B-TECH 3RD SEMESTER FINAL EXAMINATION

JANUARY 2021 [CST]

SUBJECT: DATA STRUCTURES [CS 2103]

Date of Examination: 12/01/2021

Name Abhiroop Mukherjee

Enrolment Number: 510519109

Previous Enrolment Number: 510719007

G-Suite ID: 510519109.abhirup@students.iiests.ac.in

No. of Sheets uploaded: 16

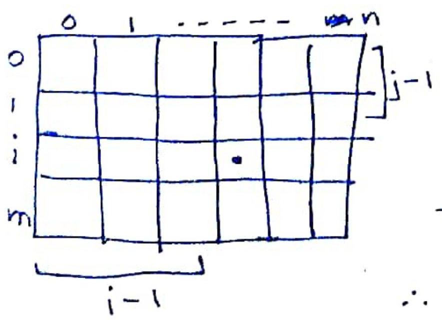
Q.1) a) ~~AB~~ Abstract Data Type [ADT] is functional definition of the data structure, it specifies, what the data structure should do, but is ~~indep~~ independant of it's implementation

Eg: Stack is an ADT with push() and pop() ~~for~~ functions

d) Advantages of Linked List:

i) Deletion ~~is~~ of element is fast [$O(1)$]

ii) We can ~~make~~ do dynamic allocation with Linked List, saving space in memory

e) 

given $A[0,0] = \alpha$

to find $A[i,j]$

→ assuming row major.

$\therefore A[i,j] = \alpha + W[(j-1)n + (i-1)]$

where W = sizeof datatype of array

f) Although Binary Search is faster than Linear Search [BS: $O(\log n)$, LS: $O(n)$], Binary Search can't work if we don't have following things

i) ~~Sorted & Array~~ Sorted Data.

ii) Easy to Access Data.

∴ → if we have unsorted data, we should prefer Linear Search ~~over~~

→ if we are using Linked List [it is harder to access middle element], we should prefer Linear Search.

[although Binary Search for Linked List is ~~possib~~ possible].

g) Component Sum hash code map is method where all the components of a data type [string for ~~exam~~ example] are summed to create a integer input for compression map,

→ The problem arise in the following example

abcd.
dacb
bacd
cbad
⋮

→ all have same component sum and will have lot of collision in a single key, hampering performance.

Scanned with CamScanner

2) a) I) Link List Based Queue

Advantages

→ all the advantage of Linked List

→ Dynamic Allocation of memory.

~~→ Easier deletion fro~~

→ Easier Deletion of incorrectly fed ~~value~~ data

Disadvantage

→ Takes up more space than similarly sized array queue due to the fact that ~~to~~ linked List also have to store the addresses

→ Due to extra steps of link arrangements, this is slower than Array based Implementation

II) Array Based Queue.

Advantage

→ Faster as there is no link ~~arra~~ rearrangements

Disadvantage

→ Fixed Size

→ Harder deletion of incorrectly fed data.

