DATA STRUCTURES USING C

NOTES

Prepared by

Prof. Prabhu Kichadi, M. Tech
9880437187

UNIT – II

CONTENTS

Recursion: Definition: Types of recursions; Recursion Technique Examples – GCD, Binomial coefficient, nCr, Towers of Hanoi, Comparison between iterative and recursive functions.
Arrays as Abstract Data Types, Representation of linear arrays in m/o, traversing linear arrays, inserting and deleting elements,
Sorting: Selection sort, bubble sort, quick sort, merge sort, insertion sort.
Searching: sequential search, binary search, iterative and recursive searching.

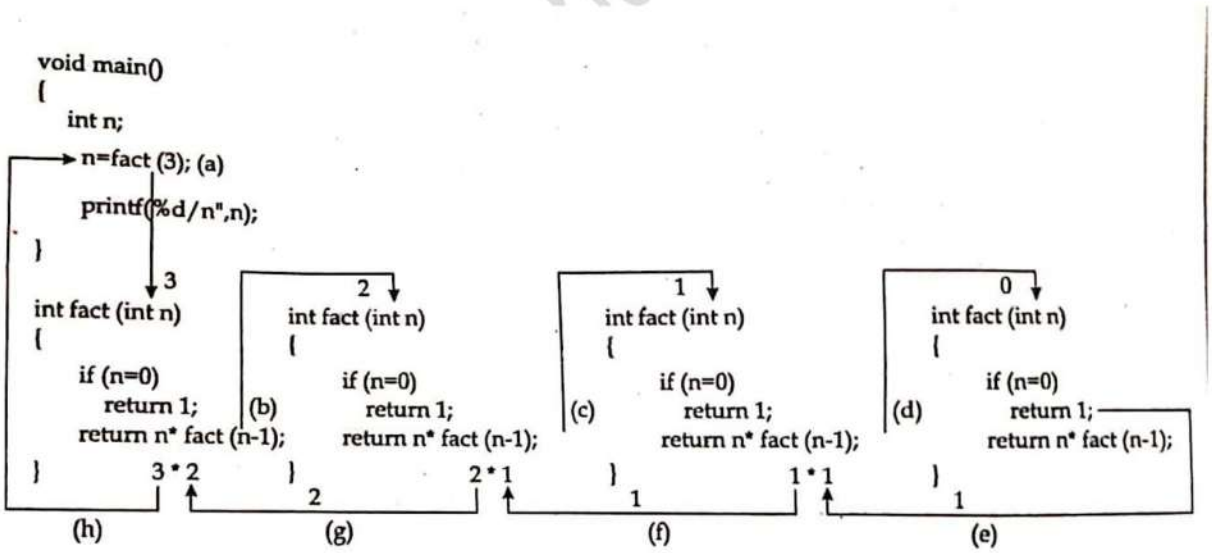
Recursion: *Recursion is the mechanism in which a function calls itself & the function is called recursive function.*

- Recursive function must have at least one termination condition.
- When each time function is called, a new set of local variables & formal parameters are allocated on the stack and execution starts from the beginning.
- The call to itself is repeated till a base-condition is reached.
- Once a base condition is reached, the function returns values to the previous copy of the function.

Ex: Finding factorial of a number.

```
int fact(int n)
{
        if(n==1)
                return 1;
        else
                return n * fact(n-1);
}
```

Below fig depicts the flow of calls and values returned by the function.



Program

```
#include<stdio.h>
int fact(int n);
int main()
{
```

```
    int n,res;

    printf("\nEnter a Number : ");
    scanf("%d", &n);

    res = fact(n);

    printf("\nFactorial = %d",res);

    getch();
    return 0;
}

int fact(int n)
{
    if(n==1)
    {
        return 1;
    }
    else
    {
        return n * fact(n-1);
    }
}
```

Output

```
Enter a Number : 5
Factorial = 120
```

Fibonacci Series

Fibonacci numbers are a series of numbers such that each number is the sum of two previous numbers.
Ex: 0 1 1 2 3 5 8

Base Condition:
If n==0 return 0
If n==1 return 1

General condition:
if n>2 return fib(n-1) + fib(n-2)

