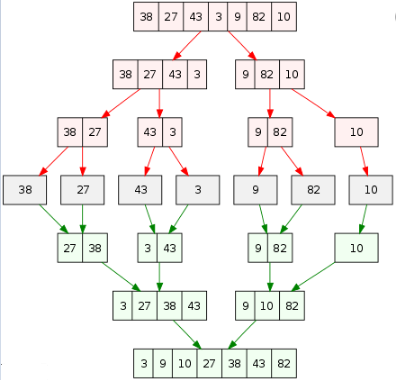
# Divide and Conquer Algorithm

Divide and conquer is a design which follows the similar principle of recursion.

The problem gets broken down in several prices until it gets solved. Many solution combined to form a solution of a problem.

.

**How to find out when to use Divide and conquer:**

Magic Framework: Optimum sub-structure.

**Property**: Fibonacci Series -> Fib (n) = Fib (n-1) + Fib (n-2)

**Examples**:

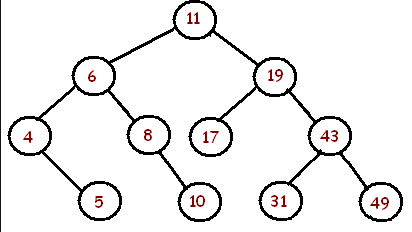
Merge Sort

Quick Sort

Binary Search

Difference between Greedy Algorithm and Divide and conquer.

## Binary Search:



Search (root, Value)

If (root equal null)

Return null

Else if (root equal to Value )

Return root.

Else if (value < root)

Search (root.left, value)

Else

Search (root.right, value)

# Quick Sort

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 3 | 2 | 4 | 9 | 8 | 7 |

**Quicksort(i, j, p)**

if(j<p)

r= partition (i, j, p)

Quicksort (i, j, r-1)

Quicksort (i, r+1, j)

partition (i, j, p)

pivot = p

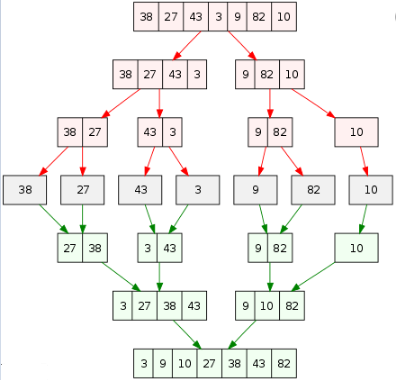
i = j-1

for (y=j to p)

if (a[y]<=a[p])

increment i and then swap a[j ,p]

# Merge Sort



mergesort(A,i,r)

if r>1

middle m= i+r/2

mergesort(a,i,m)

mergesort(a,m+1,r)

merge(a,i,m,r)

Merge A,p,m,r)

createtmp arrays L&R and copy A, p,m into L&A, m+1, r into Re-structuring

i+j=0

loop k=p to r/2

if L[i]<R[jj]

a[k]=L[i], i++

else

a[k]=R[j], j++

# Fibonacci Series:

**Definition**: It is series of number which is sum of last 2 preceding number starting from 0 and 1.

**Example** = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 …

FibonacciRecursion()

If (n < 1) return error

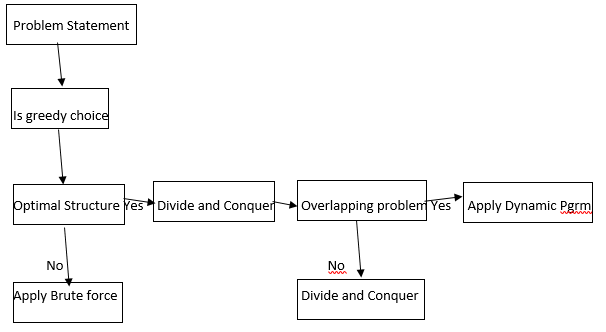
Else if (n==1): return 0

Else if (n==2): return 1

Else : return FibonacciRecursion(n-1) + FibonacciRecursion (n-2)

## House thief:

**Condition**: N house. Thief cannot steal from two adjacent house. Find maximum a thief can steal.



1. 8,9,2,20,4,3,1
2. 7,5,2,18,4

MaxMoney (NetWorth, currentIndex)

If currentIndex >= NetWorth ; Return 0

StealCurrentHouse = NetWorth[currentIndex] + MaxMoney (NetWorth, currentIndex+2)

SkipCurrentHouse = MaxMoney (NetWorth, currentIndex+1)

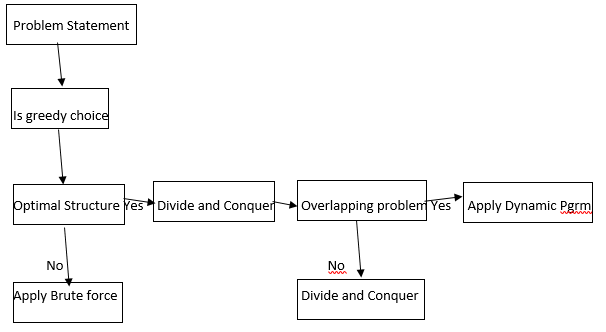
Return math.max( StealCurrentHouse, SkipCurrentHouse)

## Number Factor:

**Problem**: Number of ways to express N using different integers.

**Ex**: 5 Using 1, 3, 4

Answer is 6 -> {4,1},{3,1,1},{1,4},{1,3,1},{1,1,3},{1,1,1,1,1}



WaystoN(N)

If (n==0)|| (n==1) || (n==2) ; Return 1 //{}, {1}, {1,1}

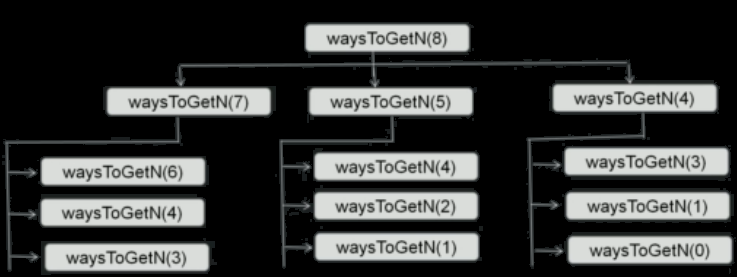
If (n==3) ; return 2 //{111},{3}

Int sub1= WaystoN(N-1)

Int sub3= WaystoN(N-3)

Int sub4= WaystoN(N-4)

Return sub1+sub3+sub4;



**Convert One string into another:**

Given S1 and S2; Need S2 to be converted to S1; Minimum count of characters to convert to S1;

Ex: S1= “Cat” ; S2 = “Bat” ; Output : 1 ;

**3 Operation:**

**Remove**: S1 = and S2 = asnd

**Add**: S1 = ant S2 = an

**Update**: S1 = and S2 = any

FindMaxOperation(S1, S2, i1, i2)

If (i1 = S1.length); return S2.length-i2; //End of S1, so all the rest of S2 should be deleted

If (i2 = S2.length); return S1.length-i1; //End of S2, so all the rest of S1 should be inserted

If(s1.charAt(i1)=S2.charAt(i2)); return(FindMaxOperation(S1, S2, i1+1, i2+1)

C1= 1+ return(FindMaxOperation(S1, S2, i1+1, i2)

C2= 1+ return(FindMaxOperation(S1, S2, i1, i2+1)

C3= 1+ return(FindMaxOperation(S1, S2, i1+1, i2+1)

Return min(C1,C2,C3)