Lecture 31
Recursion L-1

① Function calling itself is called Recursion.

② Big problem } depends on
Choti problem, then we use Recursion

$2^n = 2^* \times 2^{n-1}$

$2^4 = 2 \times 2^3$

$2^3 = 2 \times 2^2$

$2^2 = 2 \times 2$

$2 = 2 \times 2^0$

$F(n) = F(n) \times F(n-1)$

This is Recurrence Relation.

③ Factorial

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$= 5 \times 4!$

$n! = n \times (n-1)!$

$F(n) = n \times F(n-1)$

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1!$

$1! = 1 \times 0! = 1$

When we stop at a condn, it is called as base case or base condn.

($1! = 1$, $0! = 1$) is our base case

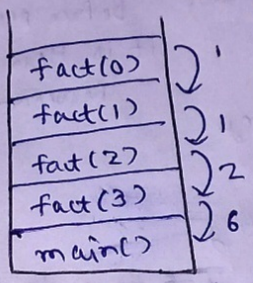
④ Base case tells where program will stop, Recurrence relation is needed.

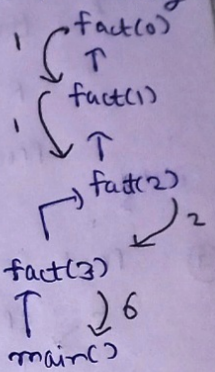
⑤ In Base case, return statement is necessary.

⑥ Without base case, Segmentation fault will come

$4 \to 3 \to 2 \to 1 \to 0 \to -1 \to -2 \to -3 \ldots$

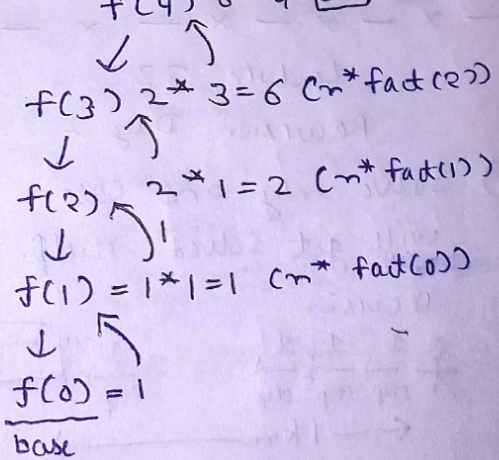
There fun^n call will keep on happening if there is no base case the memory gets filled up That is why Base case is necessary





⑦ Recursion Tree

Let $n = 4$



⑧ Head Recursion V/S Tail Recursion

```
Func()
{
    Base case
    Processing
    Recurrence Relation
}
```

When RR comes at last, it is Tail Recursion

```
Func()
{
    Base case
    RR
    Processing
}
```

When RR comes before processing it is Head Recursion.

⑨ Print Counting

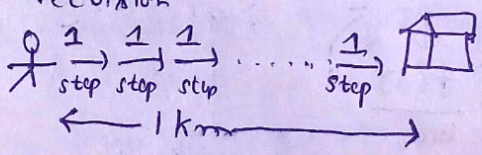
5 ↘ (print it & move to n--)
4 ↘
3 ↘
2 ↘ 1 ↘ 0 (stop)

— x — x — x — x — x —

Lecture 32

Recursion Day 2

① Solve only one case, rest will get solved itself. This is recursion

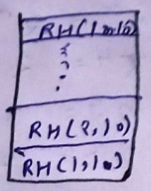


1) 1 step tele means one case solved
2) Baaki recursion solve krlega
3) Kitna pohonchke ruk jana → base case

② reachHome(int src, int dest)
(src: 1 → 10)

```
{
    // base case
    if (src == dest)
    { return; }
    // Processing
    src++;
    // Recursive call
    reachHome(src, dest);
}
```

(1,10) // start
↓
(2,10)
↓
(3,10)
↓
(4,10)
⋮
(10,10) // Hit base case



③ Fibonacci Series

0, 1, 1, 2, 3, 5, 8, 13, 21 ...

