

18CSC201J
DS LAB EXERCISES
UNIT - 4

DATE: 14.01.2022

NAME: ANINDYA SHANKAR DASGUPTA
REG NO: RA2011026010120
CSE Q1 (AI-ML)

RA2011026010120

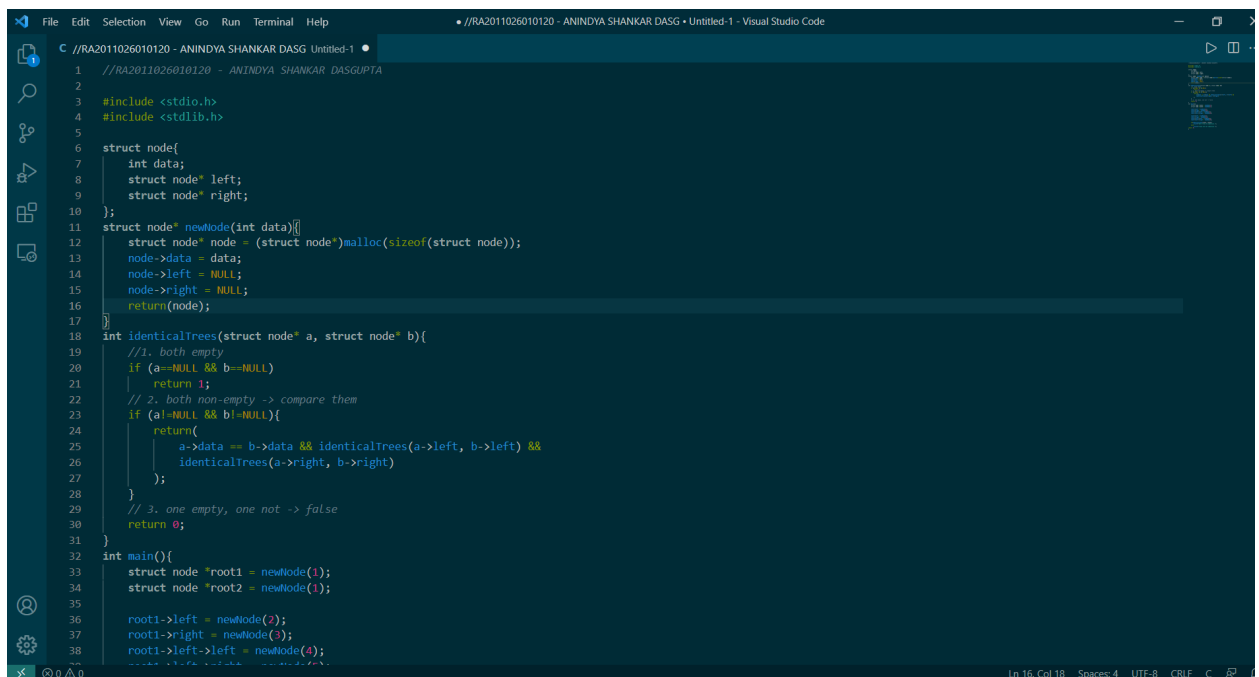
Simple Problems:

Q:

1. (1.) Given the roots two binary trees a and b. Write a program to check if both the trees are similar or not. Two binary trees are referred to as similar if they have identical structure and each node has the same value.

A:

- Approach:
 1. The basic approach to the problem is first we create 2 binary trees and then we compare each element recursively. The base condition is if the root node of both trees is NULL, still both are equal. If both the conditions are false we return 0 i.e. both the binary trees are not identical.
 2. In the program it is not stated whether the trees have to be input during run-time. In this case, we have taken pre-defined input for simplicity.
 3. Also, “identical structure” term is pretty vague, and does not relay the exact conditions to be checked.
- Code (with documentation):



```
1 //RA2011026010120 - ANINDYA SHANKAR DASG - Untitled-1
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 struct node{
7     int data;
8     struct node* left;
9     struct node* right;
10 };
11 struct node* newNode(int data){
12     struct node* node = (struct node*)malloc(sizeof(struct node));
13     node->data = data;
14     node->left = NULL;
15     node->right = NULL;
16     return(node);
17 }
18 int identicalTrees(struct node* a, struct node* b){
19     //1. both empty
20     if (a==NULL && b==NULL)
21         return 1;
22     // 2. both non-empty -> compare them
23     if (a!=NULL && b!=NULL){
24         return(
25             a->data == b->data && identicalTrees(a->left, b->left) &&
26             identicalTrees(a->right, b->right)
27         );
28     }
29     // 3. one empty, one not -> false
30     return 0;
31 }
32 int main(){
33     struct node *root1 = newNode(1);
34     struct node *root2 = newNode(1);
35
36     root1->left = newNode(2);
37     root1->right = newNode(3);
38     root1->left->left = newNode(4);
```

```

File Edit Selection View Go Run Terminal Help
C //RA2011026010120 - ANINDYA SHANKAR DASG - Untitled-1 - Visual Studio Code
34 struct node *root2 = newNode(1);
35
36 root1->left = newNode(2);
37 root1->right = newNode(3);
38 root1->left->left = newNode(4);
39 root1->left->right = newNode(5);
40
41 root2->left = newNode(2);
42 root2->right = newNode(3);
43 root2->left->left = newNode(4);
44 root2->left->right = newNode(5);
45
46 if(identicalTrees(root1, root2))
47     printf("Both tree are identical.");
48 else
49     printf("Trees are not identical.");
50 return 0;
51 }

```

● Dry-run:

<u>I</u>	<ol style="list-style-type: none"> 1. Take the input of all the elements of both the tree 2. Pass the values of both the tree into identicalTree() function 3. [IF] both the tree are NULL Return 1 [END OF IF]
<u>II</u>	<ol style="list-style-type: none"> 4. [IF] both the tree are not NULL Check if the roots are equal Repeat step 3 and 4 for the left side of the tree [RECURSIVE] Repeat step 3 and 4 for the right side of the tree [RECURSIVE] Return the final integer [1 or 0] as per the output [END OF IF]
<u>III</u>	<ol style="list-style-type: none"> 5. If one tree is NULL and other is not then return 0. in this example, the tree has nodes, so it is not empty <ol style="list-style-type: none"> i) Comparing root node we see <i>a->data = b->data = 1</i> ii) For identicaltree(a->left, b->left), we move to first left node of tree, i.e. we see the subtree whose root is a->left /b->left of main tree <i>a->data = b->data = 2</i> As it is a recursive call, for identicaltree(a->left, b->left), we move to left node of left subtree, whose root is 2, here <i>a->data = b->data = 4</i>, (1 return to the recursive call) Again, as a recursive call, indential(a->left, b->left) and identicaltree(a->right, b->right) it will fulfil base condition i.e. <i>a->data = b->data = NULL</i>, it return 1 to recursive call. Also, identicaltree(a->right, b->right), we move to right node of the left

	<p>subtree, whose root is 2, here $a \rightarrow data = b \rightarrow data = 5$, (1 return to the recursive call) Again, as a recursive call, identical(a->right, b->right) and identical(a->left, b->left) it will fulfil base condition i.e. $a \rightarrow data = b \rightarrow data = NULL$, it return 1 to recursive call.</p> <p>iii) For identicaltree(a->right, b->right) we move to the first right node of the tree, i.e. we see the subtree whose root is a->right/b->right of the main subtree, $a \rightarrow data = b \rightarrow data = 3$ Again, as a recursive call, identical(a->left, b->left) and identical(a->right, b->right), it will fulfil base condition, i.e. $a \rightarrow data = b \rightarrow data = NULL$</p> <p>iv) Now we get all the elements are identical, hence <i>both trees are identical</i></p>
--	--

- Input/Output:

Input-



```

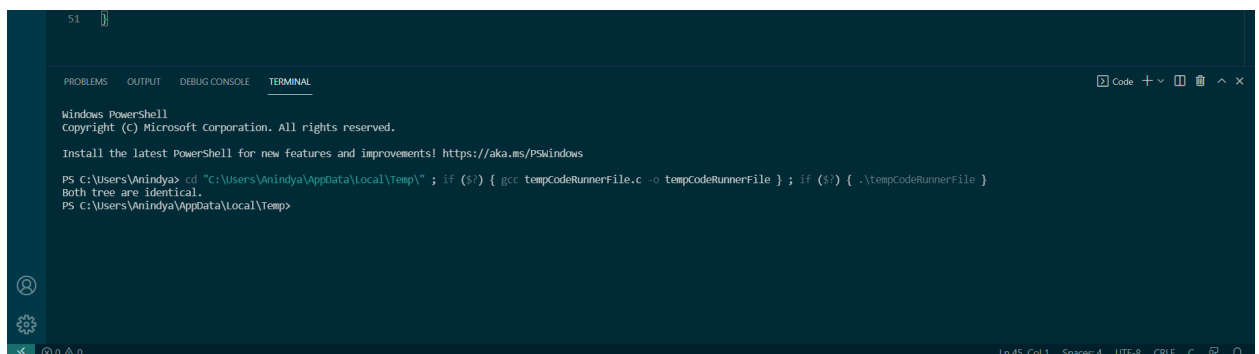
51
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Anindya> cd "C:\Users\Anindya\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Both tree are identical.
PS C:\Users\Anindya\AppData\Local\Temp>
Ln 45, Col 1 Spaces: 4 UTF-8 CRLF C

```

Output-



```

51
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Anindya> cd "C:\Users\Anindya\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Both tree are identical.
PS C:\Users\Anindya\AppData\Local\Temp>
Ln 45, Col 1 Spaces: 4 UTF-8 CRLF C

```

- Result:

1. Time-complexity:

Considering **n** nodes in binary tree with root 1 and **m** nodes in binary tree with root 2, time taken to traverse the tree will depend on number of nodes

So, for Root 1 is **$O(n)$** And for Root 2 is **$O(m)$**

Total Time complexity is $O(n+m)$

2. Space Complexity:

Considering **n** nodes in binary tree with root 1 and **m** nodes in binary tree with root 2, space occupied by the tree will depend on number of nodes

For each node we have 12 bytes of memory

For (n+m) nodes we have $((n+m)*12)$ bytes of memory

Other element can be considered with constant memory occupied = c bytes

Total Space complexity is $O(12*(n+m) + c)$

RA2011026010120

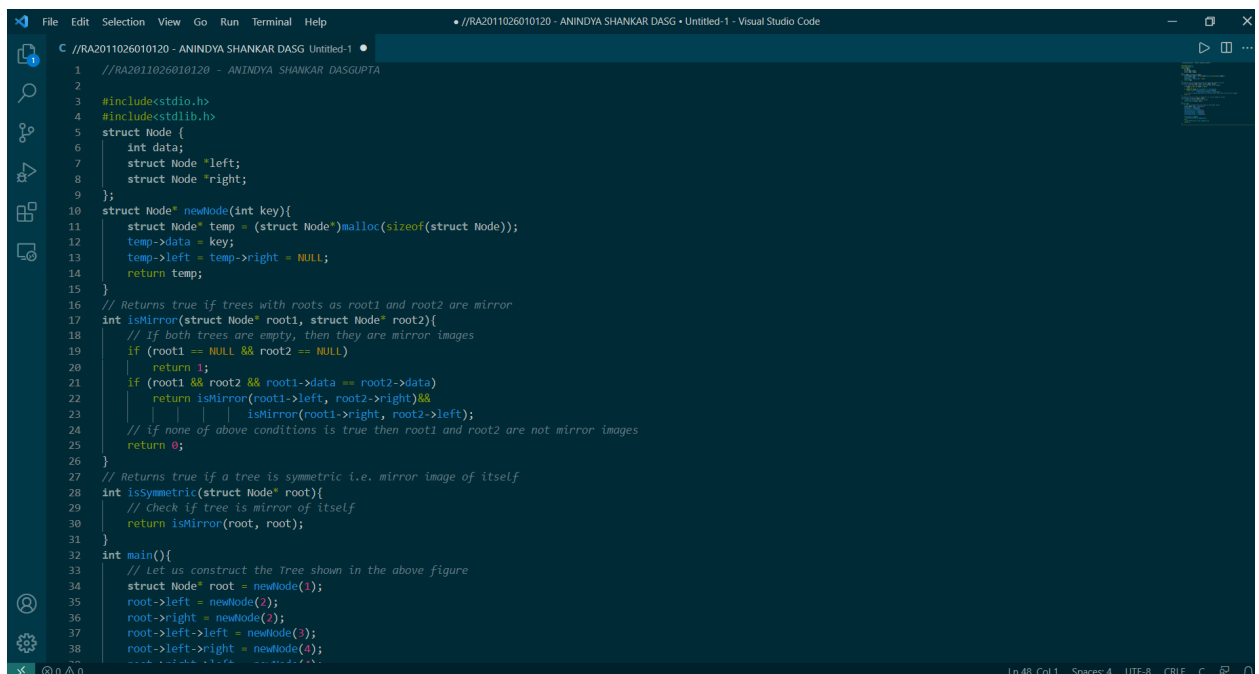
Scenario based or Difficult Problems:

Q:

1. (1.) Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

A:

- Approach:
 1. We have to create a binary tree, then we have to traverse the tree twice such that both traversals go together i.e., one traversal is to travel the left subtree and other is for the right subtree. The comparison of elements is done such that the left child node of the left subtree should be equal to the right child node of the right subtree and vice versa.
 2. The base condition will be the tree does not have any child node and points to NULL.
 3. In this question too, it is not specified whether the tree is input at run-time. In this case, we have taken pre-defined input for simplicity.
- Code (with documentation):



```
//RA2011026010120 - ANINDYA SHANKAR DASG - Untitled-1 - Visual Studio Code
1 //RA2011026010120 - ANINDYA SHANKAR DASGUPTA
2
3 #include<stdio.h>
4 #include<stdlib.h>
5 struct Node {
6     int data;
7     struct Node *left;
8     struct Node *right;
9 };
10 struct Node* newNode(int key){
11     struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
12     temp->data = key;
13     temp->left = temp->right = NULL;
14     return temp;
15 }
16 // Returns true if trees with roots as root1 and root2 are mirror
17 int isMirror(struct Node* root1, struct Node* root2){
18     // If both trees are empty, then they are mirror images
19     if (root1 == NULL && root2 == NULL)
20         return 1;
21     if (root1 && root2 && root1->data == root2->data)
22         return isMirror(root1->left, root2->right) &&
23             isMirror(root1->right, root2->left);
24     // If none of above conditions is true then root1 and root2 are not mirror images
25     return 0;
26 }
27 // Returns true if a tree is symmetric i.e. mirror image of itself
28 int isSymmetric(struct Node* root){
29     // Check if tree is mirror of itself
30     return isMirror(root, root);
31 }
32 int main(){
33     // Let us construct the Tree shown in the above figure
34     struct Node* root = newNode(1);
35     root->left = newNode(2);
36     root->right = newNode(2);
37     root->left->left = newNode(3);
38     root->left->right = newNode(4);
```

```

31 }
32 int main(){
33     // Let us construct the Tree shown in the above figure
34     struct Node* root = newNode(1);
35     root->left = newNode(2);
36     root->right = newNode(2);
37     root->left->left = newNode(3);
38     root->left->right = newNode(4);
39     root->right->left = newNode(4);
40     root->right->right = newNode(3);
41
42     if(isSymmetric(root))
43         printf("The tree is Symmetric");
44     else
45         printf("The tree is Not symmetric");
46     return 0;
47 }
48

```

● Dry-run:

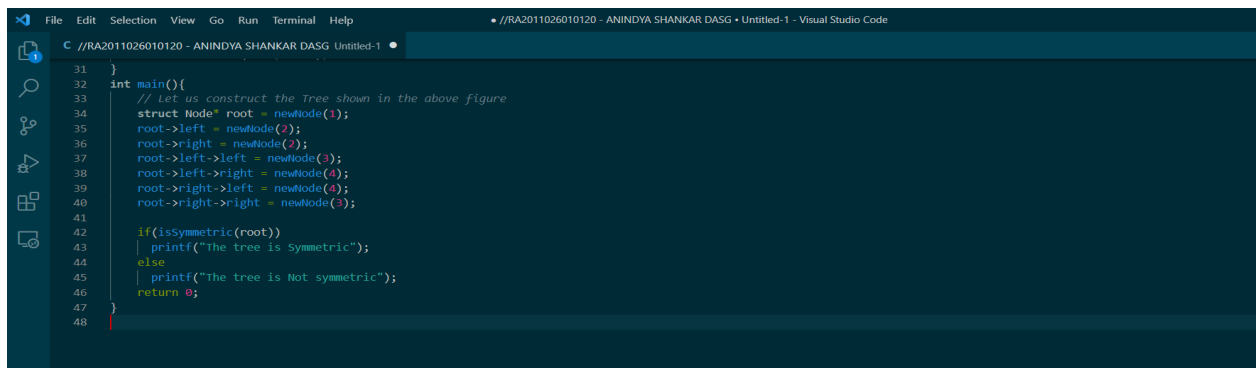
I.	<ol style="list-style-type: none"> 1. Take the input of the binary tree. 2. Pass the value of tree in isSymmetric() function 3. Here, in this function the values in tree is passed twice in the isMirror() function for both traversal. 4. [IF] value of root of the tree is NULL <p>Return 1 [END OF IF]</p>
II.	<ol style="list-style-type: none"> 5. [IF] both the trees are not NULL, root value is same <p>Repeat step 4 and 5 comparing root1->left and root2->right [RECURSIVE] Repeat step 4 and 5 comparing root1->right and root2->left [RECURSIVE] Return the final integer [1 or 0] as per the output [END OF IF]</p> <ol style="list-style-type: none"> 1. [ELSE] upper condition not satisfied, return 0 <p>In our example, we have a tree already made and is symmetrical</p>
III.	<p>Input: A= [1,2,2,3,4,4,3]</p> <ol style="list-style-type: none"> i) Since the root does not point to NULL, we traverse recursively in the isMirror(root1->left, root2->right), and in isMirror(root1->right, root2->left) ii) RECURSIVE for isMirror(root1->left, root2->right)

Now **root1->data = 2** become root of subtree and **root2->data=2** become root of subtree
 Here **root1->data = root2->data**
 Now within this recursion
 Here, **isMirror(root1->left, root2->right)**, **root1->data = 3** and **root2->data = 3** become root of the subtree
 Here **root1->data = root2->data**, return 1 to the recursive call
 Here, **isMirror(root1->right, root2->left)**, **root1->data = 4** and **root2->data = 4** become root of the subtree
 Here **root1->data = root2->data**, return 1 to the recursive call
 iii) **RECURSIVE for isMirror(root1->right, root2->left)** , now this works the same and gets the value 1 from the recursive call
 iv) Now **isSymmetric()** gets the value 1 and hence it return 1 to **main()** function
 Since the if condition of **main()** function is true – “**The Tree is Symmetric**”

- Input/Output:

Input-

a. [1,2,2,3,4,4,3]



```

31 }
32 int main(){
33     // Let us construct the Tree shown in the above figure
34     struct Node* root = newNode(1);
35     root->left = newNode(2);
36     root->right = newNode(2);
37     root->left->left = newNode(3);
38     root->left->right = newNode(4);
39     root->right->left = newNode(4);
40     root->right->right = newNode(3);
41
42     if(isSymmetric(root))
43         printf("The tree is Symmetric");
44     else
45         printf("The tree is Not symmetric");
46     return 0;
47 }
48

```

b. [1,2,2, NULL,3, NULL,3]



