**ALGORITHMICS UNIT 3 & 4   2022**

# Trial Exam 2: 2022 Solutions

SECTION A -  Multiple Choice Solutions

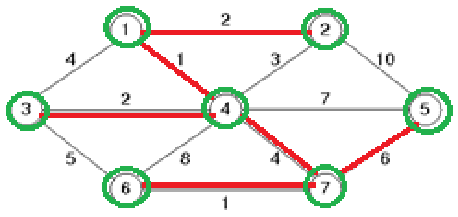
| | | | |
|---|---|---|---|
| Question 1  B | Question 6  C | Question 11  D | Question 16  A |
| Question 2  B | Question 7  D | Question 12  D | Question 17  D |
| Question 3  A | Question 8  A | Question 13  A | Question 18  D |
| Question 4  A | Question 9  D | Question 14  A | Question 19  A |
| Question 5  A | Question 10  A | Question 15  D | Question 20  C |

**SECTION B – Extended Response Questions** Answer all questions in the space provided.

**Question 1 (8 marks)**

a) Prims builds up a solution to an optimization problem one step at a time, looking only at an edge of least weight among those that connect a new vertex to the MST. It does not care about the long-term implications of the edge that is selected; it only picks the best edge based on the current step.  Greedy algorithms pick the next most favourable option, often using a priority queue to sort the options.  This is done until some criteria is met.

b) minimum spanning tree for the graph shown above using Prim's Algorithm.  State the cost of the tree.



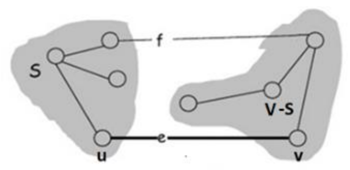
cost = 16

c) A simple implementation of Prim's, using an adjacency list graph representation and linearly searching an array of weights to find the minimum weight edge, to add requires O($|V|^2$) running time.

d) *If each edge has a distinct weight then there will be only one, unique minimum spanning tree*. If the edge weights are not unique, there may be multiple MSTs.

**Question 2** (10 marks)

a) The Loop invariant condition in this algorithm is at the end of each loop there is a MST connecting a subset of the vertices V. Let S be a subset of Vertices of V for which at each iteration of the loop that are connected by a MST.

| | |
|---|---|
|  | At the end of each loop the MST will increase by one edge, which will be the cheapest edge that connects vertices in set S to vertices in set "V-S". Hence the loop invariant condition will be maintained until the set "V-S" is empty and S=V. |

b)

```
function AlterMST(Input graph MST, edge u-v)
       TheEdgeweight := w(u-v)
       For each edge E adjacent to vertices in List {u,v} ordered by weight do
                    If (edge E is not already in MST) and (weight(E) < TheEdgeWeight) then
                              CanIncrease:= TheEdgeWeight – weight(E)-1
                    End if
       End do
       Return CanIncrease
End function
```

**Question 3** (12 marks)

a) In your own words describe the characteristics that make a problem tractable or untractable.

Tractable – best algorithm for problem has polynomial time complexity

Intractable – best algorithm for the problem has exponential or worse time complexity

b) What is the difference between problems in the class “P” and the problems in class “NP”? Give an example of a problem in “P” and “NP”.

P problems have solutions that can be found correctly in polynomial time

NP problems have solutions that cannot be found in polynomial time but correct solutions can be verified in polynomial time. P problem – ordering a list of numbers; NP problem – Knapsack problem has a solution found in exponential time

c) What are the characteristics of an NP-complete problem and how does it relate to P, NP and NP-Hard problems? Give an example of an NP-Complete problem.

NP-Complete problems are considered to be a subset of both NP and NP-Hard problems when P is not equal to NP. NP-Complete problems have solutions that cannot be found in polynomial time when a solution is found it cannot always be verified in polynomial time that is why it overlaps with NP-Hard problems cannot be solved or solutions verified in polynomial time. NP-Complete problems are grouped together since if a solution is found for one of them that is polynomial moving the problem to “P” class then all of NP-Complete will be “P” since the problems are similar. An example of an NP-Complete problem is one of {TSP, Knapsack problem, 3 colour map colouring,…..}

d) What strategies can be sued to find an “acceptable” solution for an NP-Complete problem?