0+1 = 1 , 3+5 = 8
1+1 = 2 , 5+8 = 13
2+3 = 5 , 8+13 = 21

Find 8th Fibonacci Number?

① Recursive relation

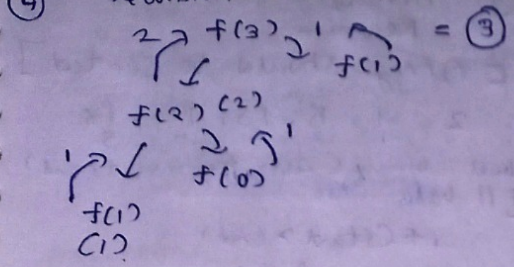
$F(n) = F(n-1) + F(n-2)$

② Base Case

```
if (n==0) return 0;
if (n==1) return 1;
```

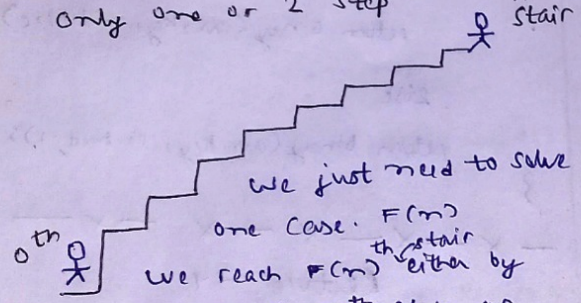
```
fib(int n)
{   if (n==0)
        return 0;
    if (n==1)
        return 1;
    int ans = fib(n-1) + fib(n-2);
    return ans;
}
```

④ Recursion Tree



⑤ Count ways to reach nth Stairs.

we are at 0th, we need to reach nth stair, we can take only one or 2 step

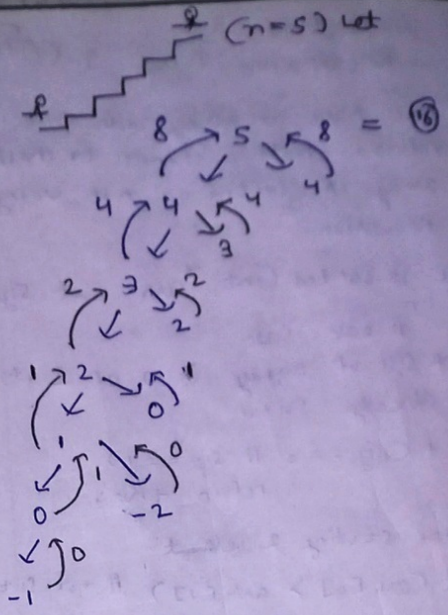


we just need to solve one case. F(n)

we reach F(n)th stair either by jumping from (n-1)th stair or (n-2)th stair

$F(n) = F(n-1) + F(n-2)$

```
// Base case
if (n < 0)        // if stair is -ve means DNE so return 0
    return 0;
if (n == 0)       // 0th stair ka
    return 1;     // 0th stair same ka
                  // no. of ways = 1
```



⑥ Say Digits

i/p → 412

o/p → Four one two

```
string arr[] = {"zero", "one" .... "nine"};

void sayDigit(int n, string arr[])
{
    if(n == 0)
        return;

    int digit = n % 10;
    n = n / 10;
    sayDigit(n, arr);
    cout << arr[digit];
}
```

Lecture 33
Recursion Day 3 (Array)

① We have an array with some values, create a funcn to check array is sorted or not using Recursion.

```
bool isSorted(int *arr, int size)
{
    // base case
// if size of array is 0 or 1 its already sorted
    if(size==0 || size==1)
        return true;
// compare starting 2 element
    if(arr[0] > arr[1]) // not sorted
        return false;
    else // check for remaining array
        bool ans = isSorted(arr+1, size-1)
        return ans;
}
```

② Linear Search using Recursion.

