Tutorial - 1

Ans1 Asymptotic notation describe the algorithms efficiency and performance in a meaningful way. It describe the behaviour of time or space complexity for large instance characteristics. They are Mathematical tool to represent the time complexity of algorithm for asymptotic analysis. There are mainly three asymptotic notations:-

1) **Big-O-Notation** :-

The Big O notation defines an upper bound of algorithm, it bounds a function only from above, for eg: insertion sort. It takes linear time in best case & quadratic time in worst case is $O(n^2)$. So we can say that TC of insertion sort is $O(n^2)$.

$$f(n) = Og(n)$$

$g(n)$ is tight upperbound of $f(n)$

$$f(n) = O(g(n))$$

if $f(n) < c \cdot g(n)$

$\forall\ n \geq n_0$, some constant $c > 0$

2) **Omega Notation (Ω-notation)**

Omega notation represent the lower bound of the running time of an algorithim. It can be useful when we have lower bound on time complexity of an algorithm.

Eg: The time complexity of insertion sort can be written as $\Omega(n)$, but is not a very useful information about insertion $\Omega(n)$, but is not a very useful information about insertion.

3) Theta Notation (Θ notation)

The theta notation bounds a function from above and below so it define exact asymptotic behaviour.

Ex: $3n^3 + 6n^2 + 600 = \Theta(n^3)$

Ans2 $O(\log n)$

Ans3 $T(n) = 3T(n-1)$ if $n>0$ otherwise 1

$= 3(3T(n-2))$

$= 3^2\, T(n-2)$

$= 3^3\, T(n-3)$

$= 3^n\, T(n-n)$

$T(n) = 3^n$

Ans4 $T(n) = 2T(n-1) - 1$ if $n>0$ otherwise 1

$= 2(2T(n-2) - 1) - 1$

$= 2^3(T(n-2)) - 2 - 1$

$= 2^3(T(n-3)) - 2^3 - 2^1 - 2^0$

$= 2^n(T(n-n)) - 2^{n-1} - 2^{n-2} \ldots - 2^0$

$T(0) = 1$

$= 2^n - (2^n - 1)$

$TC = O(2^n)$

Ans5 $TC = O(\sqrt{n})$

Ans6 $TC = O(\sqrt{n})$

Ans7 $TC = n/2 \times \log n \times \log n$

$= O(n \log^2 n)$

Ans8 $TC = O(n^3)$

Ans9 $i = j$

$1 = n$ times

2 $n/2$ times TC $O(n\log n)$

3 $n/3$ times

n n/n times

Ans10 Since polynomical grow smaller than exponential n^k has an asymptotic upper bound of $O(a^n)$

for $a = 2$, $n_0 = 2$

Tutorial - 2

Ans 1

i	j
1	2
3	3
6	4
10	5

$K^2 = n$

$K = \sqrt{n}$

$TC = O(\sqrt{n})$

$$\frac{K(K+1)}{2} = n$$

Ans 2 $T(n) = T(n-1) + T(n-2)$

$T(0) = 0 \, , \, T(1) = 1$

Let $T(n-1) \approx T(n-2)$

Using backword solution

$T(n) = 2 \times 2 (T(n-2) + 1) + 1$

$= 4(T(n-2) + 3)$

$T(n-2) = 2T(n-3) + 1$

$T(n) = 2(2(2(T(n-3] + 1) + 1) + 1)$

$= 8T(n-3) + 3$

$T(n) = 2^K T(n-K) + 2^K - 1$

$= n = K$

$T(n) = 2^n + 2^n + 1$

$TC = O(2^n)$

Ans 3 $O(n \log n)$

```
void func(int n){
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j=j*2){
            task;
        }}
    }
}
```

O(n^3)

```
void func( int n){
    for (int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            for(int k=0; k<n; k++){
                Task;
            }
        }
    }
}
```

O(log(log n))

```
void func (int n){
    for (int i=n; i>1; i=pow(i,k)){
        Task;
    }
}
```