Heuristics can be used to find an approximation to the correct solution. Strategies such as optimisation by hill climbing or other greedy methods may be used. Random exploration of options can also be used, such as combining hill climbing with random selections and probabilistic methods in the process described as simulated annealing.

e) Describe in detail that characteristics of an undecidable problem in Computer Science.

An undecidable problem is a problem that requires a Boolean (yes or no) answer that is uncomputable. Uncomputable means that no algorithm (algorithm = a set of instructions that completes in finite time) exists to solve the problem.

f) Suppose you’re working on a lab for a programming class,, have written your program, and start to run it. After five minutes, it is still going. Does this mean it’s in an infinite loop, or is it just slow and doing calculations? Explain your answers.

There is no way of knowing in advance if the program will ever stop. This is an example of an undecidable problem. Turing showed that it is impossible to determine if a program contains an infinite loop which he called the “Halting problem”.

**Question 4 (10 marks)**

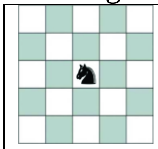
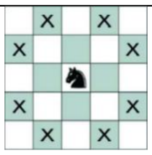
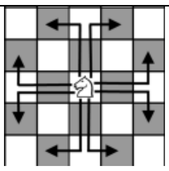
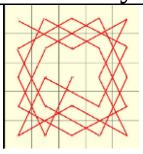
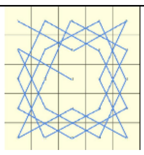
**a)** What properties of problems are best suited for solving using backtracking design patterns? How do Backtracking algorithms compare with naïve Brute Force algorithms in terms of efficiency? (3 marks)

- Backtracking is an algorithmic design pattern that tries different solutions until finds a solution that "works". Problems that are typically solved using the backtracking technique have the following property in common. These problems can only be solved by trying every possible configuration and each configuration is tried only once.
- A Naive solution for these problems is to try all configurations and output a configuration that follows given problem constraints.
- Backtracking works incrementally and is an optimization over the Naive solution where all possible configurations are generated and tried.

**b)** Describe how Backtracking algorithms strategies work to solve problems, giving details of the abstract data types that they commonly use. (3 marks)

- **Backtracking** works in an incremental way to attack problems. Typically, we start from an empty solution array (vector) and one by one add items, the meaning of item varies from problem to problem.
- When we add an item, we check if adding the current item violates the problem constraint, if it does then we remove the item and try other alternatives.
- If none of the alternatives works out then we go to the previous stage and remove the item added in the previous stage. If we reach the initial stage back then we say that no solution exists. If adding an item doesn't violate constraints then we recursively add items one by one. If the solution array (vector) becomes complete then we print the solution.

**The Knight's Tour problem** Given a N*N board with the Knight placed anywhere on an empty board. Moving according to the rules of chess knight must visit each square exactly once.

| | | | | |
|---|---|---|---|---|
|  Place a knight anywhere on a 5x5 chess board to start. |  X's indicate possible moves from start. |  25 cell array |  A possible solution is shown. |  Another possible solution . |
| Constraint: Knights chess pieces move in an **"L-shape"** that is: two squares in any direction vertically followed by one square horizontally, **or** two squares in any direction horizontally followed by one square vertically. | | | | |

**c)** Explain the main actions of a Backtracking algorithm using ADTs described in part b) which solves the Knight's Tour problem for an N*N chess board, from any starting position. (4 marks)

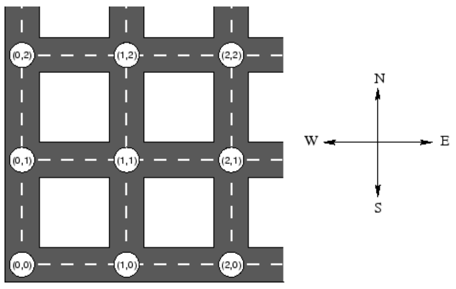
```
If all squares are visited
   return the solution
Else
   • Add one of the next moves to solution array (vector) and recursively check if this move leads to
     a solution.
   • If the move chosen in the above step doesn't lead to a solution then remove this move from the
     solution vector and try other alternative moves.
   • If none of the alternatives work then return false (Returning false will remove the previously
     added item in recursion and if false is returned by the initial call of recursion then "no solution
     exists" )
```

**Question 5 (10 marks)**

## Problem: Survival

Natural disasters have squashed Coda City and it is now a wasteland roamed by dangerous rats. You have taken refuge in the sewers underneath the city. Fortunately you have your solar powered laptop. Unfortunately, you have no food with you, and so you must leave the sewer to stock up on supplies.

