Roll Number:_____ Number of Pages:

Thapar Institute of Engineering and Technology, Patiala

Department of Computer Science and Engineering

END SEMESTER EXAMINATION

TENTATIVE SOLUTION

B. E. (Second Year): Semester-II (2018/19)	Course Code: UCS406 Course Name: Data Structures and Algorithms
May 11, 2019	Saturday, 09:00 Hrs – 12:00 Hrs
Time: 3 Hours, M. Marks: 100	Name of Faculty: SMG, SUG, SP, TBH, RKR, RAH, ASG, ANK

Note: Attempt all questions (sub-parts) in sequence. Assume missing data, if any, suitably.

Q1. Consider a hash table of size $m = 7$ and a corresponding hash function $h(k) = k \bmod m$. Compute the locations to which the keys **99, 59, 26, 50,** and **58** are mapped using the following collision resolution techniques.

(a) Chaining. **1 mark per correct entry.** **(5)**

(b) Quadratic probing. **(5)**

Answer

(a) Chaining

$h(k) = k \bmod m$ and $m = 7$

99 % 7 = 1; 59 % 7 = 3; 26 % 7 = 5; 50 % 7 = 1; 58 % 7 = 2

0	NULL		
1	99	50	NULL
2	58	NULL	
3	59	NULL	
4	NULL		
5	26	NULL	
6	NULL		

(b) Quadratic probing

99 % 7 = 1. The cell is empty, so place the key in index 1. No collisions!
59 % 7 = 3. The cell is empty, so place the key in index 3. No collisions!
26 % 7 = 5. The cell is empty, so place the key in index 5. No collisions!
50 % 7 = 1. Another key is in this cell. Collision!
Perform rehash: (1 + 1^2) % 7 => 2. The cell is empty, so place the key in index 2.
58 % 7 = 2. Another key is in this cell. Collision!
Rehash: (2 + 1^2) % 7 = 3. Another key is in this cell. Collision!
Rehash: (2 + 2^2) % 7 = 6. The cell is empty, so place the key in index 6.

0	
1	99
2	50
3	59
4	
5	26
6	58

0	50
1	99
2	58
3	59
4	
5	26
6	

Using $h(k) = (k+i+i^2) \% m$

For Quadratic Chaining, positions of first three elements remain same. For rest of the two it will vary as per the quadratic formula. Two possibilities are shown here.

Q2. (a) Draw a Binary Search Tree by sequentially inserting the following elements: **(4)**

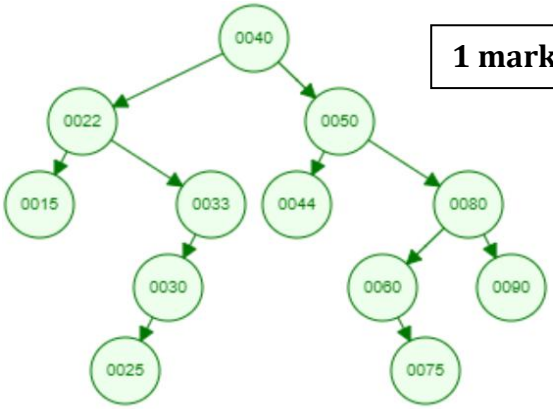
40, 50, 22, 33, 30, 80, 15, 25, 60, 90, 75, 44

(b) For the BST obtained in **Q2.(a)**, print the elements in Pre-order, In-order and Post-order traversals. **(3)**

(c) Delete in sequence 44, 50, and 40 from BST obtained in **Q2.(a)**. Show the BST after each deletion. **(3)**

Answer

(a) BST



1 mark per three correct entries.

(b) Traversals

In: 15, 22, 25, 30, 33, 40, 44, 50, 60, 75, 80, 90

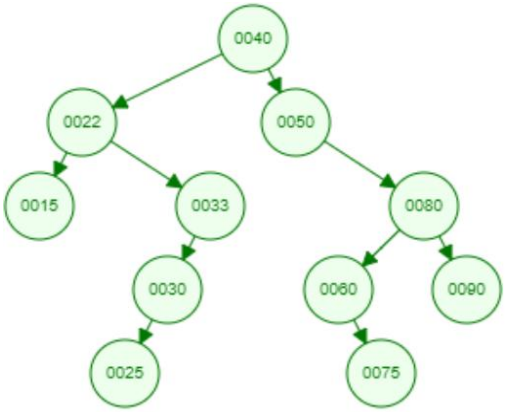
Pre: 40, 22, 15, 33, 30, 25, 50, 44, 80, 60, 75, 90

Post: 15, 25, 30, 33, 22, 44, 75, 60, 90, 80, 50, 40

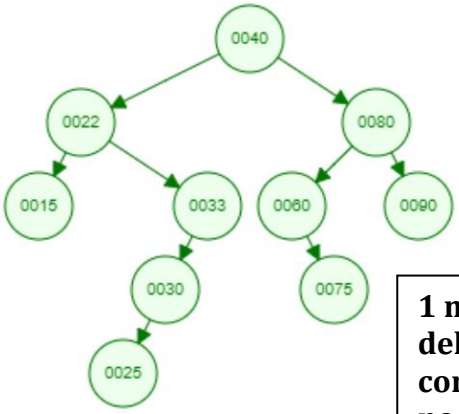
1 mark if completely correct, else 0.5 marks.

(c) Deletion

Delete 44.