3 (0), 5 (1), 1 (2), ? (3), 6 (4) key = 6

we will pass each element one by one and check if its equal to key or not

```
// base case
if(size==0) return false;
// solve only for one case
if(arr[0]==key) true;
else // rest recursion will solve
    return linear(arr+1, size-1, key);
```

③ Binary search using Recursion

[Array should be sorted]

2 (0), 4 (1), 6 (2), 10 (3), 14 (4), 16 (5)

```
bool binary(arr, key, start, end)
{ // base case
    if(start > end)
        false;
    mid = (s + (e-s)/2);
    if(arr[mid] == key)
        true
    if(arr[mid] < key)
        return binary(arr, key, mid+1, e)
    else
        return binary(arr, key, s, mid-1);
}
```

— x — x — x — x — x —

Lecture 34
Recursion Day 4 (Strings)

Q
```
i/p = "abcde"
o/p = "edcba"
i/p = "Lokesh"
o/p = "hsekoL"
```

Reverse the string using Recursion.

we need i, j, &string in parameter.

i = 0, j = str.length - 1

```
// base case
if(i>j)
    return;
swap(str[i], str[j])
i++, j--;
reverse(i, j, str)
```

② Check palindrome

String = abba → Palindrome
reverse string = abba

if (string == reverse string)
it is a palindrome

```
bool check(str, i, j)
{
    // base case
    if(i>j)
        return true;
    if(str[i] != str[j])
        return false
    else
        return check(str, i+1, j-1)
}
```

③ Power of 2 using Recursion

a^b → b is even → $a^{b/2} \times a^{b/2}$
a^b → b is odd → $a \times (a^{b/2} \times a^{b/2})$

$2^9 = 2 \times (2^4 \times 2^4)$
$2 \times (2^2 \times 2^2 \times 2^2 \times 2^2)$
$2 \times (2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2)$
$2 \times [2(2^0) \times 2(2^0 \cdots 2 \times 2^0)]$
$2 \times [2 \times 1 \times 2 \times 1 \cdots 2 \times 1]$

```
int power(int a, int b)
{
    // base case (b is power in this case)
    // (anything)^0 = 1
    if(b==0)
        return 1
    // (anything)^1 = anything
    if(b==1)
        return a

    int ans = power(a, b/2);

    if(b&1==0) // even
    {
        return ans*ans;
    }
    else { // odd
        return a*ans*ans;
    }
}
```

we can use bitwise operator to check even, odd

```
if(!(b&1))
{
    // even
}
```

b&1 == 1	means	odd
b&1 == 0	means	even

④ Bubble Sort using Recursion

```
void sortArray(int *arr, int size)
{
    // base case
    if(size==0 || 1)
    {
        return;
    }
// sirf ek element sahi place krde rest Recursion will do.
// largest element ko last mai phenka do
    for(i=0 → i<n-1)
    {
        if(arr[i] > arr[i+1])
        {
            swap(arr[i], i+1)
        }
    }
    // Recursive call
    sortArray(arr, n-1);
}
```