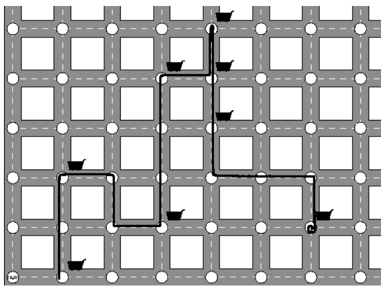
Coda City consists of streets running north-south and east-west, forming a grid. Every street intersection has a manhole. The intersections are given coordinates as shown on the diagram:



Some street intersections contain abandoned shopping trolleys containing food items, cast aside by panicking civilians during the disaster. Since being outside at all is a massive risk, you decide to visit all the shopping trolleys in one trip.

Your plan is as follows: You will emerge from any manhole. You will then run along the streets, emptying trolleys as you go. **You can only travel in an easterly direction, do not head west, as sun rays will blind you.** Once you have been to all the trolleys, you will climb down any manhole.

Your task is, given the locations of abandoned trolleys within the city, to determine the smallest possible distance that you must travel above ground in order to collect food from all the abandoned trolleys.

| Survival Input | Survival Output |
|---|---|
| Integer T the number of trolleys<br><br>List Tcoords a list of (x,y) coordinates showing the position of the kth trolley | should be a single integer which is the shortest distance you must travel above ground to collect all the food from all the trolleys. Remember, you *cannot go west* . |
| **Sample Input** | **Sample Output** |
| Algorithm Survival(8, {(1,0),(4, 3),(3, 4),(4,4),(1, 2),(3,1),(4,5),(6,1)})<br><br>The trolleys in the sample data are shown in the diagram above. | 16<br>Explanation<br>The thick black line shows one possible shortest path which visits all trolleys, starting from the bottom-left trolley and ending at the bottom-right trolley.<br><br>The length of this path is 16. Although many other paths are possible, there are no shorter paths, therefore the answer is 16. |

If a brute force naïve approach is taken to write an algorithm to find the solution, what will be the worst case time complexity of the naïve approach? Explain your response

a) The naïve brute force approach will create a solution of O(n!) since at every decision point there will be n-1 trolleys to choose from. The naïve approach will make the problem intractable for large n.

```
b) Function Survival(Input T, List TCoords(x,y) locations)
// T is the number of Trolleys, TCoords is a list of Trolleys with their (x,y) location
// Order Trolleys by x coordinate as cannot move west, only can go east, south and north
      OrderedTrolleyList := Sort(TCoords by x-coordinate)
      FirstTrolley := first item of OrderedTCoords
      X0 := FirstTrolley x-coordinate
      Y0 := FirstTrolley y-coordinate
      // emerge from any eastern most manhole is at (X0,Y0) proceed greedily heuristic
      Remove FirstTrolley from OrderedTrolleyList
      Add FirstTrolley to CollectedTrolleyList
      TotalDistance := 0
      While (OrderedTrolleyList is not empty) do
            MinDist := MaxInteger
            MinDistIndex := -1
            For i:=1 to length(OrderedTrolleyList) do
            // Find westernmost trolley with minimum distance to last collected Trolley
                  NewX := OrderedTrolleyList[i] x-coordinate
                  NewY := OrderedTrolleyList[i] y-coordinate
                  Distance[i] := (NewX – X0) + (NewY – Y0)
             End do
             FindWesternmostNearest(Distance,MinDist,MinDistIndex)
            // Next nearest westernmost trolley is MinDist given by MinDistIndex
            Add OrderedTrolleyList[MinDistIndex] to CollectedTrolleyList
            X0 := OrderedTrolleyList[MinDistIndex] x-coordinate
            Y0 := OrderedTrolleyList[MinDistIndex] y-coordinate
            TotalDistance := TotalDistance + MinDist
            Remove OrderedTrolleyList[MinDistIndex] from OrderedTrolleyList
      end do
      return TotalDistance
end Function
```

c) A greedy Nearest Neighbour heuristic approach has been used in the algorithm above. This will give a good approximate solution in most cases, but may not give the correct solution. This problem is similar to the Travelling Salesman problem with the added constraint of not being able to travel west, which makes it an NP-Complete problem requiring a heuristic approach in this case using a Greedy strategy.