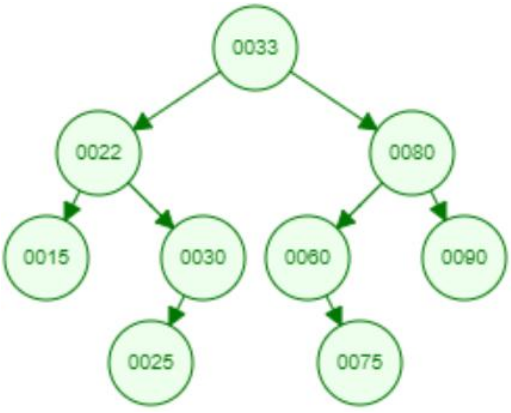


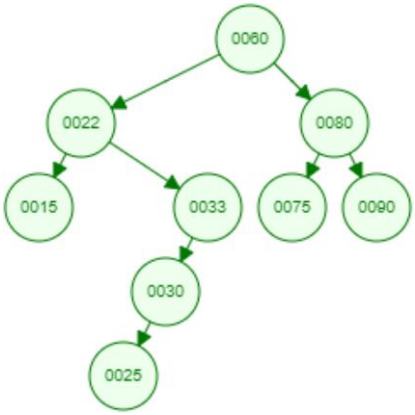
Delete 50.



1 mark per deletion if correct, else no marks.

Delete 40.





Q3. Define P and NP Complete class of problems. Give at-least one example for each class. **(4)**

P Definition: 1 Mark.
P Correct Example: 1 Mark.
NP Complete Definition: 1 Mark.
NP Complete Correct Example: 1 Mark.

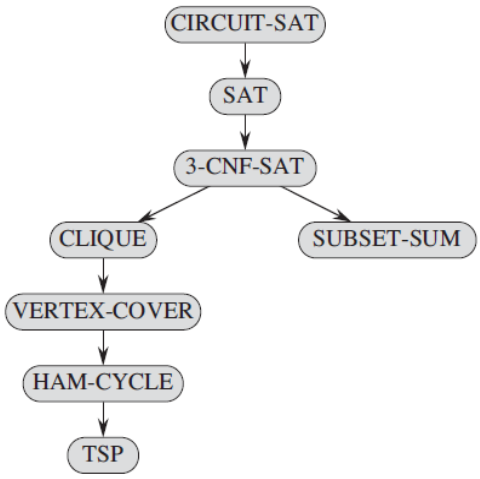
If NP is given with example, then it is not considered and zero marks are given, as NP Complete was asked in the question.

P Class:
Polynomial time solving. Problems which can be solved in polynomial time, which take time like O(n), $O(n^2)$, $O(n^3)$. Eg: finding maximum element in an array or to check whether a string is palindrome or not.

NP-Complete:
The group of problems which are both in NP and NP-hard are known as NP-Complete problem. Eg: circuit-satisfiability problem

A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

1. $L \in$ NP, and
2. $L' \leq_P L$ for every $L' \in$ NP.



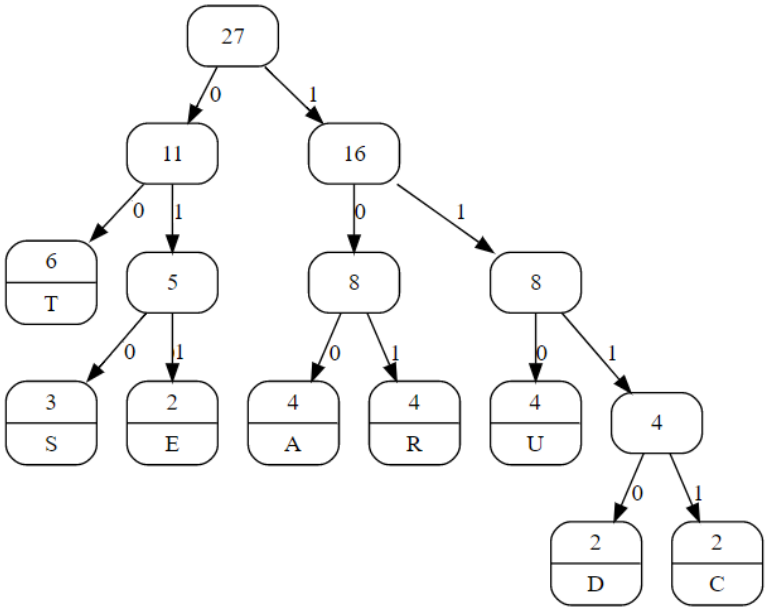
Q4. Show stepwise a tree that Huffman's greedy algorithm could produce for the sentence given in **Fig. 1**. Write optimal codes thus generated for all the different characters present in the sentence (**Fig. 1**). **(12)**

DATASTRUCTURESTRUCTURESDATA

Fig. 1

Note: Consider only those characters which are present in the given sentence.

4 Marks (frequency computation) --> 0.5 mark per character.
4 Marks (tree produced by Huffman's greedy algorithm)
4 Marks (code generation) --> 0.5 mark per character.



Char	Freq	Code
D	2	1110
A	4	100
T	6	00
S	3	010
R	4	101
U	4	110
C	2	1111
E	2	011

Q! Use dynamic programming to fully parenthesize the product of four matrices, i.e. $\boldsymbol{A}\ [\mathbf{10 \times 5}], \boldsymbol{B}\ [\mathbf{5 \times 20}], \boldsymbol{C}\ [\mathbf{20 \times 10}], \boldsymbol{D}\ [\mathbf{10 \times 5}]$, such that the number of scalar multiplications gets minimized. Show each and every step. **(10)**

Step 1: p[] = {10, 5, 20, 10, 5}, n = p.length – 1 = 5 – 1 = 4.