Lecture 35

Merge Sort Recursion

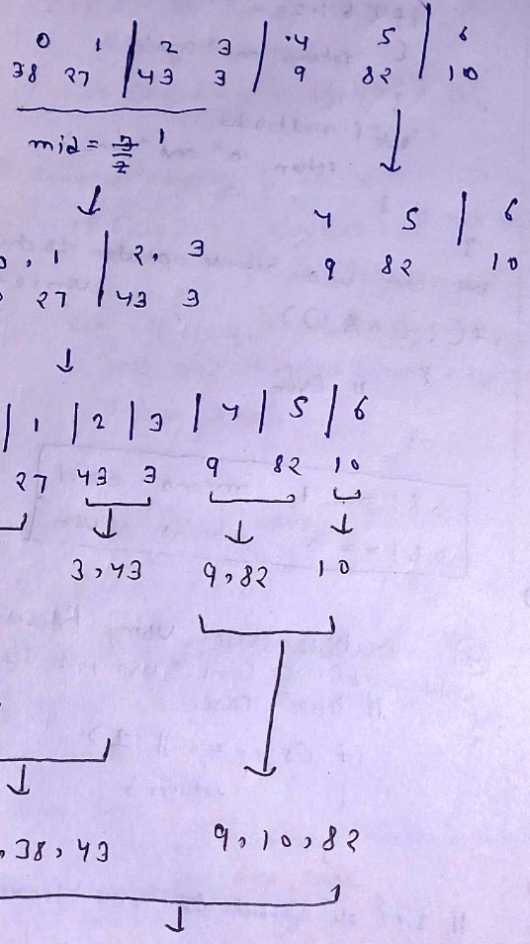
i/p: array

o/p: sorted array

38, 27, 43, 3, 9, 82, 10

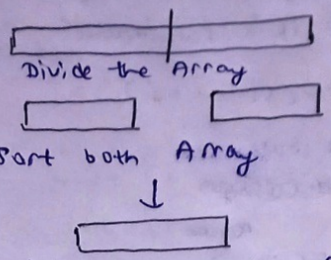
Divide Array into 2

mid = $\frac{s+e}{2}$



This is Fastest Sorting algo of all sorting algo till now.

We will do it by Recursion



1st Approach

1) create new Array, copy all values in it

2nd Approach

1) Use index start, end, mid.

Code

```
void mergeSort(int *arr, int s,
  int e )
        ↓
        n-1
{     // base case
      if (s>=e)
            return;

  mid= s+ (e-s)/2;

    // left vala divide kro
    mergeSort(arr, s, mid);

    // right kro
    mergeSort(arr, mid+1, e);

  // merge 2 Sorted Array
    using above approach of
    create 2 extra array

    merge(arr, s, e );
}
```

Max TC: O(nlogn)
SC: O(n)

O(n) to merge two halves of Array
O(logn) to Divide array into 2 halves.

- ★ It is a stable algo means it maintains relative order of equal elements in i/p array.
- ★ it performs well on large data set also as its worst TC is O(nlogn)
- ★ It is not an in-place algo means it require extra space.
- ★ Not good for small dataset
- ★ Its a Recursive algo so can result in many func calls.

— X — X — X — X — X —

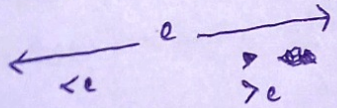
Lecture 36

Quick Sort using Recursion

(1) It is little different from merge sort

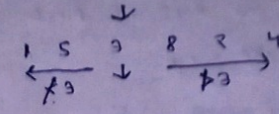
0	1	2	3	4	5
3	5	1	8	2	4

1) Take one element & put it its right place such that all elements to left of its are smaller than it, all ele on its right are greater than it



2) This is called Partitioning.

3) 3 5 1 8 2 4



1 2 3 8 5 4

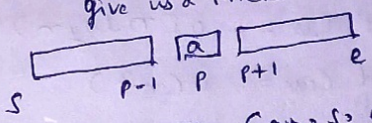
1) Put a in pos^n such that it is at its right place.
Left elements are smaller than a
Right elements are greater than a.

2) We will do for first element rest will be done by recursion

```
void quickSort (arr, start, end)
{
    // base case
    if (s >= e)
        return;

    // partition
    // We will call a func^n which will give us a index P
    p = partition (arr, s, e);

    // Recursive call
    quickSort (arr, s, p-1);
    quickSort (arr, p+1, e);
}
```



How to do Partition ?

pivot

(3)	1	4	5	2

↳ take a pivot
↳ Count no. of element < pivot
↳ put pivot in index = start + Count

4, 1, 3, 5, ?

↳ Now match <a, >a cond^n for LHS and RHS of pivot.

↳ we take i, j

4 1 3 5 2
i j

we see 4 > 2 so swap(i, j)

2 1 3 5 4
i j

↓

2 1 3 5 4
i
j

now left part < 3

now right part > 3

So we have placed 3 in its right pos^n successfully.

// TC :

Best Case = O(n log n)

Worst Case = O(n^2)

Average case = O(n log N)

☆ efficient in large dataset

☆ not eff in smaller dataset

☆ Not stable algo

☆ based on Divide & conquer algo

// SC = O(n)

Lecture 37

Day 7, Subset & subsequences

i/p = [1, 2, 3]

o/p = power set

[[], [1] [2] [3] [1, 2]
[2, 3] [1, 3] [1, 2, 3]]

Set of all subsets = Power Set

no. of elmnt in Power Set = 2^n

n = no. of elmnt in i/p set

Approach

This is based on pick & not pick concept.