Ans 4 $T(n) = T(n/4) + T(n/2) + cn^2$

assume $T(n/2) > T(n/4)$

$T(n) = 2T(n/2) + cn^2$

$c = \log_b a$

$= \log_2 2 = 1$

$n^c < f(n)$

$TC = O(n^2)$

Ans 5

i	j
1	n times
2	n/2 times
3	n/3 times
n/n	n/n times
	log n

TC = O(n log n)

Ans6

$i = 2, 2^k, 2^{k^2}, 2^{k^3} \ldots 2^{k \log k (\log n)}$

$2^{k \log k (\log n)} = n$

$2^{\log(1)} = 1$

$TC = O(\log(\log n))$

Ans7

$T(n) = T(9n/10) + T(n/10) + O(n)$

taking one branch 99% and other 1%.

$T(n) = T(99n/100) + T(n/100) + O(n)$

1^{st} level $= n$

2^{nd} level $= 99n/100 + n/100 = n$

So 3^{rd} remain same for any kind of position. If we take longer branch $= O(n \log_{\frac{100}{99}} n)$. For shorter branch $= \Omega(n \log_{10} n)$

either way base complexity of $O(n \log n)$ remain.

Ans8

(a) $100 < \sqrt{n} < \log(\log n) < \log n < n < n\log n < \log n! < n^2 < n! < 2^n < 2^{2^n}$

b) $1 < \log(\log n) < \sqrt{\log n} < \log n < \log^2 n < n < n\log n < 2n < 2^{2^n} < n!$

(c) $96 < \log_2 n < \log n! < n\log_2 n < n\log_6 n < 5n < n! < 8n^2 < 7n^3 < 8n^{2n}$

Tutorial- 3

Ans 1 Pseudocode for linear search is

```
int linear (int *arr, int n, int key){
    for (int i=0; i<n; i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}
```

Ans 2 Pseudocode of insertion sort

```
void insertion(int arr[], int n){
  for (int i=1; i<n, i++){
    int key=arr[i];
    int j=i-1;
    while(j>=0&& arr[j]>key){
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = key;
  }
}
```

Ans 3 Average case complexity of sorting algos

- Bubble sort → $O(n^2)$
- insertion sort → $O(n^2)$
- Selection Sort → $O(n^2)$
- merge sort → $O(n\log n)$
- Quick sort → $O(n\log n)$
- Heap sort → $O(n\log n)$

Ans 4

Stable : (appears in same order)

Bubble, insertion, merge

Inplace : (using constant space)

Bubble, selection, insertion, heap

Ans 5 **Pseducode for Binary Search**

```
int BS(int arr[], int n, int Key) {
    int low=0, high =n-1;
    while (low <= high) {
        int mid = low+high / 2;

        if (key == arr[mid]){
            return mid;
        }
        else if (key > arr[mid]){
            start = mid+1;
        }
        else{
            end = mid-1;
        }
    }
    return -1;
}
```

TC → O(log n)

SC → O(1)

Ans 6 Recurrence relation of Binary search is

$$T(n) = T(n/2) + 1$$

Ans 7 Assuming given array is sorted then we can find sum in linear complexity.

```
int sum(int arr[], int target){
    int i=0; j=n-1;
    while(i<j){
        if(arr[i]+arr[j] == target){
            return 1;
        }
        else if(arr[i]+arr[j] > target){
            j--;
        }
        else{
            i++;
        }
    }
    return -1;
}
```

Ans 8 Quicksort is the best sorting algorithm in Practical use as it follow the locality of reference and also its best case time complexity is O(n log n)

Ans 9 No of inversion tells us how for the array is from being sorted.

if (arr[i] > arr[j] && i < j)

→ 4, 7, 21, 31, 8, 10, 1, 20, 6, 45

inversion = 4+7+7+4+4+3+2

= 31

Ans 10 Quick Sort

* Best Case → When array is totally usorted.
* Worst case → When array is sorted or reverse sorted

Ans 11	Merge Sort	Quick Sort
Best	$2T(n/2) + \Theta(n)$	$T(n) = T(K) + T(n-K-1) + \Theta(n)$
Worst	$2T(n/2) + \Theta(n)$	$T(n) - T(n-1) + \Theta(n)$

Similarity → Both are based on divide and conquer technique.