Step 2: m[1][1] = m[2][2] = m[3][3] = m[4][4] = 0

Step 3: (Chain length = 2)

m[1][2] = m[1][1] + m[2][2] + 10 × 5 × 20 = 1000.
m[2][3] = m[2][2] + m[3][3] + 5 × 20 × 10 = 1000.
m[3][4] = m[3][3] + m[4][4] + 20 × 10 × 5 = 1000.
s[1][2] = 1, s[2][3] = 2, and s[3][4] = 3

Step 4: (Chain length = 3)

m[1][3] = min(
m[1][1] + m[2][3] + 10 × 5 × 10 = 0 + 1000 + 500 = 1500,
m[1][2] + m[3][3] + 10 × 20 × 10 = 1000 + 0 + 2000 = 3000)
m[2][4] = min(
m[2][2] + m[3][4] + 5 × 20 × 5 = 0 + 1000 + 500 = 1500,
m[2][3] + m[4][4] + 5 × 10 × 5 = 1000 + 0 + 250 = 1250)
s[1][3] = 1 and s[2][4] = 3

Step 5: (Chain length = 4)

m[1][4] = min(
m[1][1] + m[2][4] + 10 × 5 × 5 = 0 + 1250 + 250 = 1500,
m[1][2] + m[3][4] + 10 × 20 × 5 = 1000 + 1000 + 1000 = 3000,
m[1][3] + m[4][4] + 10 × 10 × 5 = 1500 + 0 + 500 = 2000)
s[1][4] = 1

m/s	1	2	3	4	
1	0	1000/1	1500/1	1500/1	7 Marks (m[][]) • 1 Mark for diagonal 0's. • 1 Mark/rest of the entries. 1 Mark (s[][])
2		0	1000/2	1250/3	
3			0	1000/3	
4				0	

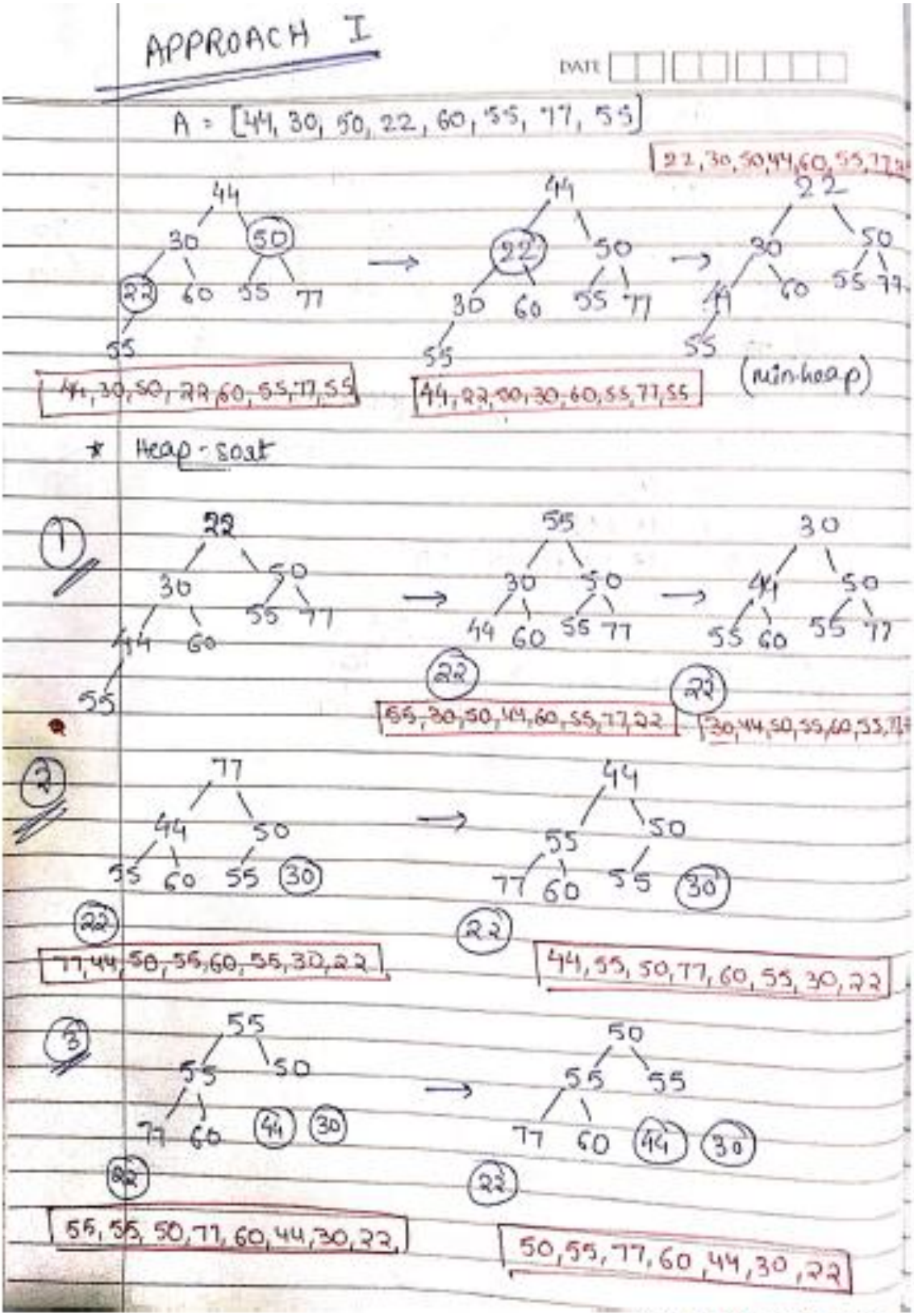
PrintOptimalParens(s,1,4) → (
PrintOptimalParens(s,1,1) → A
PrintOptimalParens(s,2,4) → (
PrintOptimalParens(s,2,3) → (
PrintOptimalParens(s,2,2) → B
PrintOptimalParens(s,3,3) → C)
PrintOptimalParens(s,4,4) → D))