2) b) structure

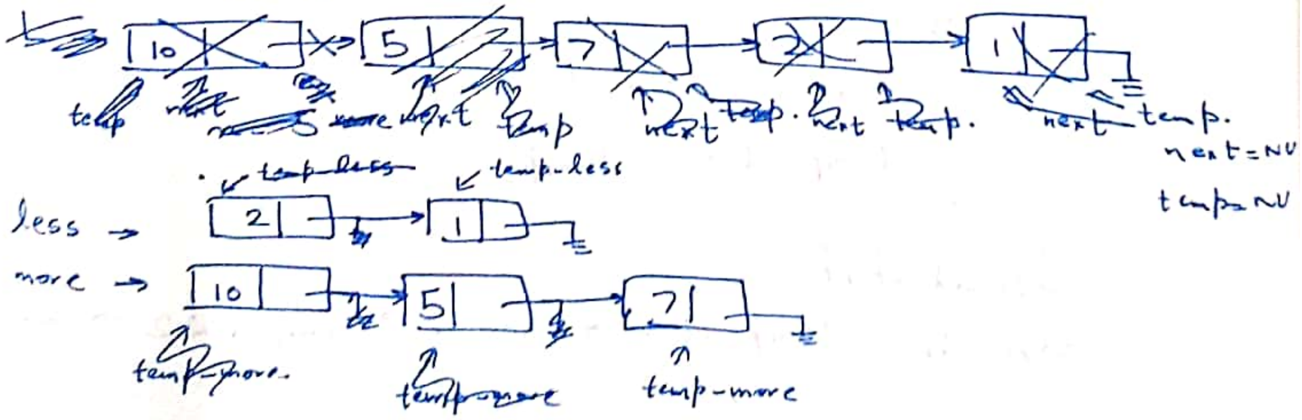
```
struct   stack_max
{ int   max ;
  int   data[100];
  int   top;
}

typedef struct stack_max  stack ;

void  print_max (stack  s)
{
  printf ("%d \n" , s. max );
}

bool  push ( stack  s,  int  value )
{
    if (s.top ==100)
    {    printf (" Stack Full");
         return false;
    }
    if(value > s.max)
        s.max = value;
    s.top= s.top + 1;
    s. data[s. top]= value ;

    return true;
}
```

```
int pop (stack s)
{ if (s.top == 0)
  {  printf ("Already Empty");
     return -1;     //error
  }
  int value = s.data[s.top];
  s.top --;
  return value;
}


int main ()
{
   stack s;  s.max = 0;  s.top = 0;

   bool test = push (s, 5);
   test = push (s, 10);
   print_max (s);
   int value = pop (s);
}  return 0;
```

```
3) a) node* ~~void~~ rearrange ( node* L, int n)
                                    // node* have value & next
{
    node* less = NULL; node* temp-less = NULL;
    node* more = NULL; node* temp-more = NULL;

    node* temp = L ;
    node* next = L ;
      L=NULL ; // avoid dangling pointer
    while ( ~~next~~ temp != NULL)
    {
        next = next -> next ;

        if(temp -> value >=n)
        {
            if ( more == NULL)
            {
                ~~temp~~ more = temp ;
                    temp-more = more ;
                    ~~more -> next~~
                    temp-more -> next=NULL;

            }
          else
            {
               temp-~~>~~more -> next= temp;
              temp-more = temp;
              temp-more -> next= NULL;
            }
        } // end of if
        else {

        (next pg.)
```

```
        if (less == NULL)
        {
            less = temp;
            temp_less = less;
            temp_less -> next = NULL;
        }
        else
        {
            temp_less -> next = temp;
            temp_less = temp;
            temp_less -> next = NULL;
        }
    } //end of else

    temp = next;
}//end of while loop.

temp_less -> next = more;

return less;
} //end of function.
```