Diffrences → Worst case TC of merge sort is $O(n\log n)$ while of Quick sort in $O(n^2)$.

Ans 12

```
void stableSelectionSort(int arr[], int n){
    for (int i=0; i<n-1; i++){
        int min = i;
        For( int j=i+1; j<n; j++){
            if(arr[min] > arr[j]){
                min = j;
            }
        }
    }
    int key = arr[min];
    while(min > i){
        arr[min] = arr[min -1];
        min--;
    }
    arr[i] = key;
  }
}
```

Ans 13 Optimised Bubblo Sort

```
for (int i=0; i<n; i++){
    bool swap = false;
    for (j=0; j<n-i-1; j++){
        if (arr[j] < arr[j+1]){
            swap (arr[j], arr[j+1]);
            swap = true;
        }
    }
    if (!swap){
        break;
    }
}
```

Ans 14 In such case, merge sort would be efficient as it is an exernal sorting algorithm i.e. data is divided into chunks and then sorted using merge sort.

→ Sorted data is dumped into files

- Internal sorting: It is a type of sorting technique in which whole sorting takes place in main memory of computer.

Tutorial - 4

Ans1 $T(n) = 3T(n/2) + n^2$

$T(n) = aT(n/b) + f(n)$, $a = 3; b = 2$

$c = \log_2 3 = 1.58$

$n^c = n^{1.58} \simeq n^2$

$f(n) = n^2$

By case 3 $f(n) > n^c$

$T(n) = \Theta(f(n)) = \Theta(n^2)$

Ans2 $T(n) = 4T(n/2) + n^2$

$a = 4$, $b = 2$, $f(n) = n^2$

$n^c = n^{\log_2^4} = n^2$

By case 2 $\rightarrow$ $f(n) = n^c$

$T(n) = \Theta(n^c \log n) = \Theta(n^2 \log n)$

Ans3 $T(n) = T(n/2) + 2^n$

$a = 1$, $b = 2$, $f(n) = 2^n$

$n^c = n^{\log_2^1} = 1$

By case $f(n) > n^c$

$T(n) = \Theta(f(n)) = \Theta(2^n)$

Ans4 $T(n) = 2^n T(n/2) + n^n$

$a = 2^n$, $b = 2$, $f(n) = n^n$

$n^c = n^{n\log_2^2} = n^n$

By case $f(n) = n^c$

$T(n) = \Theta(n^c \log n)$

$T(n) = \Theta(n^n \log n)$

Ans5 $T(n) = 16T(n/4) + n$

$a = 16, b = 4, f(n) = n$

$n^c = n^{\log_2^4} = n^2$

$n^c > f(n$

$T(n) = \Theta(n^c)$

$T(n) = \Theta(n^2)$

Ans6 $T(n) = 2T(n/2) + n\log n$

$a = 2, b = 2, f(n) = n\log n$

$n^c = n^{\log_2^2} = n$

By case $f(n) > n^c$

$T(n) = \Theta(f(n))$

$T(n) = \Theta(n\log n)$

Ans7 $T(n) = 2T(n/2) + n/\log n$

$a = 2, b = 2, f(n) = n/\log n$

$n^c = n^{\log_2^2} = n$

By case $n^c > f(n)$

$T(n) = \Theta(n)$

Ans8 $T(n) = 2T(n/4) + n^{0.51}$

$a = 2, b = 4, f(n) = n^{0.51}$

$n^c = n^{\log_4^2} = n^{0.5}$

By case $f(n) > n^c$