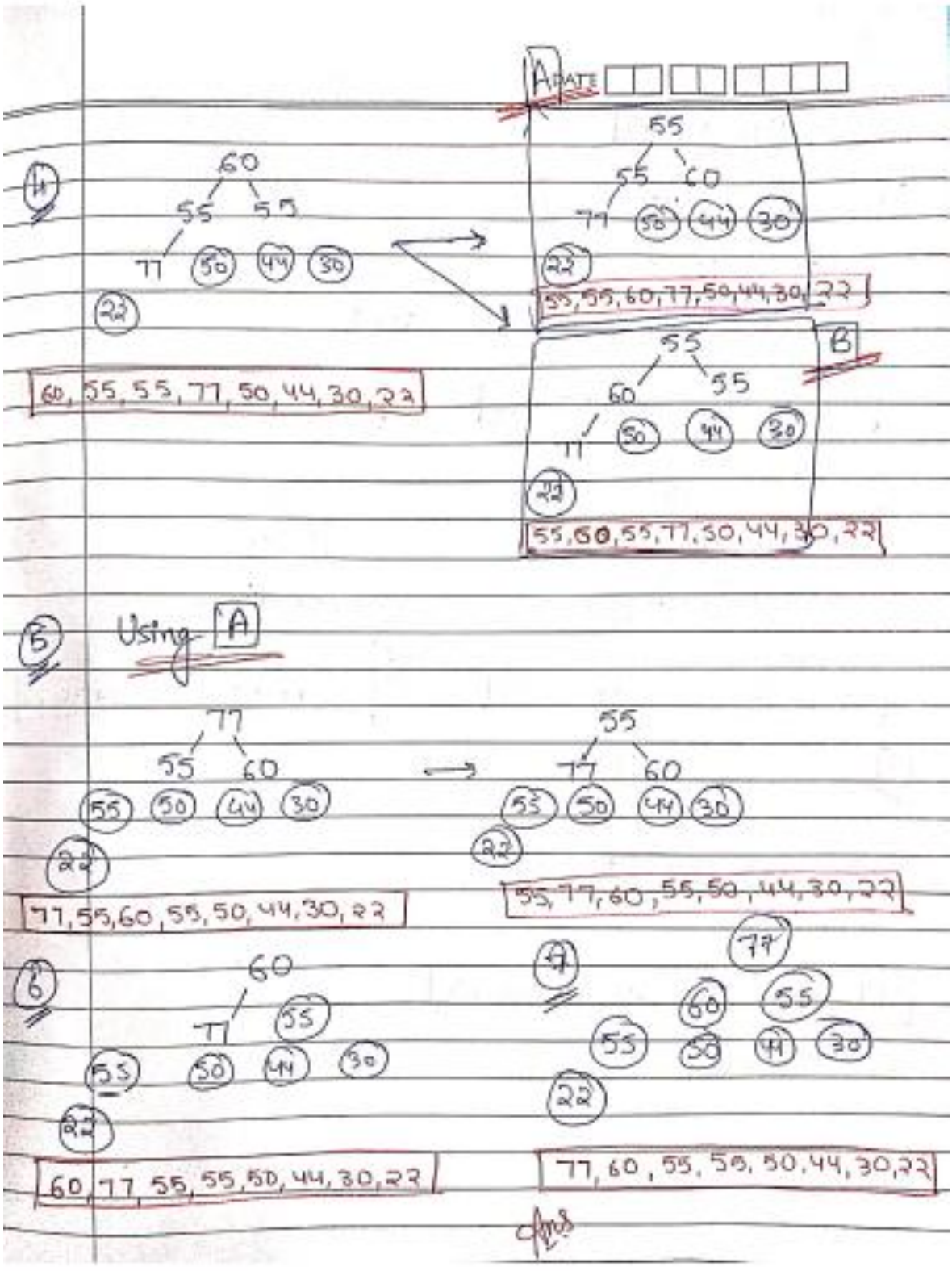
(A((BC)D))