**Question 6** (10 marks)

a. Describe in your own words and in Pseudocode how the Minimax algorithm works on a game tree. (2 marks)

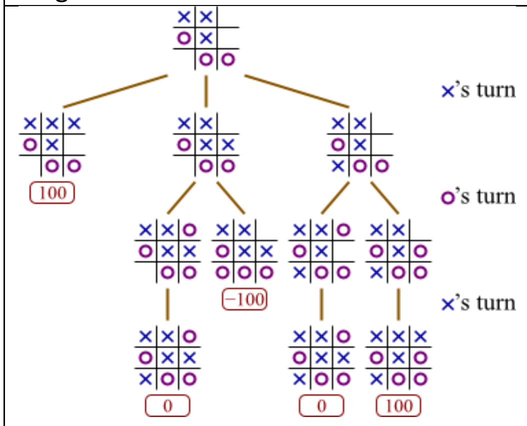
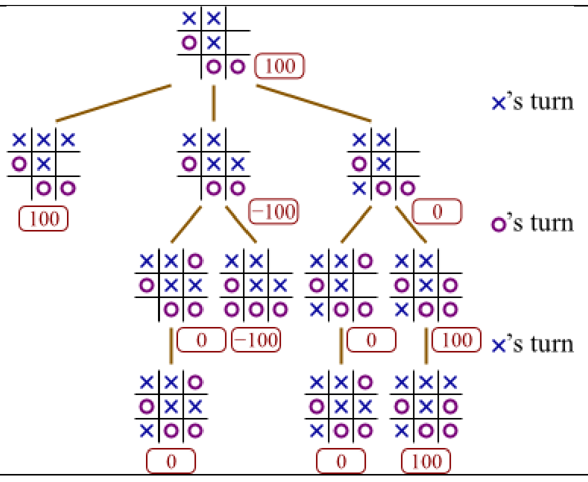
In English

- In a two player zero sum game the minimax algorithm seeks to maximise the score for a player against his opponent.
- The algorithm assumes the opponent is equally skilled and will seek to minimise the advantage to the other player.
- The scores are tabulated on a tree where the leaf nodes show the advantage to the player.
- The player seeks the maximum score of the child branches, the opponent seeks the minimum score of the child branches.

In Pseudocode

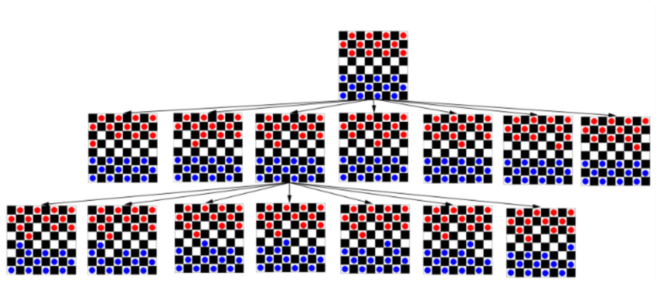
```
function minimax(node, depth, maximizingPlayer)
  if depth = 0 or node is a terminal node
    return the heuristic value of node
  if maximizingPlayer
    bestValue := -∞
    for each child of node
      val := minimax(child, depth - 1, FALSE)
      bestValue := max(bestValue, val)
    return bestValue
  else
    bestValue := +∞
    for each child of node
      val := minimax(child, depth - 1, TRUE)
      bestValue := min(bestValue, val)
    return bestValue
end function

(* Initial call for maximizing player *)
minimax(origin, depth, TRUE)
```

b. Label *all* internal nodes of the following tic-tac-toe game tree with the value that Minimax algorithm would compute. The leaves have already been labelled. (2 marks)

| Original | Solution |
|---|---|
|  |  |

**Question 6** - continued

c. Consider the following subset of game tree for the game of checkers below. In case you didn't know, Checkers is a zero-sum game played by two players on an 8x8 board. Each player begins with 12 counters and the aim is to capture the opponent's counters and remove them from the board.