Program:

```
#include<stdio.h>
int fib(int n);
int main()
{
   int n, i;

   printf("\nEnter Number of Fibonacci Terms : ");
   scanf("%d", &n);

   printf("\nFibonacci Terms: ");
   for(i=0; i<=n-1; i++)
   {
      printf("%d\t", fib(i));
   }
   getch();
   return 0;
}

int fib(int n)
{
   if(n==0)
   {
      return 0;
   }
   if(n==1)
   {
      return 1;
   }
   else
   {
      return fib(n-1)+fib(n-2);
   }
}
```

Output

```
Enter Number of Fibonacci Terms: 5

Fibonacci Terms: 0      1      1      2      3
```

Types of recursions:

1. **Direct Recursion**
2. **Indirect Recursion**
3. **Tail Recursion**
4. **No Tail/ Head Recursion**
5. **Linear recursion**
6. **Tree Recursion**

1. **Direct Recursion:** When a function calls itself within the same function repeatedly, it is called the direct recursion.

Example:

```
void fun()
{
      // write some code
      fun();
      // some code
}
```

2. **Indirect Recursion:** When a function is mutually called by another function in a circular manner, the function is called an indirect recursion function.

Example:

```
fun1()
{
      //  write some code
      fun2();
}

fun2()
{
      // write some code
      fun3() ;
      // write some code
}
```

```
fun3()
{
      // write some code
      fun1() ;
}
```

3. Tail Recursion: A recursive function is called the tail-recursive if the function makes recursive calling itself, and that recursive call is the last statement executes by the function. After that, there is no function or statement is left to call the recursive function.

Example:

```
#include <stdio.h>
// function definition
void fun1( int num)
{
   // if block check the condition
   if (num == 0)
      return;
   else
      printf ("\n Number is: %d", num); // print the number
   return fun1 (num - 1);       // recursive call at the end in the fun()
function
}

int main ()
{
   fun1(7); // pass 7 as integer argument
   return 0;
}
```

Output:

4. No Tail/ Head Recursion: A function is called the non-tail or head recursive if a function makes a recursive call itself, the recursive call will be the first statement in the function. It means there should be no statement or operation is called before the recursive calls.

Example:

```
#include <stdio.h>
void head_fun (int num)
{
	if ( num > 0 )
	{
	// Here the head_fun() is the first statement to be called
		head_fun (num -1);
	printf (" %d", num);
	}
}

int main ()
{
	int a = 5;
	printf (" Use of Non-Tail/Head Recursive function \n");
	head_fun (a); // function calling
	return 0;
}
```

Output:

5. Linear recursion:

6. Tree Recursion

Recursion Technique Examples:

GCD:

Problem

Find the greatest common divisor (GCD) for the given two numbers by using the recursive function in C programming language.

Solution

The solution to find the greatest common divisor (GCD) for the given two numbers by using the recursive function is as follows –

Algorithm

Refer an algorithm given below to find the greatest common divisor (GCD) for the given two numbers by using the recursive function.

Step 1 – Define the recursive function.

Step 2 – Read the two integers a and b.

Step 3 – Call recursive function.

a. if a>b

b. then return the function with parameters a, b

c. if a ==0

d. then return b

e. else return the function with parameters a, b%a

Program:

```
#include<stdio.h>
int gcd(int m, int n);
int main()
{
        int a,b,res;

        printf("\nEnter Two Numbers : ");
```

```
        scanf("%d%d", &a, &b);

        res = gcd(a,b);

        printf("\n\nGCD(%d, %d) = %d",a,b,res);


         return 0;
}

int gcd(int m, int n)
{
        if(n == 0)
        {
            return m;
        }
        else if(n>m)
        {
           return gcd(m, n%m);
        }
        else
        {
            return gcd(n, m%n);
        }
}
```

Output:

```
Enter Two Numbers : 10 20
GCD(10, 20) = 10
```

Binomial coefficient:

The Problem

Write a function that takes two parameters n and k and returns the value of Binomial Coefficient C(n, k). For example, your function should return 6 for n = 4 and k = 2, and it should return **10** for **n = 5** and **k = 2**.

Optimal Substructure

The value of C(n, k) can be recursively calculated using the following standard formula for Binomial Coefficients.