$T(n) = \Theta(n^{0.51})$

Ans 9 $T(n) = 0.5T(n/2) + 1/n$

$a = 0.5$, $b = 2$, $f(n) = 1/n$

$n^c = n^{\log_2 0.5} = n^{-1} = 1/n$

By case $f(n) = n^c$

$T(n) = \Theta(1/n \log n)$

Ans 10 $T(n) = 16T(n/4) + n!$

$a = 16$, $b = 4$, $f(n) = n!$

$n^c = n^{\log_4 4^2} = n^2$

By case $n^c < f(n)$

$T(n) = \Theta(n!)$

Ans 11 $T(n) = 4T(n/2) + \log n$

$a = 4$, $b = 2$, $f(n) = \log n$

$n^c = n^{\log_2 2^2} = n^2$

By case $n^c > f(n)$

$T(n) = \Theta(n^2)$

Ans 12 $T(n) = \sqrt{n}\, T(n/2) + \log n$

$a = n^{1/2}$, $b = 2$, $f(n) = \log n$

$n^c = n^{\log_2 n^{1/2}} = n^{1/2 \log}$

By case $f(n) < n^c$

$T(n) = \Theta(n^{1/2 \log n})$

Ans 13 $T(n) = 3T(n/2) + n$

$a = 3$, $b = 2$, $f(n) = n$

$n^c = n^{\log_2 3}$

By case $f(n) < n^c$

$T(n) = \Theta(n^{\log_2 3})$

Ans14 $T(n) = 3T(n/3) + \sqrt{n}$

$a=3, b=3, f(n)=\sqrt{n}$

$n^c = n^{\log_3 3} = n$

By case $n^c > f(n)$

$T(n) = \Theta(n)$

Ans15 $T(n) = 4T(n/2) + cn$

$a=4, b=2, f(n)=cn$

$n^c = n^2$

By case $n^c > f(n)$

$T(n) = \Theta(n^2)$

Ans16 $T(n) = 3T(n/4) + n\log n$

$a=3, b=4, f(n)=n\log n$

$n^c = n^{\log_4 3}$

By case $f(n) > n^c$

$T(n) = \Theta(n\log n)$

Ans17 $T(n) = 3T(n/3) + n/2$

$a=3, b=3, f(n)=n/2$

$n^c = n$

By case $n^c > f(n)$

$T(n) = \Theta(n)$

Ans-18 $T(n) = 6T(n/3) + n^2\log n$

$a=6, b=3, f(n)=n^2\log n$

$n^c = n^{\log_3 6} = n^{1.63}$

By case $f(n) > n^c$

$T(n) = \Theta(f(n))$

$T(n) = \Theta(n^2\log n)$

Ans-19 $T(n) = 4T(n/2) + n/\log n$

$a = 4,\ b = 2,\ f(n) = n/\log n$

$n^c = n^{2\log_2^2} = n^2$

By case $n^c > f(n)$

$T(n) = \Theta(n^2)$

Ans20 $T(n) = 64T(n/8) - n^2\log n$

$a = 64,\ b = 8,\ f(n) = -n^2\log n = n^2\log 1/n$

$n^c = n^2$

By case $n^c < f(n)$

$T(n) = \Theta(n^2\log 1/n)$

Ans 21 $T(n) = 7T(n/3) + n^2$

$a = 7,\ b = 3,\ f(n) = n^2$

$n^c = n^{\log_3^7}$

By case $n^c < f(n)$

$T(n) = \Theta(n^2)$

Ans22 $T(n) = T(n/2) + n(2 - \cos n)$

$a = 1,\ b = 2,\ f(n) = n(2 - \cos n)$

$n^c = n^0 = 1$

By case $f(n) > n^c$

$T(n) = \Theta(n(2 - \cos n))$

Tutorial - 5

Ans1 BFS → It stands for Breadth First Search. It uses queue to find the shortest path. It is better when target is closer to source. It consider all neighbour, so it is not suitable for decision tree used for puzzel game. It is slower than DFS. TC → O(V+E).