NP = Not Pick, P = Pick

C([1, 2, 3], [ ])

1 NP ↙ ↓ 1 P

[1, 2, 3], [ ] ; [1, 2, 3] [1]

2 NP ↙ ↓ 2 P ; [1] [1, 2]

[1, 2, 3] [ ] [1, 2, 3] [2] ; [1, 2], [1, 2, 3]

3 NP ↙ ↓ 3 P ↙ ↓

[ ] [3] [2] [2, 3] [1] [1, 3]

now i has moved out of ~~array~~ array so its the base case.

code

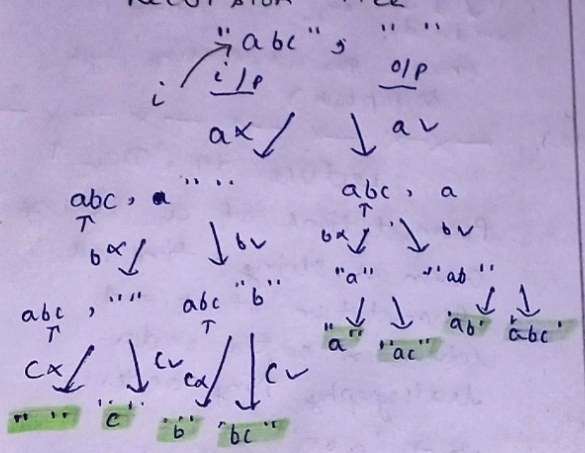
(i/p array, o/p array, index) are our parameters.

Subsequences of String

i/p = "abc"

o/p = "a", "b", "c", "ab", "bc", "ac", "abc"

Recursion Tree



Code

```
void solve (str, string output, int
  index, vector<string> &ans)
{   // base case
    if (index >= str.length())
    {  if (output.length() > 0)   // Do not include "" case
       {  ans.push(output);
       }
    }
    // exclude
    solve (str, output, index+1, ans);

    // include
    char elem = str[index];
    output.push(elem);
    solve (str, output, index+1, ans);
}
```

Lecture 38 (Day 8)

Phone Keypad problem using Recursion (Taste of Backtracking)

① i/p = string = "1234" [2-9]

In a Keypad we have some alphabets associated with each no. Give subsets of all such alphabets that can be formed by pressing numbers in i/p string.

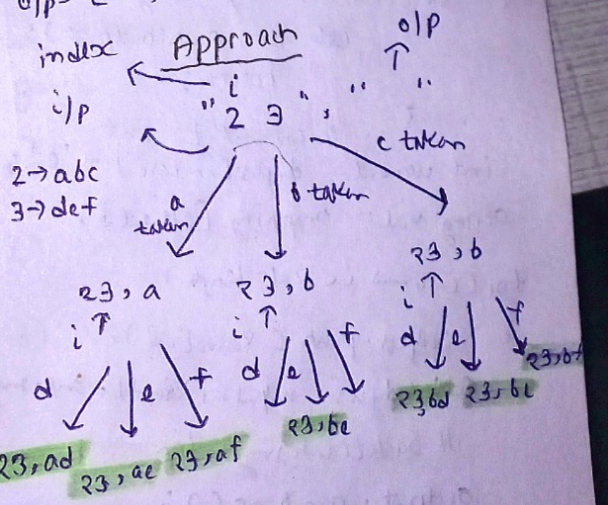
1	2	3
∞	abc	def
4	5	6
ghi	jkl	mno
7	8	9
pqrs	tuv	wxyz
*	0	⇧
+	␣	#

i/p = "23" b/w [2-9]

o/p = "ad", ae, af, bd, be, bf, cd, ce, cf

i/p = "2" b/w [2-9]