Dry Run $n = 5$

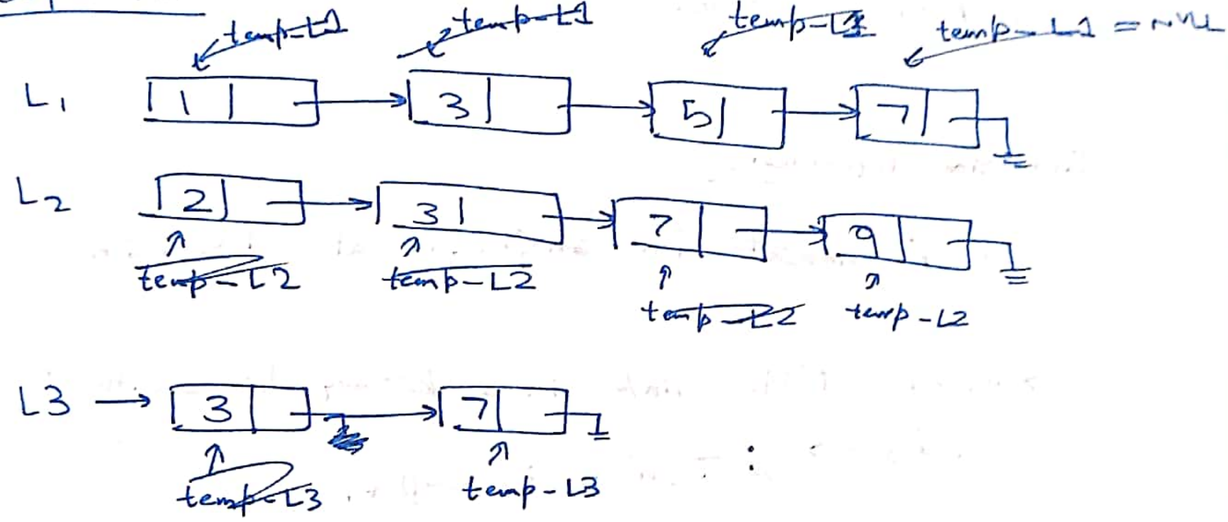


3) b) node* intersection (node* L1, node* L2)

```
{
    // L1 and L2 sorted in increasing order.
    // node have value & next
    ~~int~~ node* temp_L1 = L1;
    node* temp_L2 = L2;

    node* ~~temp~~ L3 = NULL;
    ~~while~~ node* temp_L3 = NULL;

    while ( temp_L1 != NULL && temp_L2 != NULL)
    {
        if (temp L1 -> value == temp L2 -> value)
        {
            ~~if (L3~~
            node* temp = (node*) malloc (sizeof (node));
            temp -> value = temp L1 -> value;
            temp -> next = NULL;
            temp_L1 = temp_L1 -> next;
            temp_L2 = temp_L2 -> next;
            if (L3 == NULL)
            {   L3 = temp;
                temp_L3 = L3;
            }
            else {
                temp_L3 -> next = temp;
                temp_L3 = temp;
            }
        } // end of if
        else if (temp_L1 -> value < temp_L2 -> value)
        {
            temp_L1 = temp_L1 -> next ;
        }
```

```
else
{
    temp-L2 = temp-L2 -> next;
}
}//end of while

return L3;
}//end of function.
```

Dry Run



4) a) T.P in a k-ary tree, with n nodes no. of NULL links $\Rightarrow n(k-1)+1$

→ <u>Base Case</u>

$n=1$

NULL links $= k = 1(k-1)+1$

$\therefore n=1$ is true.

<u>Induction Hypothesis</u>

let ~~no. of~~ for ~~all i ≤ n~~ all $1 \le i \le n$

→ no. of NULL links in a k-ary tree with i nodes → $T(n) = i(k-1)+1$

→ we know that a tree with i nodes have $i-1$ ~~total~~ non-NULL links ~~and ik~~ ~~total links~~

$\therefore$ for i nodes → $i(k-1)+1$ null links
→ $i-1$ non-null link
→ ik total links

Induction Step

→ Let a node be added in the i-node tree so following things happen.

→ ~~Total links = (i+1) k~~

→ ~~non-null links = i - 1 + 1 = i~~

∴ ~~Not~~

~~Null Links = Total - (non-null) = (i+1) k - i~~

→ Null links ⇒ $T(n+1)$

→ now addition of a node will remove a null link; but introduce more k null links.

$$\therefore T(n+1) = T(n) - 1 + k$$

$$= n(k-1) + 1 - 1 + k$$

$$= n(k-1) + 1 + (k-1)$$

$$= (n+1)(k-1) + 1$$

∴ ~~P...~~

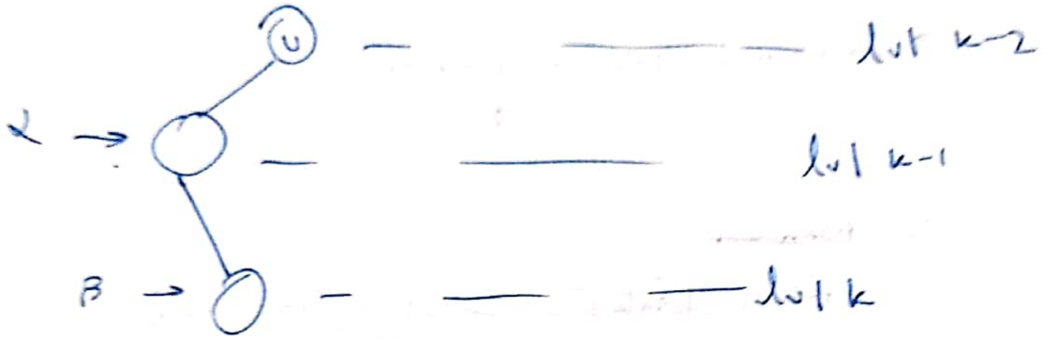
→ If we assume $T(n)$, we can prove $T(n+1)$

→ $T(1)$ is true, so $T(2)$ is true, so $T(3)$ is true and so on and so forth

∴ $T(n)$ is proved.

4) b) Let us prove this using contradiction.

Let us ~~assume~~ suppose that there is a non-leaf node at level k−2 with children < 2, i.e 1 children in an AVL Tree



→ node β is a leaf node, it has height 1

→ node α ~~is~~ has height 2, and is height balanced.

→ u is not height balanced as it's lchild have height 2 and ~~rc~~ rchild have height 0

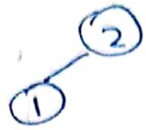
∴ The tree is not AVL, which contradicts the fact that we assumed the the tree is AVL

∴ What we ~~as~~ assumed was wrong

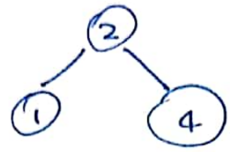
∴ If the closest leaf in an AVL tree is at level k, then all the levels from 1 to k−1 has maximum possible no. of nodes.

4c) I) 2

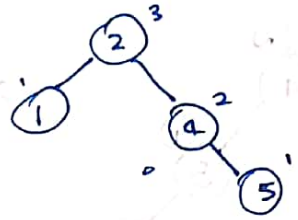
II) 1



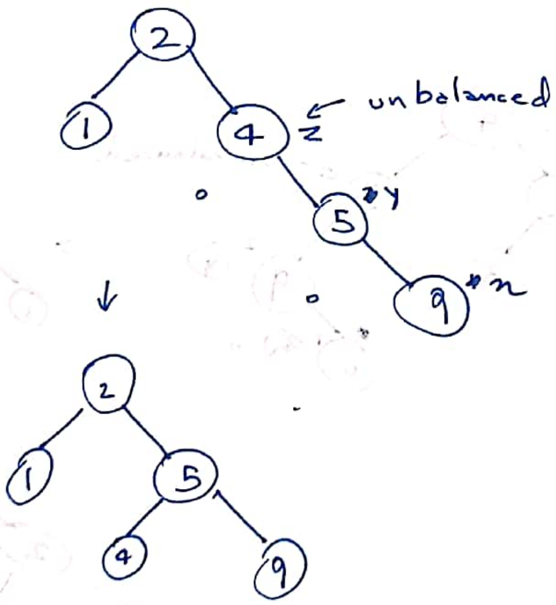
III) 4



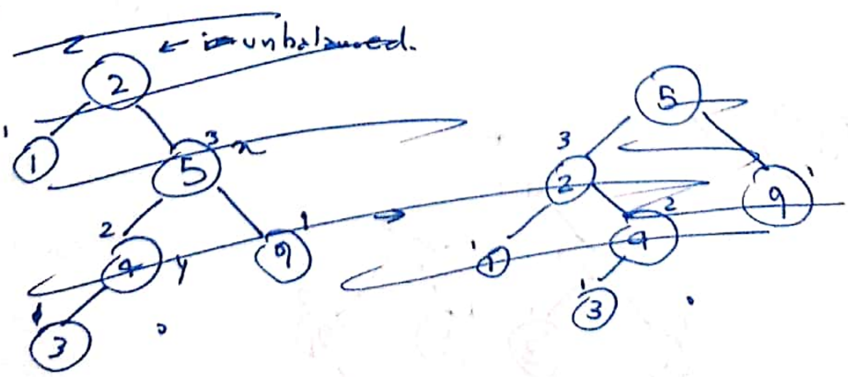
IV) 5



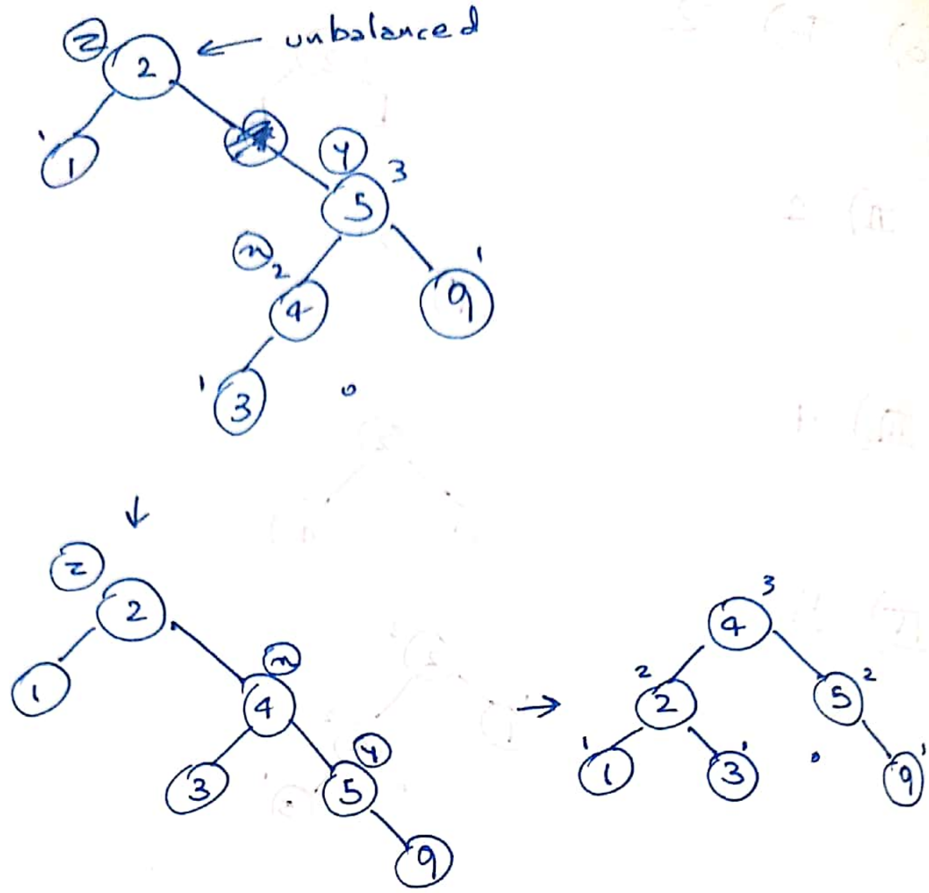
V) 9



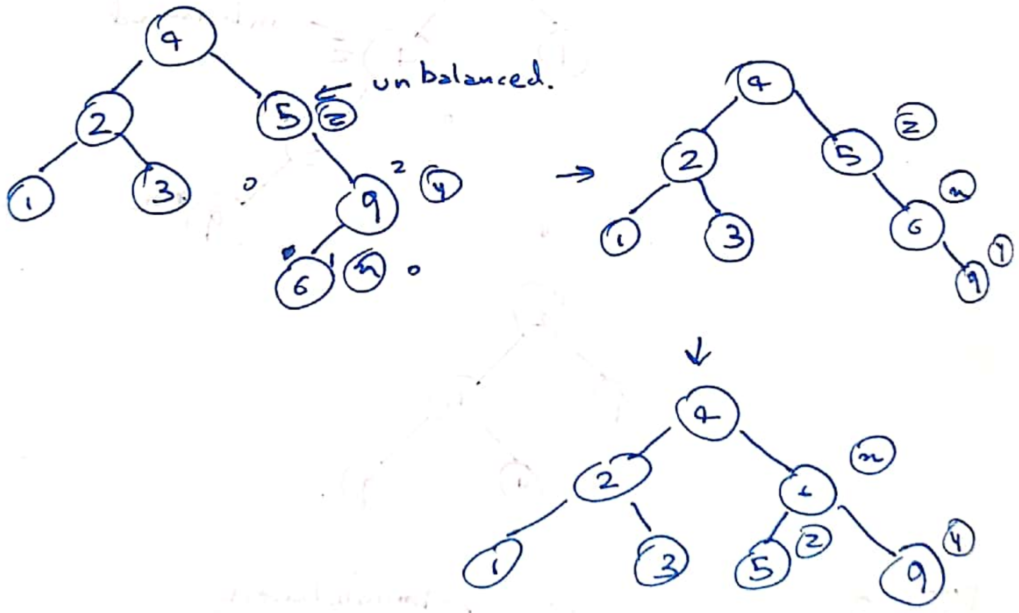
~~VI) 3~~



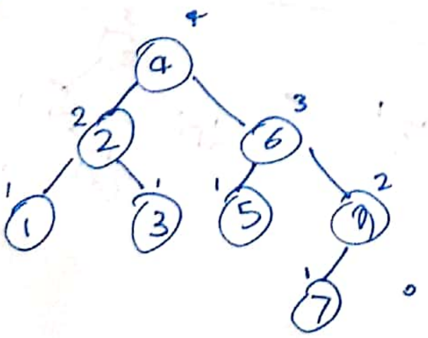
VI) 3



VII) 6



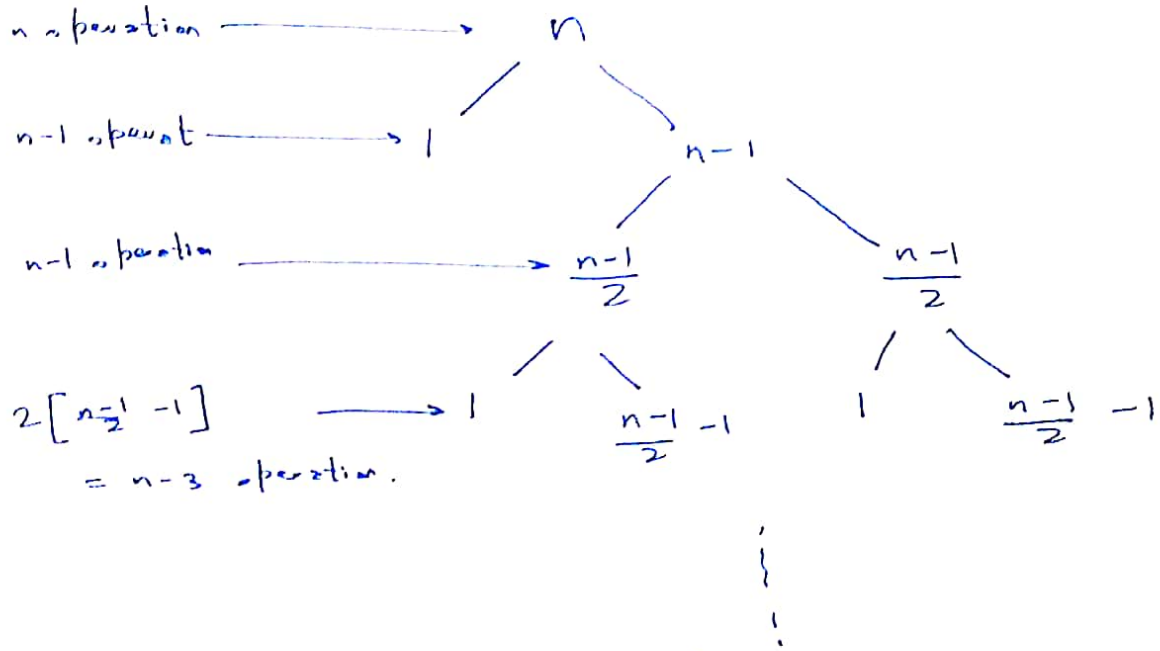
VIII) 7



5) a) i) as data is almost sorted, quicksort will perform poorly, Here we can use bubble ~~iii)~~ sort, as it works fast on almost ~~order sor~~ sorted data.

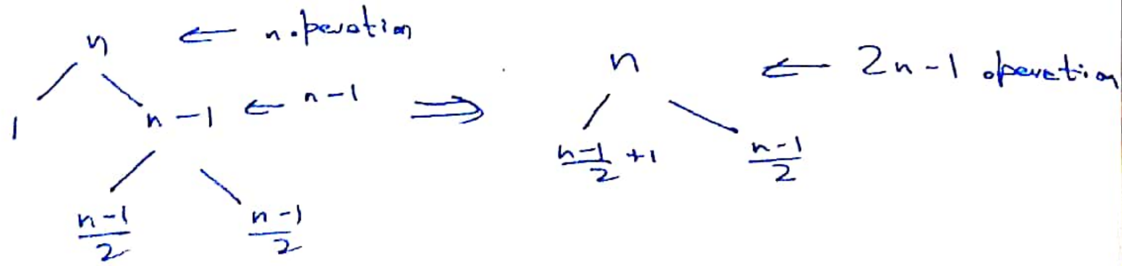
ii) as all the data are random, quicksort will work in $O(n\log n)$, so quicksort will be applicable.

b) Alternate Lucky and unlucky



→ This will be the order of partitions.

→ also we can do this.



∴ It's Recursion will be

$$L(n) = 2U\left(\frac{n}{2}\right) + \Theta(n)$$
$$U(n) = L(n-1) + \Theta(n)$$

where $U \rightarrow$ unlucky case

$L \rightarrow$ Lucky case.

$$\therefore \quad L(n) = 2\left[L\left(\frac{n}{2}-1\right) + \Theta\left(\frac{n}{2}\right)\right] + \Theta(n)$$
$$= 2L\left(\frac{n}{2}-1\right) + 2\Theta\left(\frac{n}{2}\right) + \Theta(n)$$

which simplifies to

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

∴ ~~[illegible]~~ ∴ Hence Proved.