DFS → It stands for Depth First Search. It uses stack to find the shortest path. It is better when target is for from source. It is more suitable as with one direction, we need to traverse for further to argument the decision. It is faster than BFS.
TC → O(V+E)

Ans2 Stack is used to implement DFS, because it is we first traverse the whole branch of the tree and later on visit the adjacent branch, since this is similar to LIFO, therefore stack is used.
Queue is used to implement BFS, it is because queue uses FIFO instead because BFS is to test the immediate children first and after all immediate children are tested, to their children & so forth.

Ans3 Sparse Graph : Graph where number of edge is much less than the possible number of edges.

Dense Graph : Graph where number of edges is much more than close to maximal number of edges.

- If Graph is dense it should be represented by adjiancey matrix
- If Graph is sparse it should be represented by adjianley list.

Ans4 BFS : In undirected graph, do a BFS traversal on given graph, for each visited vertex V, if the there is any adjacent 'a' such that 'V' is already visited & 'U' is not parent of 'V' then there is cycle in a graph.

DFS :

Run DFS prove that from a node and mark their node as visited now for any other vertices if its neighbour is already visited & that neighbour is not the parent then their ~~children~~ exist a cycle in graph.

Ans5 : Disjoint Set Data structure

The disjoint set can be defined as the sub set where there is no common element between set operation are 1 union, 2 make new set, 3 Find.

Ans6 BFS

A → B → C → D → E

G → H → F

DFS

A → D → C → B

G → F → H

Ans7 Connected component → 4, vertices → 10

Ans8 Topological sort → 0-1-2-3-4-5 , DFS : 5→2→1→3→0

Ans9 Yes Heap DFS can be used to make Priority Queue

1) Dijkastra to find shortest path
2) Prim's Algo
3) Hoffman Algo

Ans10 Min Heap → Root element is smallest

Max Heap → Root element is largest.

Tutorial - 6

Ans1 Minimum Spanning Tree

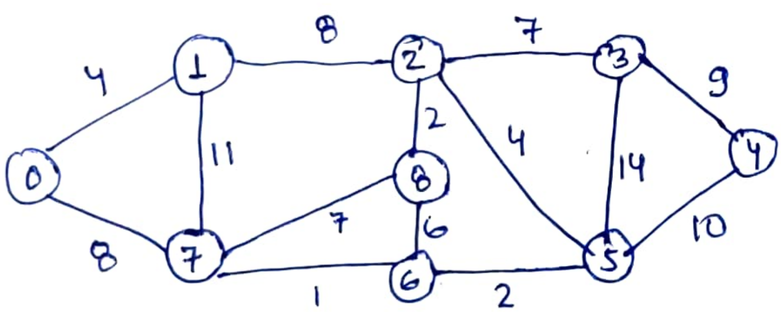
A spanning Tree of an undirected graph is a subgraph that is a tree and joined by all vertices. One of those tree which has minimum total cost would be its minimum spanning tree.

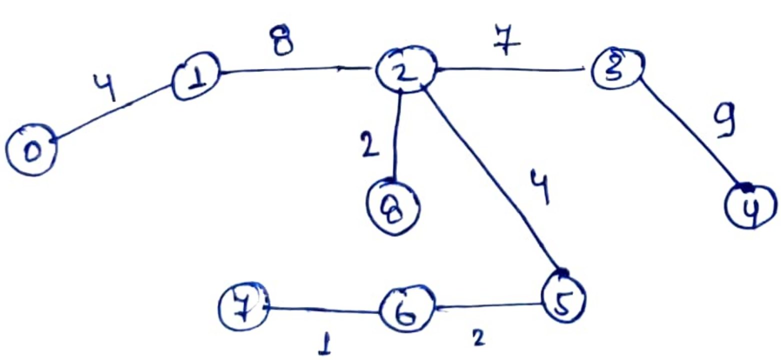
- It has direct application in the design of network including to computer network, telecommunication network, transportation network etc.