Is it feasible to analyse every possible node in the game tree? Justify your response. (2 marks)

- It is not feasible to analyse every node due to the exponential growth of the game tree.
- As there are from 1-24 counters on the board at any one time that can move to 64 possible locations.

d. How could a heuristic be applied to the checkers game tree? Describe the mathematical principles that could be used to determine a "good guess" in terms of what the next best move might be for a player. (2 marks)

- One strategy could be to only look ahead only a few moves and then apply a heuristic function to the tree.
- A heuristic function could be created to analyse the optimum move based on a combination of minimax and backtracking. The function could determine good paths and prune back paths that are considered not so favourable, this would then reduce the size of the decision tree and allow a decision to be made in a feasible time.

e. How can you evaluate how good the checkers game heuristic is? Explain your response. (2 marks)

- Can it beat a skilled human checkers player?
- A good heuristic could be evaluated by many human opponents who were expert at checkers.

**Question 7** (10 marks)

Divide and Conquer – Master Theorem

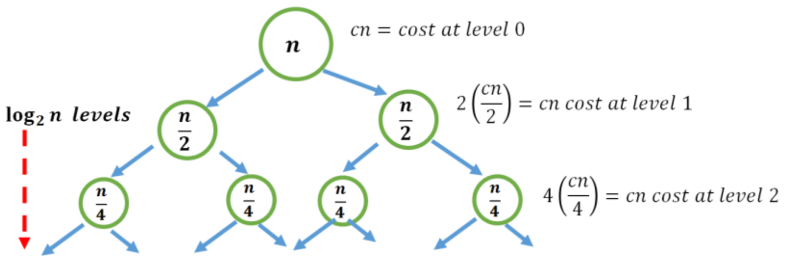
a) Classify the following Master Theorems in terms of Big-O notation.

- $T(n) = 5T\left(\frac{n}{8}\right) + n$ a=5,b=8, k=1 => $\frac{a}{b^k} = \frac{5}{8^1} < 1 \therefore\ O(n)$
- $T(n) = 4T\left(\frac{n}{2}\right) + n^2$ a=4, b=2, k=2=> $\frac{a}{b^k} = \frac{4}{2^2} = 1 \therefore\ work\ is\ O(n^2 \log_2 n)$
- $T(n) = 3T\left(\frac{n}{4}\right) + f(\sqrt{n})$ a=3, b=4, k=0.5 => $\frac{a}{b^k} = \frac{3}{4^{0.5}} > 1\quad O(n^{\log_2 3})$

b) Consider the pseudocode and show the complexity of MergeSort is O(nlogn)

Informal Analysis of Mergesort Runtime for $\boldsymbol{n}$ items

"Unroll" the recursion and represent it by a tree where each recursion level has more children. Let "c" be a constant value representing how many statements are needed to sort at each level.



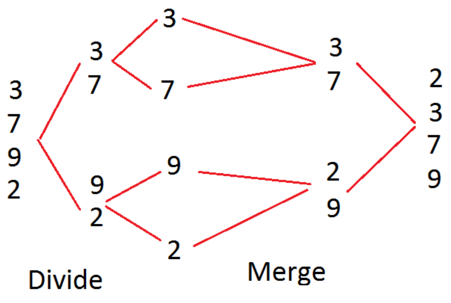
$n$
$cn = cost\ at\ level\ 0$
$\log_2 n\ levels$
$\frac{n}{2}$
$\frac{n}{2}$
$2\left(\frac{cn}{2}\right) = cn\ cost\ at\ level\ 1$
$\frac{n}{4}$
$\frac{n}{4}$
$\frac{n}{4}$
$\frac{n}{4}$
$4\left(\frac{cn}{4}\right) = cn\ cost\ at\ level\ 2$

$Total\ runtime\ = cn \times \log_2 n + cn$

$Runtime\ in\ Big\ O\ notation\ is\ O(nlogn)$

c) Master theorem $a = 2, b = 2, k = 1\ \frac{a}{b^k} = \frac{2}{2^1} = 1\ => O(nlogn)$

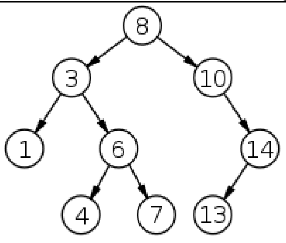
d) Given an unsorted list of numbers {3,7,9,2} show the detailed steps that mergesort would use to sort the list.



e) Mergesort is a divide and conquer design pattern and has better time complexity than the Brute Force method such as BubbleSort.

**Question 8** (10 marks)

Consider the following Algorithm that converts a Binary Tree into an ordered List.