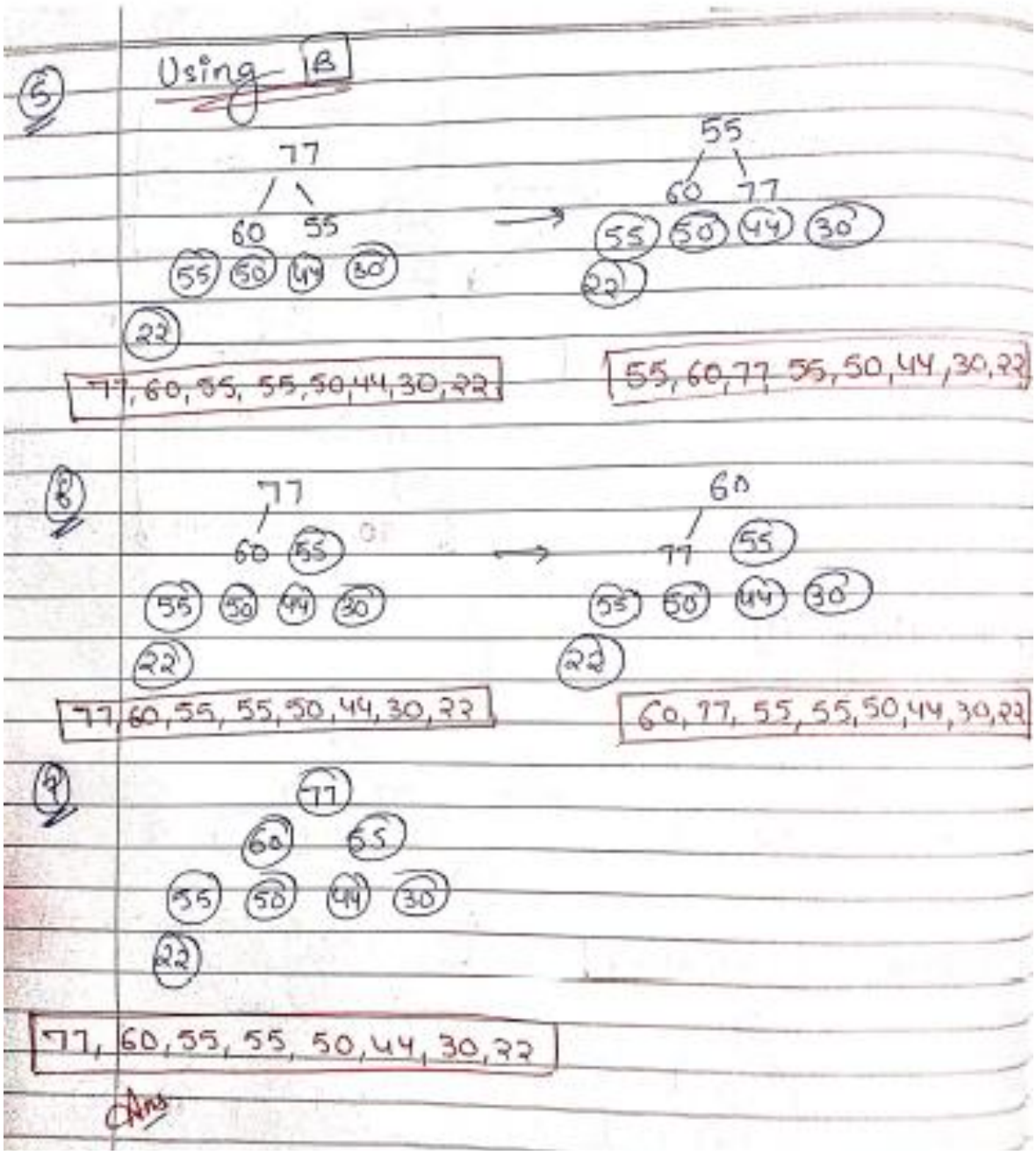
5 × 20 × 10 + 5 × 10 × 5 + 10 × 5 × 5 = 1000 + 250 + 250 = 1500

2 Marks for correct call of PrintOptimalParens() and the final parenthesized answer.
Only 1 Mark for direct parenthesized answer, otherwise ZERO.

Q(Illustrate stepwise execution of the Heapsort algorithm on an array $\boldsymbol{A} = [\mathbf{44}, \mathbf{30}, \mathbf{50}, \mathbf{22}, \mathbf{60}, \mathbf{55}, \mathbf{77}, \mathbf{55}]$ to arrange its elements in descending order. **(10)**



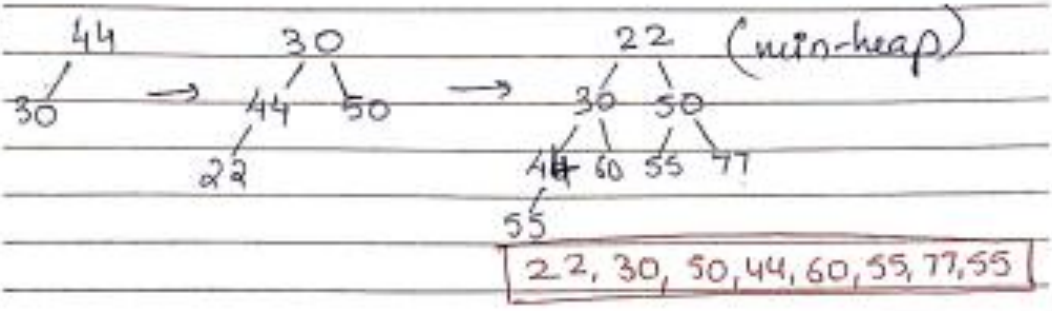

A
DATE
60
55 55
77 50 44 30
22
60, 55, 55, 77, 50, 44, 30, 22
55
55 60
77 50 44 30
22
55, 55, 60, 77, 50, 44, 30, 22
B
55
60 55
77 50 44 30
22
55, 60, 55, 77, 50, 44, 30, 22
Using A
77
55 60
55 50 44 30
22
77, 55, 60, 55, 50, 44, 30, 22
55
77 60
55 50 44 30
22
55, 77, 60, 55, 50, 44, 30, 22
60
77 55
55 50 44 30
22
60, 77, 55, 55, 50, 44, 30, 22
77
60 55
55 50 44 30
22
77, 60, 55, 55, 50, 44, 30, 22


(5) Using MB
77
60 55
55 60 44 30
22
77, 60, 55, 55, 50, 44, 30, 22
55
60 77
55 50 44 30
22
55, 60, 77, 55, 50, 44, 30, 22
(8)
77
60 55
55 50 44 30
22
77, 60, 55, 55, 50, 44, 30, 22
60
77 55
55 50 44 30
22
60, 77, 55, 55, 50, 44, 30, 22
(9)
77
60 55
55 58 44 30
22
77, 60, 55, 55, 50, 44, 30, 22
Ans

APPROACH II.

DATE

A = [44, 30, 50, 22, 60, 55, 77, 55]



22, 30, 50, 44, 60, 55, 77, 55

Steps for heap-sort remain same as min-heap obtained by both the approaches is same.

Marking scheme:

Min-heap creation - 2 Marks

Heapsort - 7 Marks
(1 mark per step)

Max-heap creation - 1½ marks

Heapsort - 3½ marks
(½ marks per step)

Q7. In n-Queens, some solutions are simply reflections of others. For example, two solutions (**Fig. 2**) for 4-Queens problem are equivalent under reflection. **(10)**

(a) Modify NQueens Algorithm (**Fig. 3**) to get unique solutions only.

(b) Execute the NQueens Algorithms devised in **Q7. (a)** for $n = 1, 2, 3,$ and 4. Draw the associated state space search trees separately for each value of n.