Ans2

	Prim's Algo	Kruskal's Algo	Dijkastra's Algo	Bellman Ford
TC	$O(V^2)$	$O(E \log V)$	$O(V + E\log V)$	$O(VE)$
SC	$O(V+E)$	$O(\lvert E\rvert + \lvert V\rvert)$	$O(V^2)$	$O(V^2)$

Ans3

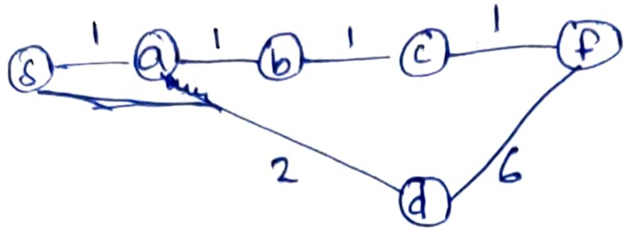




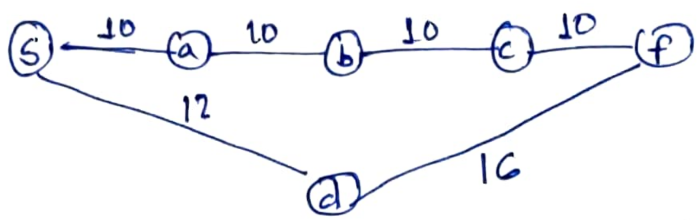
Min width → 37

Ans 4 (i) If 10 unit is added to each edge, the overall weight of the path may change.

Eg



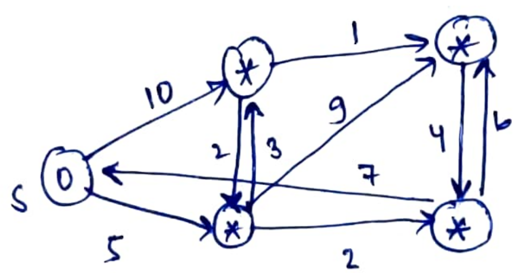
Shortest path s → a → b → c → d with min weight = 4



Shortest Path s → d → f with min weight = 28

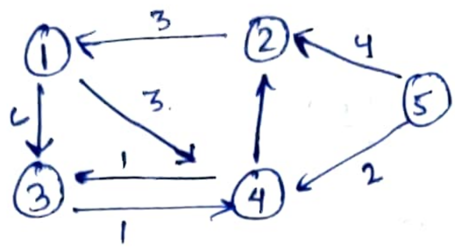
(ii) Multiplying the weight of each edge by 10 unit will have no impact on shortest path.

Ans 5



S	U	V	X	Y
0	∞	∞	∞	∞
0	10	∞	5	∞
0	10	11	5	∞
0	10	11	5	7

Ans 6 All part shortest path algorithm – Floyd Warshall



$$A_0 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 0 & \infty & \infty \\ 3 & \infty & \infty & \infty & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & 0 \end{array}$$

$$A_1 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & 0 \end{array}$$

$A^0 = [2,3] = \infty$

$A^0[2,1] + A^0[1,3] = 3+6 = 9$, $9 < \infty$

$A^0[2,4] = \infty$

$A^0[2,1] + A^0[1,4] = 3+3 = 6$, $6 < \infty$

$A^0[2,5] = \infty$

$A^0[2,1] + A^0[1,5] = 3+\infty$

$$A_2 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & 7 & 4 & 13 & 2 & 0 \end{array}$$

$A[1,3] = 6$

$A_1[1,2] + A_1[2,3] = \infty + 9, \quad 6 < \infty + 9$

$$A3 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & 7 & 4 & 13 & 2 & 0 \end{array}$$

$$A4 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 4 & 4 & 3 & \infty \\ 2 & 3 & 0 & 7 & 6 & \infty \\ 3 & \infty & 3 & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & 7 & 3 & 3 & 2 & 0 \end{array}$$

$$A5 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 4 & 4 & 3 & \infty \\ 2 & 3 & 0 & 7 & 6 & \infty \\ 3 & \infty & 3 & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & 7 & 3 & 3 & 2 & 0 \end{array}$$