C(n, k) = C(n-1, k-1) + C(n-1, k)

C(n, 0) = C(n, n) = 1

Following is a simple recursive implementation that simply follows the recursive structure mentioned above.

Implementation:

```
#include<stdio.h>
int bc(int n, int k);
int main()
{
   int a,b,res;

   printf("\nEnter Two Numbers : ");
   scanf("%d%d", &a, &b);

   res = bc(a,b);

   printf("\n\nBC(%d, %d) = %d",a,b,res);


   return 0;
}
int bc(int n, int k)
{
   if(k==0 || k==n)
   {
      return 1;
   }
   else
   {
      return bc(n-1, k-1)+bc(n-1, k);
   }
}
```

Output:

```
Enter Two Numbers : 5 2
```

BC(5, 2) = 10

nCr:

Approach: The idea is simply based on the below formula.
$^{N}C_{r} = N! / (r! * (N-r)!)$

Implementation Code:

```
#include<stdio.h>
int findFact(int n);

int main()
{
   int n,r,res;

   printf("\nEnter N Value : ");
   scanf("%d", &n);

   printf("\nEnter R Value : ");
   scanf("%d", &r);

   res = findFact(n)/(findFact(r)*findFact(n-r));
   printf("\nNCR(%d, %d) = %d",n,r,res);

   return 0;
}
int findFact(int n)
{
   if(n==1)
   {
      return 1;
   }
   else
   {
      return n * findFact(n-1);
   }

}
```

Output:

Enter N Value : 5

Enter R Value : 3

NCR(5, 3) = 10

Towers of Hanoi:

In this problem, there are three needles(stands) A, B & C. There are n discs of different diameters in the needle A and are placed one above the other such that always a smaller disc is placed above the larger disc.

The two needles B & C are empty. All the disks from needle A are to be placed on needle C using B as auxiliary(temporary) needle.

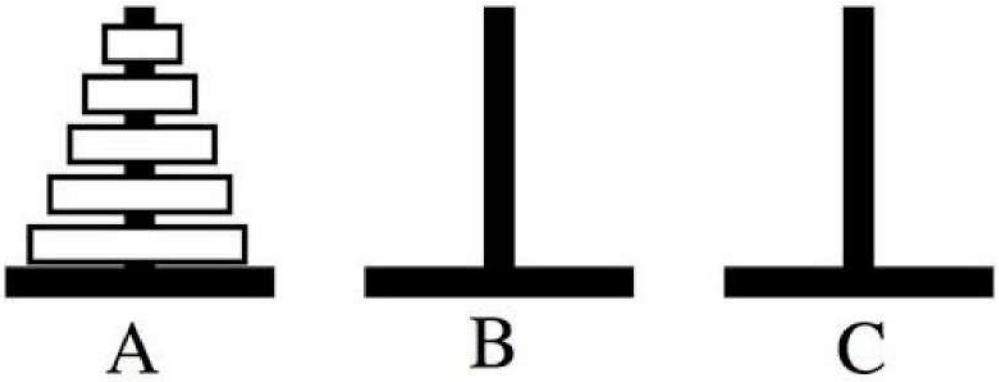
- Only one disk is moved at a time from one needle to another needle.
- Smaller disc is on top of the larger disc at any time.
- Only one needle can be used for storing intermediate disks.

Base Case: This case occurs when there is only one disk. In this case the disk from A to C so,

General Case: If more than two disks to be transferred from source to destination.
Using below steps,

- Move n-1 disk recursively from source to temp.
- Move nth from source to destination
- Move n-1 disc from temp to destination.

we can break the problem down into a few moves. First, we need to move n-1 discs to the spare disc. Then, we are free to move the largest disc (nth disc) to the destination peg. Once we have the largest disc on the destination peg, we need to move the original n-1 discs (now on the spare peg) to the destination peg. With this logic, we can define a function that looks like the algorithm in the figure below.