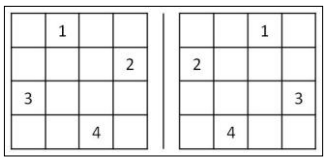


Fig. 2

```
// Algorithm to place queens in a n×n grid.

Algorithm NQueens(k,n)
{  for i = 1 to n do
   {  if (Place(k,i)) then
      {  x[k] = i;
         if (k == n)       print x[1..n];
         else              NQueens(k+1,n);
      }
   }
}


// Returns true if Queen can be placed in k^th row i^th
// column.

Algorithm Place(k,i)
{  for j = 1 to k - 1 do
     if ((x[j] == i) OR (Abs(x[j] - i) == Abs(j - k)))
                  return false;
   return true;
```

Fig. 3

(a) Modified Algorithm

```
#include<iostream>
#include<cmath>
using namespace std;
int x[4]={0};
bool Place(int k,int i)
{
        for (int j=1;j <= (k-1);j++)
        {       if((x[j] == i) || ((x[j]-i)==(k-j)) || ((x[j]-i)==(j-k)))       return false;       }
        return true;
}
void NQueens(int k,int n)
{
        int n1=0;
        /* SOLUTION 1: In case you are placing the 1st Queen, limit the search space to half
for Even n
        In case of Odd n, take ceil of n/2 , For considering extra solution  for odd path (n/2+1)
        for n=1 (i=1), n=2(i=1), n=3(i=1,2), n=4(i=1,2), n=5(i=1,2,3) and so on...                */
        if (k==1)
                n1=ceil(n/2.0);         or   n1=ceil((n+1)/2);
        else
                n1=n;
        for (int i=1;i<=n1;i++)
        {
                if (Place(k,i))
                {       /*SOLUTION 2: */
                //      if(x[1]<=ceil(n/2.0))
                //      {
                        x[k]=i;
                        if(k==n)
                        {       for(int j=1;j<=n;j++)   cout<<x[j]<<" ";      cout<<"\n";       }

                        else
                                NQueens(k+1,n);
                }       /*SOLUTION 3: */
                //                      if((k==1) && i == ceil(n/2.0))
```

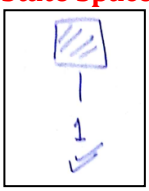
```
                    //                  break;
                    //          }
            }
}
int main()
{           int n=6;
            for(int i=1;i<=n;i++)
            {           cout<<"Final Solution(s) for n = "<<i<<":"<<endl;
                        for(int j=1;j<=n;j++)   x[j]=0;                 NQueens(1,i);                   cout<<endl;
            }
            return 0;
}
```

**** Assuming these below solutions are applied with correct concept and algorithm, then only marks written against Max. Marks Assigned are given, else variations (less marks) are done in marking scheme according to the concept or algorithm part.**

Solution (s)	Examples	Where Wrong??	**Max. Marks Assigned**
*If any of the above correct solution (1, 2 or 3) **With complete algorithm**	Solution 1,2,3 Given Above in Code	CORRECT	**4**
*If any of the above correct solution (1, 2 or 3) Concept is applied to 1st queen and ((n+1)/2) or ceil(n/2) is taken correctly **Without algorithm** **OR** ***If either concept is applied for 1st queen OR ((n+1)/2) or ceil(n/2) is taken **With algorithm**	Solution 1,2,3 Given above Given Above in Code	Algorithm not given, only concept discussed OR Concept is half correct with algorithm	**3**
If either concept is applied for 1st queen OR ((n+1)/2) or ceil(n/2) is taken **Wihout algorithm**	Solution 1,2,3 Given above Given Above in Code	Half Concept Correct And Algorithm missing	**2**
If **-Reverse of answer is checked for uniqueness** **OR** **-Transpose is checked** **With Complete CORRECT Algorithm and Explanation**	Stored solution in 2-D matrix and checked using Check Function()	Not efficient as you are still exploring the complete search space tree and just introducing additional complexity algorithm for checking this.	**3**

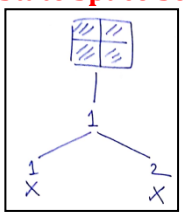
Solution (s)	Examples	Where Wrong??	**Max. Marks Assigned**
If Only First Solution is Obtained. (**Correctly applied with Algorithm)	Break statement; Row-sum-concept; 2+4+1+3=(n+1)=5 3+1+4+2=(n+1)=5 (Don't print)	Applies only to Nqueens where N is <=4, **Not generalized Solution	**2**
Any other modification that does not lead to a solution	Modifying Place(k,i) algorithm	Place(k,i) modification will lead to alterations in placement of N Queens and will be unable to place them at correct locations	**0**

(b)<u>For n=1</u>
State Space Search Tree for N = 3 X 3 for inequivalent Solution is shown below:



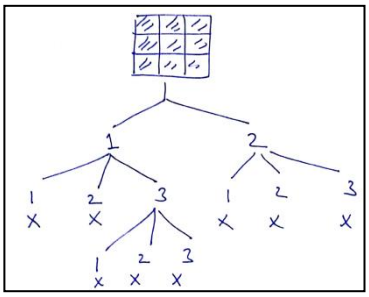
No. of Unique Solutions: **1.**
One can place the 1-Queen at the following position: [1]

<u>For n=2</u>
State Space Search Tree for N = 2 X 2 for inequivalent Solution is shown below:



No. of Unique Solutions: **0 (None).**

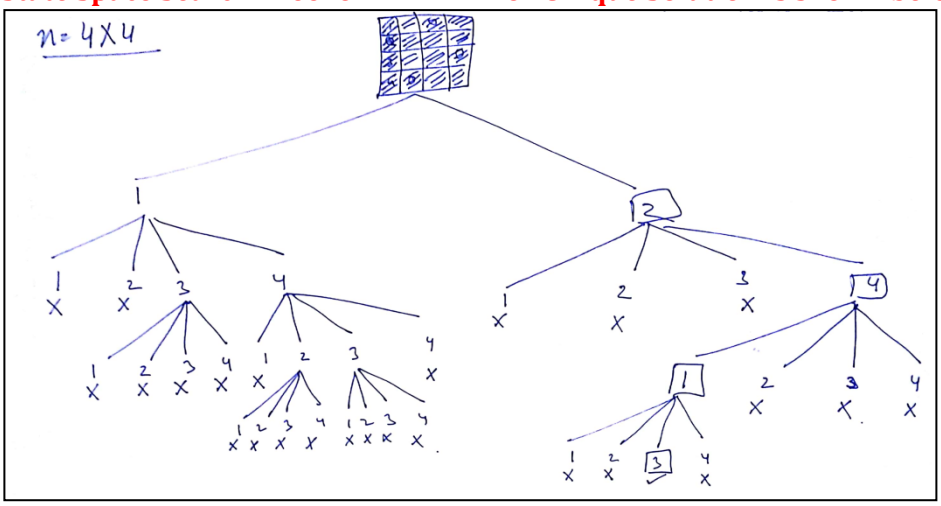
<u>For n=3</u>
State Space Search Tree for N = 3 X 3 for Unique Solution is shown below:



No. of Unique Solutions: **0 (None).**

For n = 4

State Space Search Tree for N = 4 X 4 for Unique Solution is shown below:



No. of Unique Solutions: **1.**

One can place the 4-Queens at the following positions: [2 4 1 3]

MARKING SCHEME

6 Marks.

n = 1 --> 1 Mark.

n = 2 --> 1 Mark. [0.5 Marks if two possibilies are considered to place first queen]

n = 3 --> 2 Mark. [1 Mark if one/three possibilies are considered to place first queen]

n = 4 --> 2 Mark. [1 Mark if one/three/four possibilies are considered to place first queen]

Q8. (a) Run Dijkstra's algorithm on the directed graph (**Fig. 4**), starting at vertex ***S***. Show all the intermediate graphs in deriving the final shortest path tree. What is the order in which vertices get removed from the priority queue? **(5)**

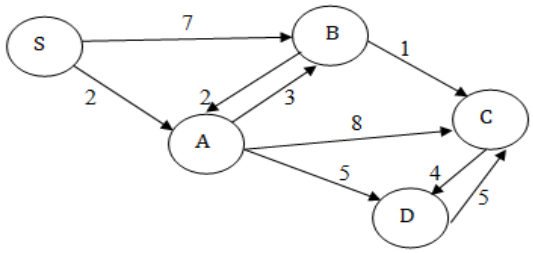
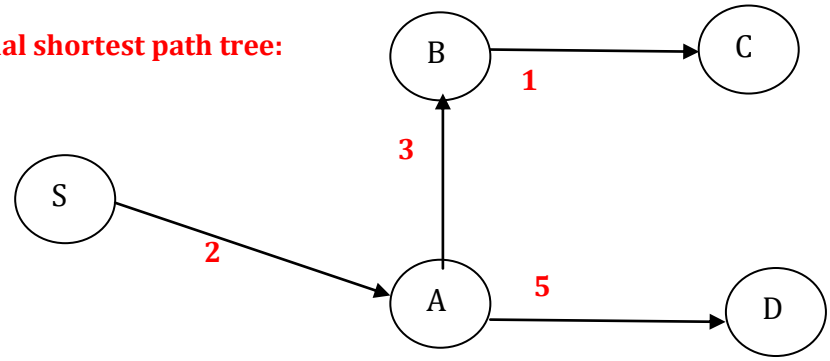


Fig. 4

4 Marks

- **1 mark each for correct path from S --> A, S-->B, S-->C, S-->D if shown with intermediate steps so total 4 marks**
- **0.5 mark deducted for each path if steps are not shown and direct answer is written so total 2 marks.**

The final shortest path tree:



1 mark for telling the order in which vertices are extracted from priority queue.
The order in which the vertices get removed from the priority queue: S, A, B, C, D

Q8. (b) Write sequence in which nodes of the graph (**Fig. 5**) have been traversed using DFS and BFS, starting at vertex ***A***. To make a unique solution, assume that whenever you faced with a decision of which node to pick from a set of nodes, pick the node whose label occurs earliest in the alphabet. **(5)**

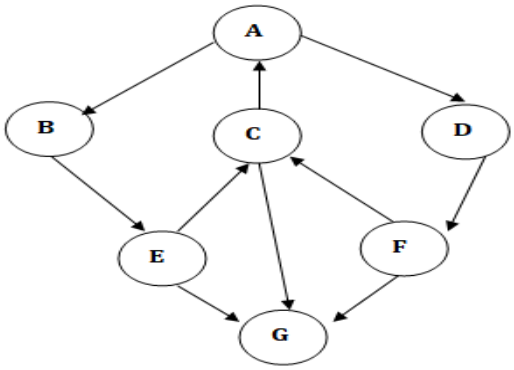


Fig. 5

(b)
BFS: A B D E F C G. (2.5)

DFS: A D F G C B E (2.5)
DFS: A B E C G D F (2.5)
DFS: G C E B F D A (2.5)

First three in sequence: 1 Mark
First five in sequence: 2 Marks

Q9. (a) Let each node x in a binary search tree keeps an attribute $x.successor$ that points to x's inorder successor. Give a pseudocode for **INSERT** on a binary search tree T using this representation. **(6)**

(a) The procedure can also be implemented using a pure iterative approach. However the specified boundary conditions remain same and thus the marking scheme.