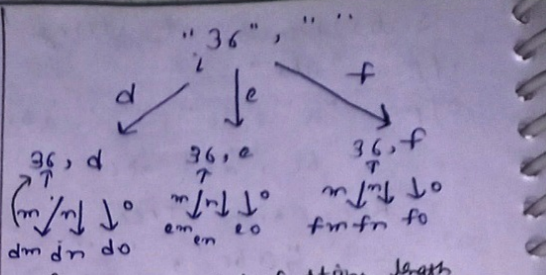
o/p = ["a", 'b', 'c']



It is same as subset or subsequence Question.
just that we need to create a string array to do mapping of 0 - 9 with alphabets
we need to back-track also i.e pop-back value before returning

Code

```
vector<string> combination(string digits)
{
    if(digits.length()==0)
    { return {};
    }
    vector<string> ans;
    string output = "";
    int index = 0;
String mapping[10] = {"", "", "abc", "def"
, "ghi", "jkl", "mno", "pqrs", "tuv",
"wxyz"};
solve(digits, output, index, ans, mapping);
return ans;
}
Void solve(digits, output, index, &ans, mapping)
{   if(index >= digit.length)
    {   ans.push(output);
        return;
    }
    // convert to int
int element = digits[index] - '0';
string val = mapping[element];
for(i=0 → i<val.length)
{ output.push(value[i]);
solve(digits, output, index+1, ans, mapping)
// backtracking step
output.popBack();
}
}
```



Lecture 39, Day 9

Permutations of a string

Given an string, find all permutation of it and return in sorted order lexicography ling order.

i/p → "abc"
o/p → 'abc', 'bac', 'cab', 'acb', 'bca', 'cba'

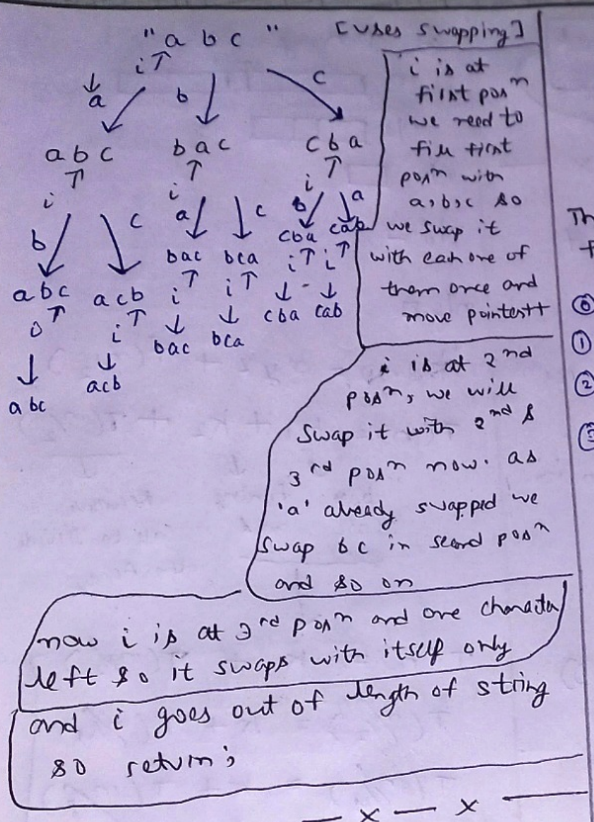
Approach

we have 3 places and we have to fill them with a, b, c in every possible way.

if length of string = n
no. of permutation = n!

Approach 1 (Take more SC)
1 DS, 1 mapp to check which place is filled or not to store o/p

Approach 2 (SC optmised)



— x — x — x — x —

Lecture 40 (Day 10)

Rat in a Maze Problem

Q Given 2D Array of
x rows, y cols
1 means open path
0 means blocked path

Source = 0,0

destination = n-1, n-1

can only ↑, →, ←, ↓

0 → 1 not possible

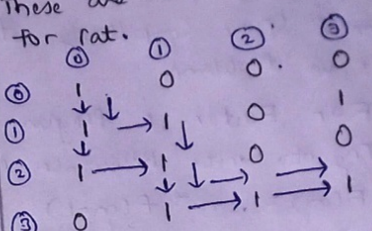
1 → 1 possible

1 → 0 not possible

Give all possible ways fulfilling cond^n

up → (x-1, y)
down → (x+1, y)
Left → (x, y-1)
Right → (x, y+1)

These are our 4 movements for rat.