Program

```
#include<stdio.h>
void towers(int n, char src, char aux, char dest);
int main()
{
   int n;
   char src='A',dest='C',aux='B';

   printf("\nEnter Number of Disks : ");
   scanf("%d", &n);

   towers(n,src,aux, dest);
}

void towers(int n, char src, char aux, char dest)
{
   if(n == 1)
   {
      printf("\nMove Disk %d from %c to %c",n,src,dest);
      return;
   }
   towers(n-1, src, dest,aux);
   printf("\nMove Disk %d from %c to %c",n,src,aux);
   towers(n-1, aux, src,dest);
}
```

Output

```
Enter Number of Disks : 3
```

Move Disk 1 from A to C
Move Disk 2 from A to C
Move Disk 1 from C to B
Move Disk 3 from A to B
Move Disk 1 from B to A
Move Disk 2 from B to A
Move Disk 1 from A to C

Comparison between iterative and recursive functions:

The Recursion and Iteration both repeatedly execute the set of instructions. Recursion is when a statement in a function calls itself repeatedly. The iteration is when a loop repeatedly executes until the controlling condition becomes false.

The primary difference between recursion and iteration is that recursion is a process, always applied to a function and iteration is applied to the set of instructions which we want to get repeatedly executed.

Recursion:

- Recursion uses **selection structure**(function calls).
- **Infinite recursion** occurs if the recursion step does not reduce the problem in a manner that converges on some condition (**base case**) and Infinite recursion can crash the system.
- Recursion terminates when a **base case** is recognized.
- Recursion is usually **slower than iteration** due to the overhead of maintaining the stack.
- Recursion uses **more memory than iteration**.
- Recursion makes the **code smaller**.

Iteration:

- Iteration uses **repetition structure(looping statements)**.

- An infinite loop occurs with iteration if the loop condition test never becomes false and Infinite looping uses CPU cycles repeatedly.
- An iteration **terminates** when the **loop condition fails**.
- An iteration does not use the **stack** so it's **faster than recursion**.
- Iteration consumes **less memory.**
- Iteration makes the **code longer**.

Iteration	Recursion
Iteration explicitly user a repetition structure.	Recursion achieves repetition through repeated function calls.
Iteration terminates when the loop continuation.	Recursion terminates when a base case is recognized.
Iteration keeps modifying the counter until the loop continuation condition fails.	Recursion keeps producing simple versions of the original problem until the base case is reached.
Iteration normally occurs within a loop so the extra memory assigned is omitted.	Recursion causes another copy of the function and hence a considerable memory space's occupied.
It reduces the processor's operating time.	It increases the processor's operating time.

Arrays as Abstract Data Types:

Representation of linear arrays in m/o:

traversing linear arrays:

Inserting and deleting elements:

Sorting:

Sorting is the process of rearranging the data items in some order like ascending or descending.

Ex: telephone directory, names are sorted alphabetically.

Sorting types and techniques:

1. Bubble sort.
2. Selection sort
3. Insertion sort
4. Merge sort
5. Quick sort

1. BUBBLE SORT:

Bubble sort is the simplest sorting technique, in this technique two successive elements a[j] & a[j+1] are swapped repeatedly when compared a[j] >= a[j+1].

Procedure:

For example, consider the elements shown below,
40, 50, 30, 20, 10

The elements can be sorted as shown in following fig. In the first pass 40 is compared with 50 and they are in order, so no exchange is required, next 50 is compared with 30 and they are exchanged since 50 is greater than 30. In the same if we complete 1 pass, one larger element sinks to the bottom of the array.

	1st pass				2nd pass			3rd pass		4th pass
A[0]=40	40	40	40	40	30	30	30	20	20	10
A[1]=50	50	30	30	30	40	20	20	30	10	20
A[2]=30	30	50	20	20	20	40	10	10	30	30
A[3]=20	20	20	50	10	10	10	40	40	40	40
A[4]=10	10	10	10	50	50	50	50	50	50	50

Base:
1. if n elements mean n-1 passes required.
2. two successive elements must be exchanged based on a[j]>a[j+1].

