
The value 26 is the smallest in center row/column and smaller than +oo; and the smllest value of it's neighbors is 24.

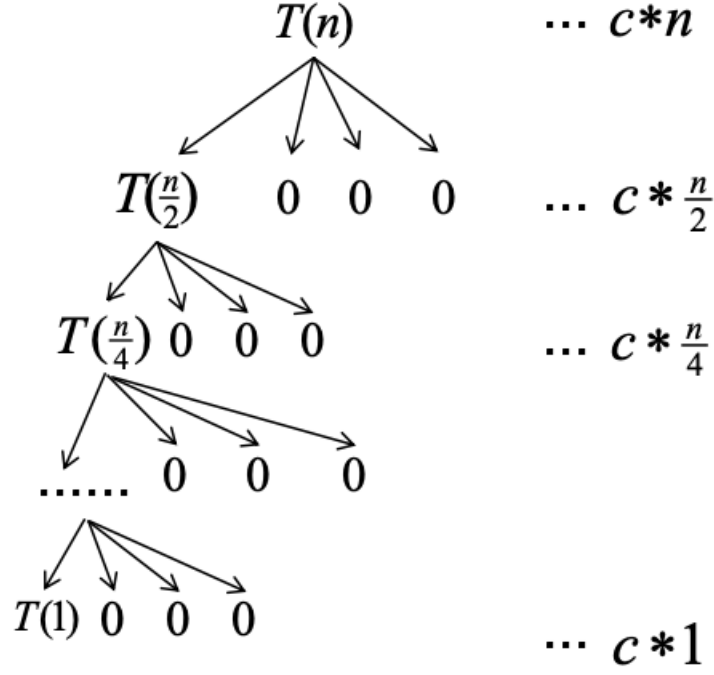The value 5 is the smallest in center row/column and smaller than 26; the smllest value of it's neighbors is 1.

Figure 3: Find Local Minumum

3.5 Correctness

- Because we only update the current minimum when we find a smaller value, the current minimum would never increase as we descend in recursion.

- If the program enter a quadrant, the quadrant must include a local minimum.

    Becase we calculate a current minimum of the grid graph G, and the coordinate of current minimum must located in the boundary of this quadrant. In additional, We only enter this quadrant when the current minimum's neighbor which is located in this quadrant less than current minimum, so there are at least one element of this quadrant is less than it's boundary. Thus the quadrant must include a local minimum.

- The local minimum of each quadrant is also the local minimum of overall graph.

3.6 Complexity Analysis

Reduce $n * n$ grid graph into $\frac{n}{2} * \frac{n}{2}$ quadrant in $O(n)$ time. In the lines 7 14, it cost $2 * n$ time.

$$
\begin{aligned}
T(n) &= T(\frac{n}{2}) + 2n \\
&= T(\frac{n}{2}) + cn \\
&= T(1) + c(n + \frac{n}{2} + \frac{n}{4} + ... + 2 + 1) \\
&= 2cn \\
&= O(n)
\end{aligned}
$$