DDRDRR & DRDDRR

We make a visited Array which is initialised with 0.

The cell we pass in actual array we mark it 1 in visited Array.

now while going D/L/R/U we check whether is visited already or not.

we check 3 conditions before going or making any movement.

i) i, j should be inside matrix

2) m[k][l] = 1 (non block path)

3) visited[k][l] = 0 (The path is not visited earlier)

While coming back we mark visited = 0 again

// backtracking step

— x — x — x — x —

Lecture 41

Time & Space Complexity of Recursive Problems

① For Factorial Program

1) Try to write Recurrence rel^n

$F(n) = n * f(n-1)$

2) Some Const time will happen in Base case say k_1

3) n^* take time = k_2

4) $f(n-1)$ take time = k_3

$T(n) = k_1 + k_2 * + (T(n-1))$

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

$T(n-1) = k + T(n-2)$

⋮

$T(n-k) = k + T(0)$

n times

now $T(0) = k_1$

$T(n) = nk + k_1$

const ignore

$T(n) = n * k$

const ignore

$T(n) = O(n)$

② Binary Search

mid

n

$n/2$

$n/4$

$n/8$

⋮

n/k

$F(n) = xyz + f(n/2)$

$T(n) = k_1 + k_2 + T(n/2)$

k_1: base case; k_2: finding mid; $T(n/2)$: Recursive call to Divide the Array

$T(n) = k + T(n/2)$

$T(n/2) = k + T(n/4)$

$T(n/4) = k + T(n/8)$

⋮

$T(2) = k + T(1)$

$T(1) = k$

a time

Add all eq^n

$T(n) = a * k$

what is a?

$n \to n/2 \to n/4 \cdots \to 1$

$= \frac{n}{2^a} = 1 \Rightarrow a = \log n$

So $T(n) = k \log n$

const ignore

$T(n) = O(\log n)$

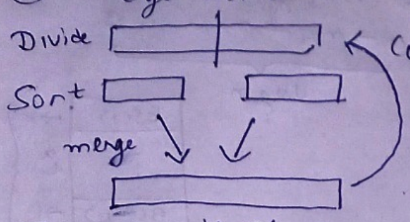
③ Divid

Sor

T

T(

③ Merge Sort



There are 4 steps

$T(n) = k_1 + k_2 + T(n/2) + T(n/2)$

$T(n/2)$: left part, $T(n/2)$: Right part

$+ \underbrace{k_3 n}_{\text{merge } n \text{ size array}} + \underbrace{k_4 n}_{\text{Copy } n \text{ size array}}$

$T(n) = k + 2T(\frac{n}{2}) + \underbrace{n(k_3 + k_4)}_{k_5}$

$T(n) = k + 2T(\frac{n}{2}) + nk_5$

Const ignore

$T(n) = 2T(\frac{n}{2}) + nk_5$

$2 * [T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}k_5]$

$2 * [T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{4}k_5]$

⋮

$2 * [T(1) = k_5]$

a times

$T(n) = a * (nk_5)$

$n \to n/2 \to n/4 \cdots\cdots 1 = a$ times

$\frac{n}{2^a} = 1, \quad a = \log n$

$T(n) = (\log n)(nk)$

k is const so ignore

$\boxed{T(n) = n \log n}$

④ Fibonacci Series

```
fib(int n)   // Base case
{ if(n==0 || n==1) → k1
      return n;
  // Recursive call
  return fib(n-1) + fib(n-2);
}
```

fib(n-1) → T(n-1), + → k_2, fib(n-2) → T(n-2)