Bubble sort program.

```
#include<stdio.h>
int main()
{

   int a[100],n,i,temp,j;

   printf("\nEnter Number of Elements : ");
   scanf("%d", &n);

   for(i=0; i<n; i++)
   {
      printf("\nEnter an Element : ");
      scanf("%d", &a[i]);
   }
```

```
    printf("\nBefore Sorting\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t",a[i]);
    }
    for(i=0; i<n-1; i++)
    {
        for(j=0; j<n-1; j++)
        {
            if(a[j]<a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    printf("\nAfter Bubble Sorting\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t",a[i]);
    }

    return 0;
}
```

Output

```
Enter Number of Elements : 5
Enter Element : 40
Enter Element : 50
Enter Element : 30
Enter Element : 20
Enter Element : 10
Before  Sort    : 40   50     30     20     10
After Bubble Sort: 10   20     30     40     50
```

Advantages:

1. Very simple and easy to program
2. Straight forward approach

Disadvantages:

1. It runs slowly and hence it is not efficient
2. Even if elements are sorted, n-1 passes are required.

Selection sort:

In selection sort, find the position of smallest element in each pass and then after pass, swap it with the element from first, second likewise.

For example, consider the following elements,

45 20 40 5 15

We apply selection sort technique,
Steps are,
1. 1st smallest element is 5 exchange it with 1st item.
2. 2nd smallest is 15, exchange it with 2nd item.
3. 3rd smallest item is 20, exchange it with 3rd item.
4. 4th smallest item is 40 exchange it with 4th item
5. Finally all the elements are sorted.

Given items	After pass 1	After pass 2	After pass 3	After pass 4
A[0]=45	5	5	5	5
A[1]=20	20	15	15	15
A[2]=40	40	40	20	20
A[3]=5	45	45	45	40
A[4]=15	15	20	40	45

Selection Sort Program:

```
#include<stdio.h>
int main()
{

   int a[100],n,i,temp,j,min;

   printf("\nEnter Number of Elements : ");
   scanf("%d", &n);

   for(i=0; i<n; i++)
```

```
    {
        printf("\nEnter an Element : ");
        scanf("%d", &a[i]);
    }

    printf("\nBefore Sorting\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t",a[i]);
    }

    for(i=0; i<n-1; i++)
    {
        min = i;
        for(j=i+1; j<n; j++)
        {
            if(a[j]<a[min])
            {
                min = j;
            }
        }
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }

    printf("\nAfter Selection Sort\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t",a[i]);
    }


    return 0;
}
```

Output:

Enter Number of Elements : 5

Enter an Element : 4

Enter an Element : 5

Enter an Element : 1

```
Enter an Element : 3

Enter an Element : 2

Before Sorting
4       5       1       3       2
After Selection Sort
1       2       3       4       5
```

Insertion sort: The idea behind the insertion sort is that first take one element, iterate it through the sorted array

Quick sort:

Merge sort:

Searching:

Searching: *Searching is a process of finding a particular data-item(Key) from huge amount of data present in a data structure.*

Two important and simple searching techniques are,

1. Sequential Search (Linear Search)
2. Binary Search

1. Sequential Search(Linear Search): A sequential search is a simple searching technique where key item is searched with a list of data-items linearly.
data-item to be searched is **key element**.

For example, **key item 20** is to be searched from the items list
20, 50, 40,30,33, 55, 66, 77, 88, 20, 10.

In linear search we need to compare key element with all the elements one-by-one until it matches.

```
if     item = a[0]
else item = a[1]
else item = a[2]
else item = a[3]
```

else item = a[4] then search is successful it is found at position 4 from a given array.

Advantages:

1. Very simple approach
2. Works well with small list
3. Best for unsorted list

Disadvantages

1. Less efficient if the array size is larger.
2. Not efficient if array elements are sorted.

Example program.

```
#include<stdio.h>
int main()
{
   int a[50],n,i,key,pos;

   //Read number of elements
   printf("\nEnter Number of Elements : ");
   scanf("%d",&n);

   //Read data
   for(i=0; i<n; i++)
   {
      printf("\nEnter Element : ");
      scanf("%d", &a[i]);
   }

   //Read key Element
   printf("\nEnter Key Element : ");
   scanf("%d", &key);

   pos = -1;
   //Linear Search
   for(i=0; i<n; i++)
   {
      if(key == a[i])
      {
         printf("\nKey %d is Found at Position %d",key, i);
         pos = i;
         break;
      }
   }
   if(pos == -1)
   {
```

```
        printf("\nKey %d is Not Found",key);
    }
    return 0;
}
```

Output

```
Enter Number of Elements : 5
Enter Element : 50
Enter Element : 40
Enter Element : 30
Enter Element : 20
Enter Element : 10
Enter Key Item to be Searched : 10
Element Found at Position : 4
```

2. Binary Search

Binary search is a simple searching technique where key element is searched with sorted array and compared with low & mid-section OR mid & high section.