A binary search tree is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted *left* and *right*. The tree additionally satisfies the binary search tree property, which states that the key in each node must be greater than all keys stored in the left sub-tree, and smaller than all keys in right sub-tree.

```
Procedure Tree2List(input TreeNode, output NodeList)
            // TreeNode.right follows the right subtree
            // TreeNode.left follows the left subtree
            If (TreeNode has children) then
                  Tree2List(TreeNode.right, NodeList)
                  Append TreeNode to NodeList
                  Tree2List(TreeNode.left, NodeList)
            Else
                  Append TreeNode to NodeList
            End if
End procedure
```

| |
|---|
| |
| |
| |
| |
| O(1) if |
| |
| T(n/2) |
| O(1) |
| T(n/2) |
| O(1) |
| O(1) |
| |
| |
| |
| |
| |
| |

a. Label the time complexity of each command on the pseudocode above using the table. (2 marks)

b. Determine the recurrence relation for the time complexity of this algorithm, (2 marks)

Since the tree traversal is being split into two sub-problems, left of current node and right of current node, this is an example of divide and conquer.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^o)$$

c. and hence the time complexity of the algorithm above. (1 mark)

*Using the Master Theorem*

- $a = 2, b = 2, k = 0 \Rightarrow \frac{a}{b^k} = \frac{2}{2^0} > 1 \Rightarrow \quad O\left(n^{\log_2 2}\right) = O(n)$

**Question 8** (condintued

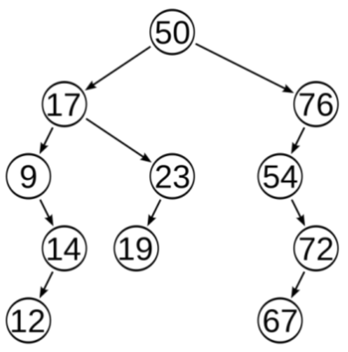
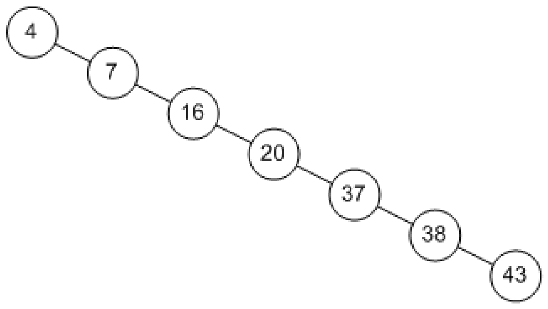
d. What is the time complexity of searching for a particular value in a binary tree? Explain and justify your answer. (2 marks)

```
function Find-recursive(key, node):   // call initially with node = root
    if node = Null or node.key = key then
        return node
    else if key < node.key then
        return Find-recursive(key, node.left)
    else
        return Find-recursive(key, node.right)
```

if we use the recurrence relation for this algorithm is T(n)=T(n/2)+1 this is assuming that splitting of data will occur, roughly half and half, *Using the Master Theorem*

- $a = 1, b = 2, k = 0 \Rightarrow \frac{a}{b^k} = \frac{1}{2^0} = 1 \implies \; O(n^0 \log_2 n) = O(\log n)$

e. Consider the following binary trees and explain how their structure may or may not impact on searching for a particular value. (3 marks)

| Case 1 | Case 2 |
|---|---|
|  50<br>17<br>76<br>9<br>23<br>54<br>14<br>19<br>72<br>12<br>67 |  |
| In this balanced tree – the value will be found on average in… $O(\log n)$<br><br>because on average recurrence relation for this algorithm is T(n)=T(n/2)+1 this is assuming that splitting of data will occur, roughly half and half | In this unbalanced tree – the worst case of the time complexity will occur where every value will be compared to the particular value.<br><br>This will result in "n" comparisons, therefore the worst case time complexity is shown O(n)<br><br>the recurrence relation for this algorithm is T(n)=T(n-1)+1 this is assuming the split divides the data into 1 and (n-1)<br><br>By telescoping<br><br>T(n)-T(n-1)=O(1)<br>T(n-1)-T(n-2)=O(1)<br>……………………<br>T(2)-T(1)=O(1)<br>T(1)-T(0)=O(1)<br>===============<br><br>T(n)-T(0)=n*O(1) ➔T(n)=O(n) |