$T(n) = k_1 + k_2 + T(n-1) + T(n-2)$

$T(n) = k + T(n-1) + T(n-2)$

$T(n-1) = k + T(n-2) + T(n-3)$

$T(n-2) = k + T(n-3) + T(n-4)$

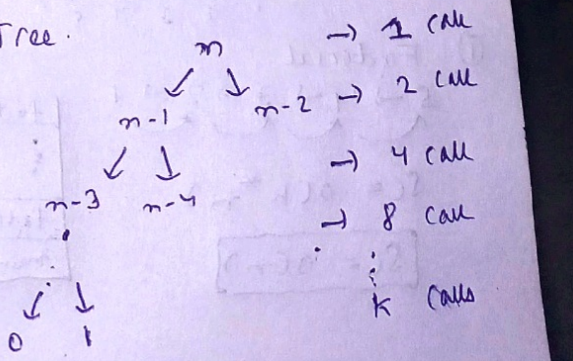
⋮

$T(1) = k_1$ (base case)

$T(0) = k_1$ (base case)

But $T(n-2)$, $T(n-3)$... ..... is not cancelling so This way is not possible

Let's try making Recursive Tree.



If each node takes k time then Total time will be k * total node

no. of node

1	(2^0) calls
2	(2^1) calls
4	(2^2) calls
8	(2^3) calls
16	(2^4) calls
32	(2^5) calls
⋮	⋮
2^n	Calls.

Total node = $2^{n+1} - 1$

as $1 + 2^1 + 2^2 \cdots 2^n$

$TC = k * (2^{n+1} - 1)$

const

$TC = k(2^{n+1})$

$\boxed{TC = 2^n}$ (ignore const)

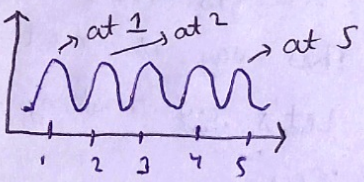
(exponential TC)

— x — x — x — x — x — x —

Space Complexities

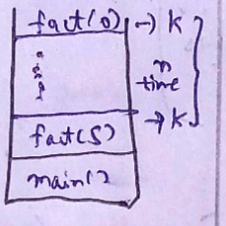
Space reqd as func^n of i/p.

max space reqd at any instant of time



① Factorial

5 → 4 → 3 → 2 → 1



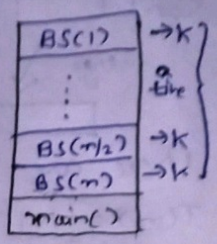
$SC = O(k * n)$

$\boxed{SC = O(n)}$

② Binary Search

$n \to n/2 \to n/4 \cdots 1$

$\log n$



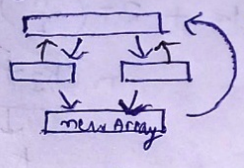
$SC = k * \log n$

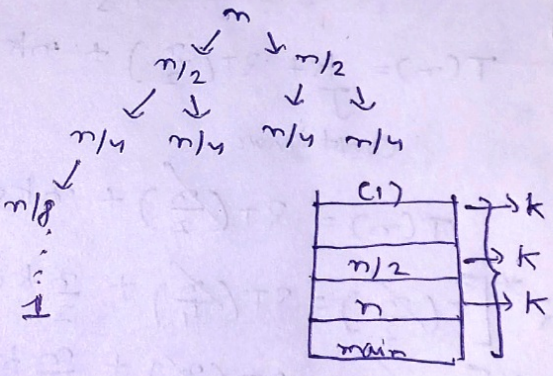
const

$SC = \log n$

$\boxed{SC = O(\log n)}$

③ Merge Sort





$SC = k \log n + n$ (new array to copy)

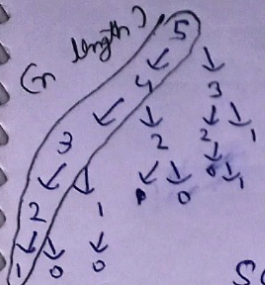
new Array is same size of n.

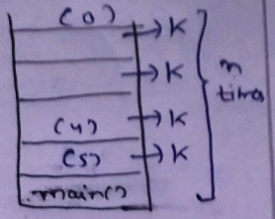
$\boxed{SC = \log n + n}$

Smaller than n so ignore

$\boxed{SC = O(n)}$

④ fibonacci Series





$SC = n * k$

$SC = O(n)$