Here, z is the new node to be inserted, x is root of the subtree, and y is the parent of x.

```
TREE-INSERT-WITH-SUCCESSOR(y,x,z)
1.  if x ≠ NIL then
2.      if z.key < x.key then          0.5 Marks for correct left subtree selection
3.          TREE-INSERT-WITH-SUCCESSOR(x,x.left,z)
4.      else                           0.5 Marks for correct right subtree selection
5.          TREE-INSERT-WITH-SUCCESSOR(x,x.right,z)
6.      end if
7.  end if
8.  if y == NIL then                   0.5 Marks for checking NULL
9.      T.root = z                     0.5 Marks for creating root
10. else if z.key < y.key then         0.5 Marks for correct 10 and 11 line
11.     y.left = z
12.     z.succ = y                     1 Mark for setting successor of new node
13. else
14.     y.right = z                    0.5 Marks for correct insertion
15.     z.succ = y.succ                1 Mark for setting successor of new node
16.     y.succ = z                     1 Mark for updating successor of parent
17. end if
```

One can call inorder successor function at lines 12, 15, and 16. In that case:

1 mark

- **1 mark: three calls to inorder successor function are present in insert function.**
- **0.5 marks: two or one call to inorder successor function is present in insert function.**
- **0 marks: otherwise**

2 marks are for the code of the inorder successor function.

Q9. (b) Given two integer arrays $\boldsymbol{a}[\mathbf{1}..\boldsymbol{m}]$ and $\boldsymbol{b}[\mathbf{1}..\boldsymbol{n}]$, find an integer that appears in both arrays (or report that no such integer exists). Assuming $\boldsymbol{m} \leq \boldsymbol{n}$, give a crisp and concise algorithm in structured English satisfying following performance requirements: **(6)**

- The amount of extra space (besides $\boldsymbol{a}[\,]$ and $\boldsymbol{b}[\,]$) must be constant. It is fine to modify $\boldsymbol{a}[\,]$ and $\boldsymbol{b}[\,]$.
- The worst case running time must be $\boldsymbol{O}(\boldsymbol{n}\ \boldsymbol{log}\ \boldsymbol{m})$.

Note: Algorithm will be evaluated on correctness, efficiency, and clarity.

(b)

- **6 Marks: heap sort (3)+ binary search (3)**

  Use heap sort on a[ ]
  Worst case running time: O(m log m)
  Then use binary search for each element of b[ ] in array a[ ]
  Worst case running time: O(n log m)
  Total Worst case running time: O(m log m) + O(n log m)
  As $m \leq n$, hence, Worst case running time: O(n log m).

- **3 Marks: merge sort (0)+ binary search (3)**

  Use merge sort on a[ ]
  Worst case running time: O(m log m)
  Then use binary search for each element of b[ ] in a[ ]
  Worst case running time: O(n log m)
  Total Worst case running time: O(m log m) + O(n log m)
  As $m \leq n$, hence, Worst case running time: O(n log m).
  But merge sort is not applicable because of constant extra space which indicates one should go for in place sorting mechanism.

- **2 Marks:**

  Apply heap sort on b[ ] then binary search for each element of a[ ].
  Total Worst case running time: O(n log n) + O(n log m)
  As $m \leq n$, hence, Worst case running time: O(n log n).

- **2 Marks:**

  Any other sorting technique with good explanation + binary search

- **0 Marks:**

  Otherwise

- **0 Marks:**

  Binary search without a proper sorting mechanism.

Q10. (a) **Fig. 6** presents an algorithm to find the minimum in a stack of integers without affecting its contents. Determine if there is any bug in this code with respect to the following. Give suitable examples for each justification. **(6)**

i. Change in stack contents.

ii. Incorrect answer.

```
// S and temp are stacks. S contains some integers whose minimum is to be
// determined. isEmpty(S) returns true if S is empty, else returns false.
// min and t are integer variables.
       if (!isEmpty(S))
       {  min = pop(S);
          while (!isEmpty(S))
          {  t = pop(S);
             if (t < min)
                 min = t;
             push(temp, t);
          }
          while (!isEmpty(temp))
             push(S,pop(temp);
          return min;
       }
```

Fig. 6

(a) Let S = [5 3 1 8 9]

S	temp
5	
3	
1	
8	
9	

min **t**

S	temp
3	
1	
8	
9	

min 5 **t**

S	temp
1	
8	
9	3

min 3 **t** 3

S	temp
8	1
9	3

min 1 **t** 1

S	temp
	8
	1
9	3

min 1 **t** 8

S	temp
	9
	8
	1
	3

min 1 **t** 9

S	temp
3	
1	
8	
9	

Final contents of S = [3 1 8 9] and minimum value returned is min = 1.

i. Yes, the contents will change.

1 Mark for YES.
2 Marks for correct justification, else 1 or 1.5 Marks depending on the type of explanation given.

ii. No, always correct minimum value will be returned.

1 Mark for NO.
2 Marks for correct justification, else 1 or 1.5 Marks depending on the type of explanation given.

(b) Consider a recursive static function $f()$, given below to answer the following questions **(6)**

```
static int f(int n)
{ if (n == 0) return 0;
  if (n == 1) return 0;
  if (n == 2) return 1;
  return f(n - 1) + f(n - 2) - f(n - 3);
}
```

i. What is the value of $f(3)$? **1** **(1 Mark)**
ii. What is the value of $f(4)$? **2** **(1 Mark)**
iii. What is the value of $f(7)$? **3** **(1 Mark)**
iv. How many calls on $f()$ are made to compute $f(7)$, including the first one? **46** **(1.5 Mark)**
45 or 47 **(1 Mark)**
v. What is the value of $f(101)$? **50** **(1.5 Mark)**

----------**ALL THE BEST**----------