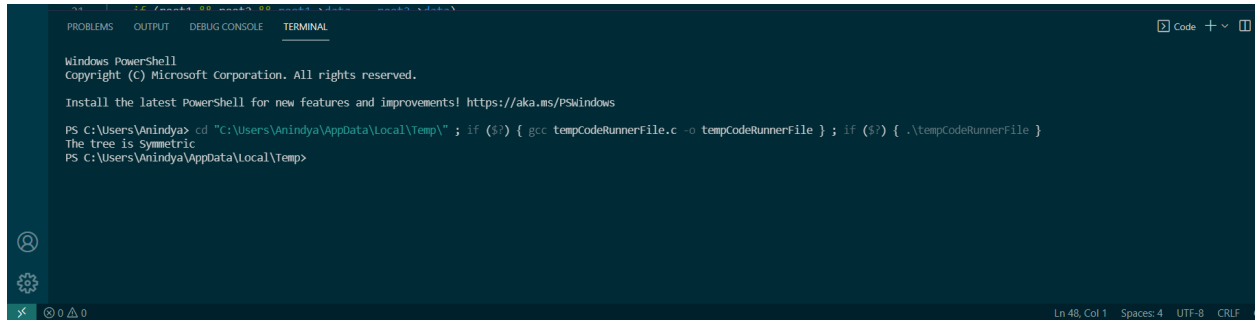
```

31 }
32 int main(){
33     // Let us construct the Tree shown in the above figure
34     struct Node* root = newNode(1);
35     root->left = newNode(2);
36     root->right = newNode(2);
37     root->left->right = newNode(4);
38     root->right->right = newNode(3);
39     if(isSymmetric(root))
40         printf("The tree is Symmetric");
41     else
42         printf("The tree is Not symmetric");
43     return 0;
44 }
45

```


Output-

a. For 1st input:

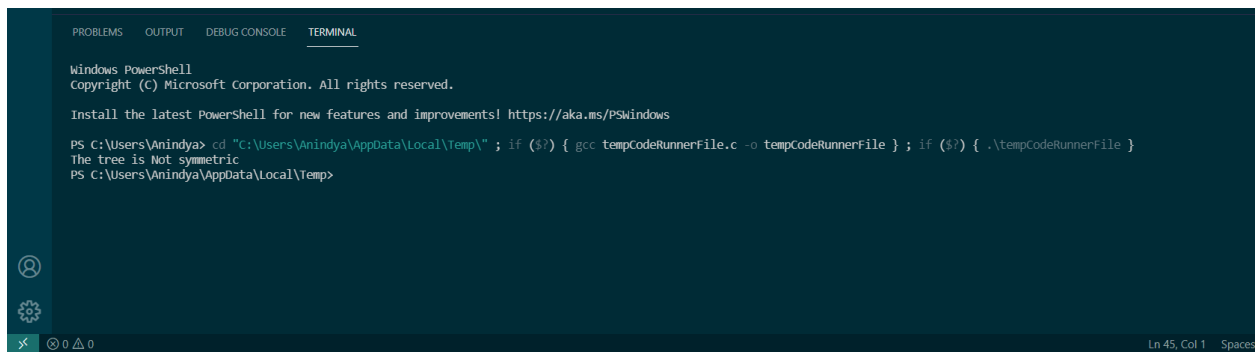


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Anindya> cd "C:\Users\Anindya\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
The tree is Symmetric
PS C:\Users\Anindya\AppData\Local\Temp>
```

b. For 2nd input:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Anindya> cd "C:\Users\Anindya\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
The tree is Not symmetric
PS C:\Users\Anindya\AppData\Local\Temp>
```

- Result:

1. Time-complexity:

Since the both the traversal depends on at least half of the elements in of the tree,
so time complexity of each traversal is $O(n/2)$.

Total Time complexity is $O(n)$.

2. Space Complexity:

For each node we have 12 bytes of memory

For n nodes we have $(n*12)$ bytes of memory

Other element can be considered with constant memory occupied = c bytes

Total Space complexity is $O(12*n + c)$.
