UNIT-6

RECURSION

Recursion is the process in which a function calls itself and directly or indirectly.

Advantage:

* Solution to base case is provided
* Solution of bigger problem is expressed in terms of smaller problem
* program is easy to visualize and prove

Disadvantage:

* Memory utilization is high
* speed is reduced.

Example:

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

Types: 1. Based on calling:

i) direct recursion

ii) indirect recursion

i) direct recursion: The function calls the same function.

Eg

```
int fibo(int n)
{
    if(n==1||n==2)
```

Scanned by CamScanner

```
        return 1;
    else
        return (fibo(n-1) + fibo(n-2));
}
```

→ fibo() is direct recursive, fn calls itself

ii) indirect recursion: The function calls another function.

Eg

```
int func1(int n)
{
    if (n<=1)
        return 1;
    else
        return func2(n);
}

int func2(int n)
{
    return func1(n);
}
```

→ func1( ) calls another function func2( ) and vice versa.

2. Based on location:

i) Tailed recursion: recursive call is the last statement of the function:

(Eg) Above code

ii) Non tailed recursion: recursive call is not the last statement of the function

Scanned by CamScanner

Example:

```
Void printfun(int test)
{
    if (test<1)
      return
    else
    {
        printf(%d,test);
        printfun(test-1);
        printf(" %d", test);
        return ;
    }
}
Int main()
{
   int test=3;
   printfun(test);
}
```

Output:

3 2 1 1 2 3

Scanned by CamScanner

Factorial of a number:

```
#include <stdio.h>
int fact (int i)
{
    if (i<=1)
        return 1;
    else
        return i*fact(i-1);
}
int main()
{
    int n;
    printf(" Enter a positive integer");
    scanf("%d", &n);
    printf("\nFactorial of %d is
                        %d", n, fact(n));
    return 0;
}
```

Output:

```
Enter a positive integer: 6
Factorial of 6 is 720
```

Fibonacci Series:

```
#include <stdio.h>
int fib(int n)
{
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
```

Scanned by CamScanner

```
    return (fib(n-1) + fib(n-2));
}
int main()
{
    int k, i=0, c;
    printf("Enter the number");
    scanf("%d", &k);
    printf("Fibonacci series\n");
    for(c=1; c<=k; c++)
    {
        printf("%d\n", fib(i));
        i++;
    }
    return 0;
}
```

Output:

Enter the number: 10

Fibonacci Series 0 1 1 2 3 5 8 13 21 34 55

Ackermann function:

- Example of total computable function that is not primitive recursive
- illustrates not all total computable functions are primitive recursive

$$A(m,n) = \begin{cases} n+1 & m=0 \\ A(m-1,1) & n=0 \\ A(m-1, A(m,n-1)) & m,n \neq 0 \end{cases}$$

Scanned by CamScanner

Example:

```
#include<stdio.h>
Int A(int m, int n);
Void main()
{
    int m,n;
    printf("Enter 2 numbers:\n");
    scanf("%d %d", &m, &n);
    printf("\n OUTPUT: %d \n", A(m,n));
}
Int A(int m,int n)
{
    if(m==0)
        return n+1;
    else if(n==0)
        return A(m-1,1);
    else
        return A(m-1, A(m,n-1));
}
```

Output: (First Pass)

Enter 2 numbers

1

0

Output: 2

Second Pass

Enter two numbers

0

5

OUTPUT: 6

Scanned by CamScanner

Quick Sort:
- Divide and conquer algorithm

steps:
1. Pick an element from array - pivot element
2. Divide unsorted array into two arrays. values less than pivot in first subarray, and values greater than pivot in second subarray (Partition process)
3. Recursively repeat step 2 until subarrays are sorted (i.e. do partitioning)

Example:

```
#include <stdio.h>
void quicksort(int number[25],
               int first, int last)
{
    int i, j, pivot, temp;
    if (first<last)
    {
        pivot = first;
        i = first;
        j = last;
        while(i<j)
        {
            while(number[i] <= number[pivot]
                    && i<last)
                i++;
            while(number[j]>number[pivot])
                j--;
            if(i<j)
            {   temp = number[i];
                number[i] = number[j];
```

Example

2	5	1	3 (Pivot)

no.s < 3, left
no.s > 3, right

[2 | 1] [5]
↓ sort ↓
[1 | 2] + [3] + [5]

Scanned by CamScanner

```
            number[j] = temp;
        }
    }

    temp = number[pivot];
    number[pivot] = number[j];
    number[j] = temp;
    quicksort(number, first, j-1);
    quicksort(number, j+1, last);
  }
}

int main()
{
    int i, count, number[25];
    printf("How many elements are you
             going to enter?:");
    scanf("%d", &count);
    printf("Enter %d elements:", count);
    for(i=0; i<count; i++)
       scanf("%d", &number[i]);
    quicksort(number, 0, count-1);
    printf("Order of sorted elements");
    for(i=0; i<count; i++)
       printf("%d", number[i]);
    return 0;
}
```

Output:

How many elements are you going to enter?: 6

Enter 6 elements

30 3 45 6 80 23

Order of sorted elements:

3 6 23 30 45 80

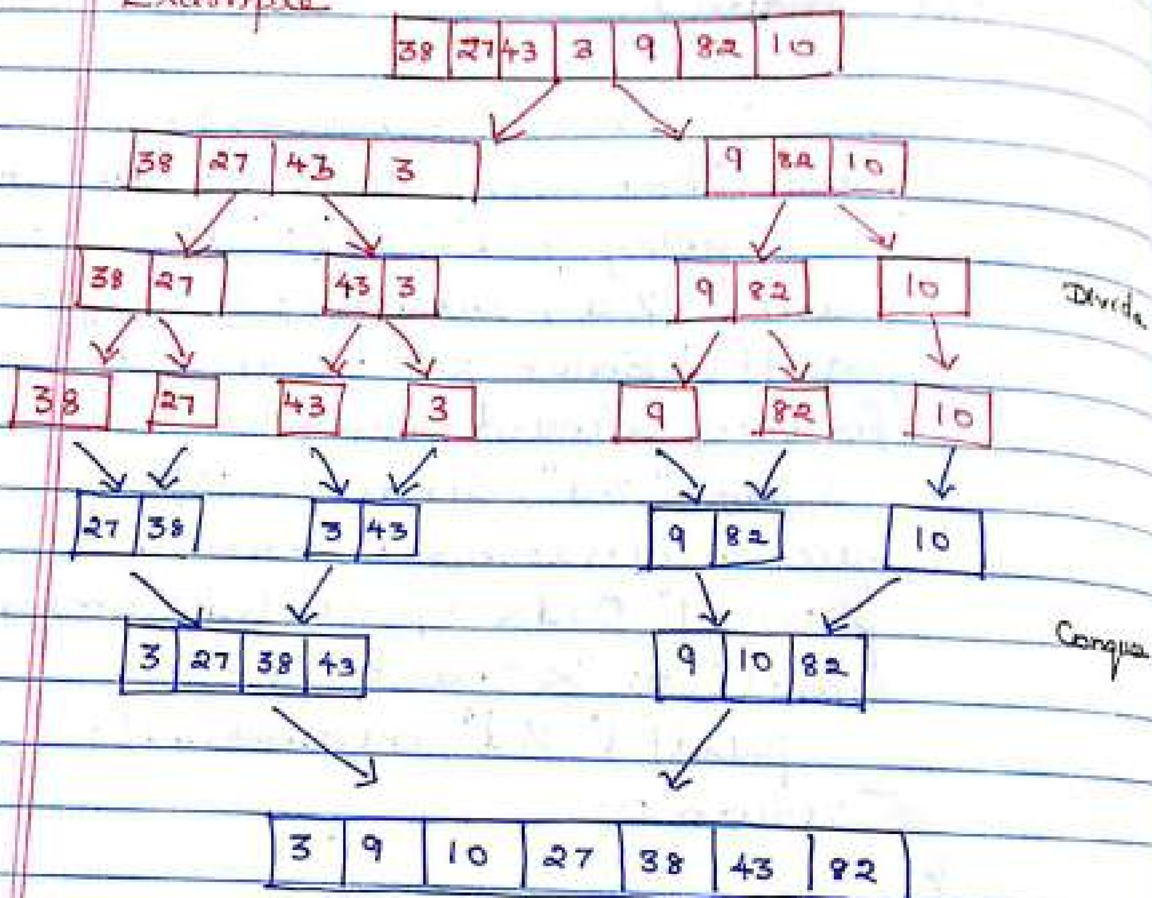
Scanned by CamScanner

Merge Sort:
- uses divide & conquer technique

Steps:

* Array is split into 2 using center.
* The arrays are then sorted & merged.
* Splitting and merging is done recursively till there is one element in the array.

Example



Program:

```
#include<stdio.h>
void mergesort(int a[],int i, int j);
void merge(int a[], int i1,int j1,int i2,int j2);
int main()
{
```

```
int a[30], n, i;
printf(" Enter no. of elements");
scanf("%d", &n);
printf(" Enter array elements");
for(i=0; i<n; i++)
    scanf("%d", &a[i]);
mergesort(a, 0, n-1);
printf("\n Sorted array is:");
for(i=0; i<n; i++)
    printf("%d", a[i]);
return 0;
}
void mergesort(int a[], int i, int j)
{
    int mid;
    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a, i, mid);      // left recursion
        mergesort(a, mid+1, j);    // right recursion
        merge(a, i, mid, mid+1, j); // merging of
                                    // two sorted sub-arrays
    }
}
void merge(int a[], int i1, int j1, int i2, int j2)
{
    int temp[50];
    int i, j, k;
    i = i1;     // beginning of first list
    j = i2;     // beginning of second list
    k = 0;
    while(i<=j1 && j<=j2) // while elements in
                          // both lists
```

```
{
    if(a[i]<a[j])
        temp[k++]=a[i++];
    else
        temp[k++]=a[j++];
}
while(i<=j1) //copy remaining elements of the first list

    temp[k++]=a[i++];
while(j<=j2) //copy remaining elements of second list

    temp[k++]=a[j++];
//Transfer elements from temp[] back to a[]
for(i=i1,j=0;i<=j2;i++,j++)
    a[i]=temp[j];
}
```

Output:

```
Enter no. of elements
5
Enter array elements
5   15   10   3   6
Sorted array is
3   5   6   10   15
```

Scanned by CamScanner