Procedure:

- Array elements must be sorted in ascending or descending.
- Find the mid element by using (low+high)/2.
- Compare key element with mid, then check whether it is present in both halves based greater than or less than.
- Reset the low, mid, high values based on key, do repeatedly until we find key element with mid.

Using Iterative procedure:
Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a[20],n,i,key,mid,low,high;

    printf("\nEnter Number of Elements : ");
    scanf("%d", &n);

    printf("\nEnter Elements in Ascending Order ....");
    for(i=0; i<n; i++)
    {
```

```
        printf("\nEnter Element : ");
        scanf("%d", &a[i]);
    }

    printf("\nEnter Key Item to be Searched : ");
    scanf("%d",&key);

    low = 0;
    high = n-1;

    while(low<=high)
    {
        mid = (low+high)/2;

        if(key == a[mid])
        {
            printf("\nKey Found at Position : %d",mid);
            break;
        }
        if(key<a[mid])
        {
            high = mid-1;
        }
        else
        {
            low = mid+1;
        }
    }
if(low>high)
{
        printf("\nKey not found….");
}

    getch();
    return 0;
}
```

Output

```
Enter Number of Elements : 5
Enter Elements in Ascending Order ....
Enter Element : 10
Enter Element : 20
Enter Element : 30
Enter Element : 40
Enter Element : 50
```

```
Enter Key Item to be Searched : 30
Key Found at Position : 2
```

Using Recursive Procedure.

Base case:

1. Item not found means,
   if(low>high) return -1.
2. if item present when compare with mid,
   If(key == a[mid]); return mid;
3. make recursive call if
   mid = (low+high)/2;
   compare mid element with key
   if(key < a[mid])
   call fun bin(a,key,low, mid-1)
   else
   call fun bin(a,key,mid+1, high)

Program:

```
#include<stdio.h>
int bin_search(int a[], int key, int low, int high);
int main()
{
   int a[100],n,i,key, pos;

   printf("\nEnter Number of elements : ");
   scanf("%d", &n);

   for(i=0; i<n; i++)
   {
      printf("\nEnter Element : ");
      scanf("%d", &a[i]);
   }

   printf("\nEnter a key to be searched : ");
   scanf("%d", &key);
   pos = bin_search(a, key, 0, n-1);
   if(pos == -1)
   {
      printf("\nKey Not found..");
   }
   else
   {
      printf("\nKey Present at position : %d",pos);
```

```
    }

    getch();
    return 0;
}

int bin_search(int a[], int key, int low, int high)
{
    int mid;

    if(low>high)
    {
        return -1;
    }

    mid = (low+high)/2;

    if(key == a[mid])
    {
        return mid;
    }

    if(key < a[mid])
    {
        return bin_search(a,key,low, mid-1);
    }
    else
    {
        return bin_search(a,key,mid+1, high);
    }
}
```

Output

```
Enter Number of elements : 5
Enter Element : 1
Enter Element : 2
Enter Element : 3
Enter Element : 4
Enter Element : 5
Enter a key to be searched : 3
Key Present at position : 2
```

Advantages:

1. Simple approach.
2. Efficient technique.
3. Best for sorted list.

Disadvantages

1. Elements must be sorted.

Differences between Linear search & Binary search.

Binary search	Sequential search
1. Works on sorted items	Works on sorted a well as unsorted list
2. Very efficient of items are sorted	Very efficient if items are sorted & item present at beginning
3. Works with arrays only	Works with arrays as well as linked list
4. Less number of comparisons	More number of comparisons

Iterative and recursive searching:

The major difference between the iterative and recursive version of Binary Search is that the recursive version has a space complexity of O(log N) while the iterative version has a space complexity of O(1).

Hence, even though recursive version may be easy to implement, the iterative version is efficient.

Binary search is a search algorithm that finds the position of a key or target value within a array. Binary search compares the target value to the middle element of the array; if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful.