① Fibonnaci series

```
int fibonnacci (int num){
  //base case
  if (num == 1){
      return 0;
  }
  if (num == 2){
      return 1;
  }
  //recursive relation.
  return fibonnacci (num-1) + fibonnacci (num-2);
}
int main (){
  int num;
  cout << "Enter the n'th term : ", cin >> num.
  int fib = fibonnacci (num);
  cout << fib;
```

Output: 5th term

3

(2) Factorial

```
int factorial (int num){
  //base case
  if (num == 1){
    return 1;
  }
  //recursive relation
  int fact = num * factorial (num-1);
  return fact;
}

int main(){
    int num;
    cout << num;
    int fact = factorial(num);
    cout << "The factorial of" << num << "is" fact;
```

Output 5
The factorial 5 is 120

(3) printcount in reverse HEAD & TAIL Recursion

```
void printCount (int num){
  //base case
  if (num == 0){
    return 0;
  }
  //processing
  cout << num << " ";
  //recursive relation
  printCount(num-1);
}

int main(){
    int num;
    cout << "Enter the value of num :"; cin >> num;
    printCount (num);
}
```

Output
num = 5
5 4 3 2 1

④ Power of 2:

```
int powerof two (int num){
  //base case
  if(num == 0){
     return 1;
  }
  //recursive relation.
  int ans = 2*powerof two (num-1);
  return ans;
}
int main(){
  int num;
  cout << num;
  int power = powerof two (num);
  cout << "The power of 2^ <<num << " is :" << power;
```

output

num = 0
The power of 2^0 is 1
num = 3
The power of 2^3 is 8.

Recursive Tree

num = 4

ans = 2 * powerof two (3) 2*8 = 16

ans = 2 * powerof two (2) 2*4 = 8

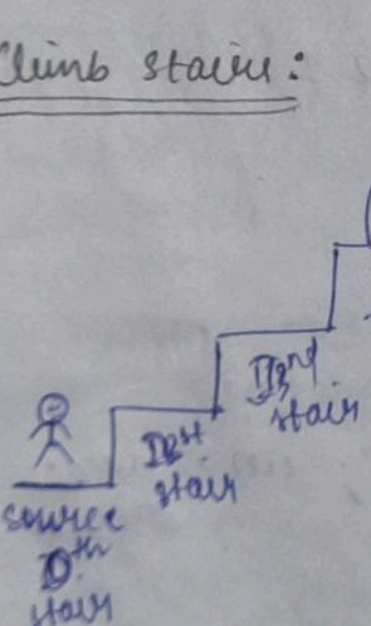
ans = 2 * powerof two (1) = 2*2 = 4

ans = 2 * powerof two (0)
return 1

2*1 = 1 = 2

ans = 16

⑤ Climb stairs:



we can ~~take~~ use 1 stairs at a time or 2 stairs at a time to reach destination.

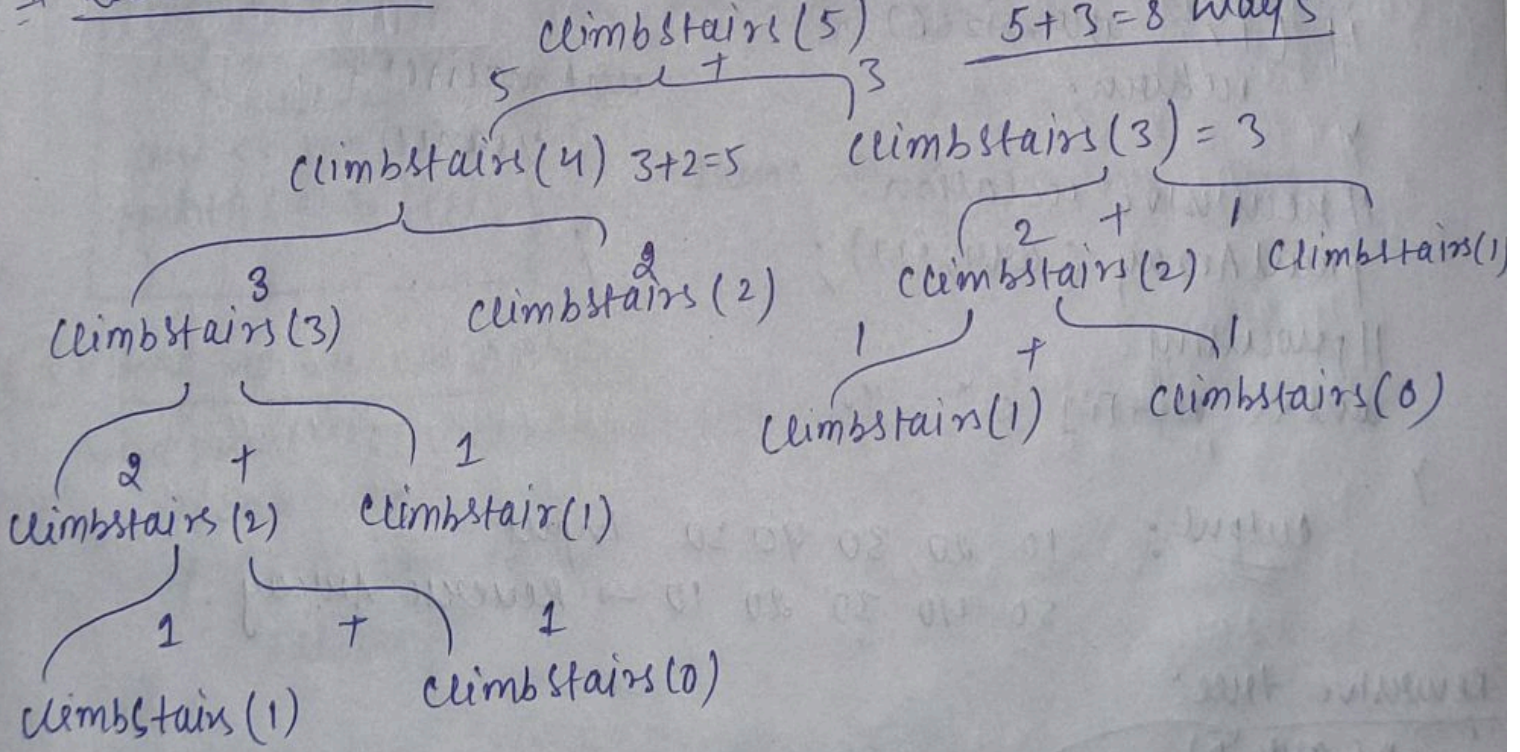
$f(n) = f(n-1) + f(n-2)$

because find total cases

```
int climbstairs (int num){
    //base case
    if (num == 0 || num == 1){
        return 1;
    }
    //recursive relation.
    int totalways = climbstairs (num-1) + climbstairs (num-2);
    return totalways;
}
int main(){
    int num;
    cout << "Enter the stairs you want to reach : ";
    cin >> num;
    int totalcases = climbstairs (num);
    cout << "The total ways to climb is : " << totalways;
}
```

Output: Enter the stairs you want to reach : 5
The totalways to climb is : 8.

* Recursive Tree:



(6) Print Array with Recursion

```
void printArray(vector<int> arr, int i){
  //base case
  if (i >= arr.size()){
      return;
  }
  //processing
  cout<< arr[i] <<" ";
  //recursive relation
  printArray(arr, i+1);
}
```

```
int main(){
  int size;
  cin>>size;
  int i=0;
  vector<int> arr(size);
  // input of an array
  for(int i=0; i<arr.size(); i++){
      cin>> arr[i];
  }
  printArray(arr, i);
```

Output:

10 20 30 40 50 → Input
10 20 30 40 50 → Recursion will print Array.

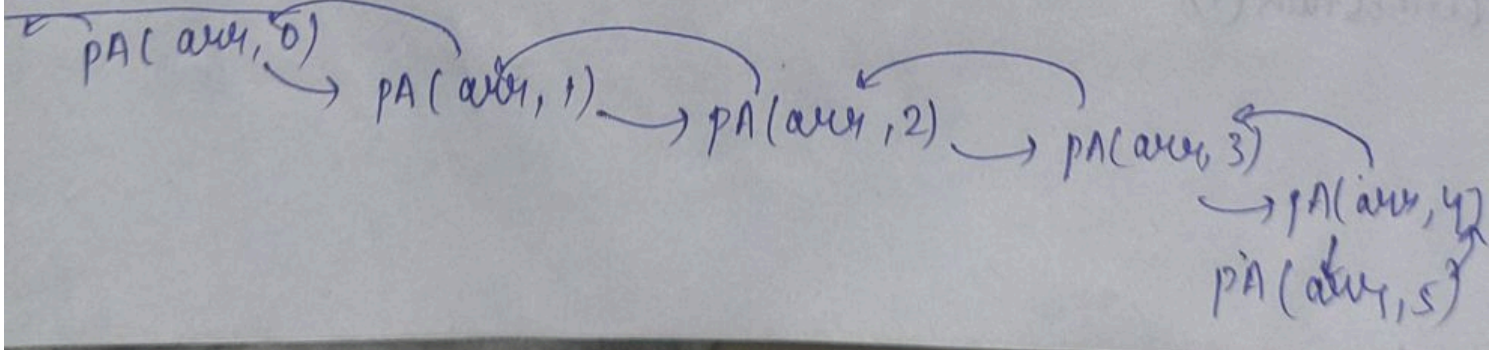
(7) Print Array with Recursion in reverse Order

```
void printArray(vector<int> arr, int i){
  //base case
  if (i >= arr.size()){
      return;
  }
  //recursive relation
  printArray(arr, i+1);
  //processing
  cout<< arr[i] <<" ";
}
```

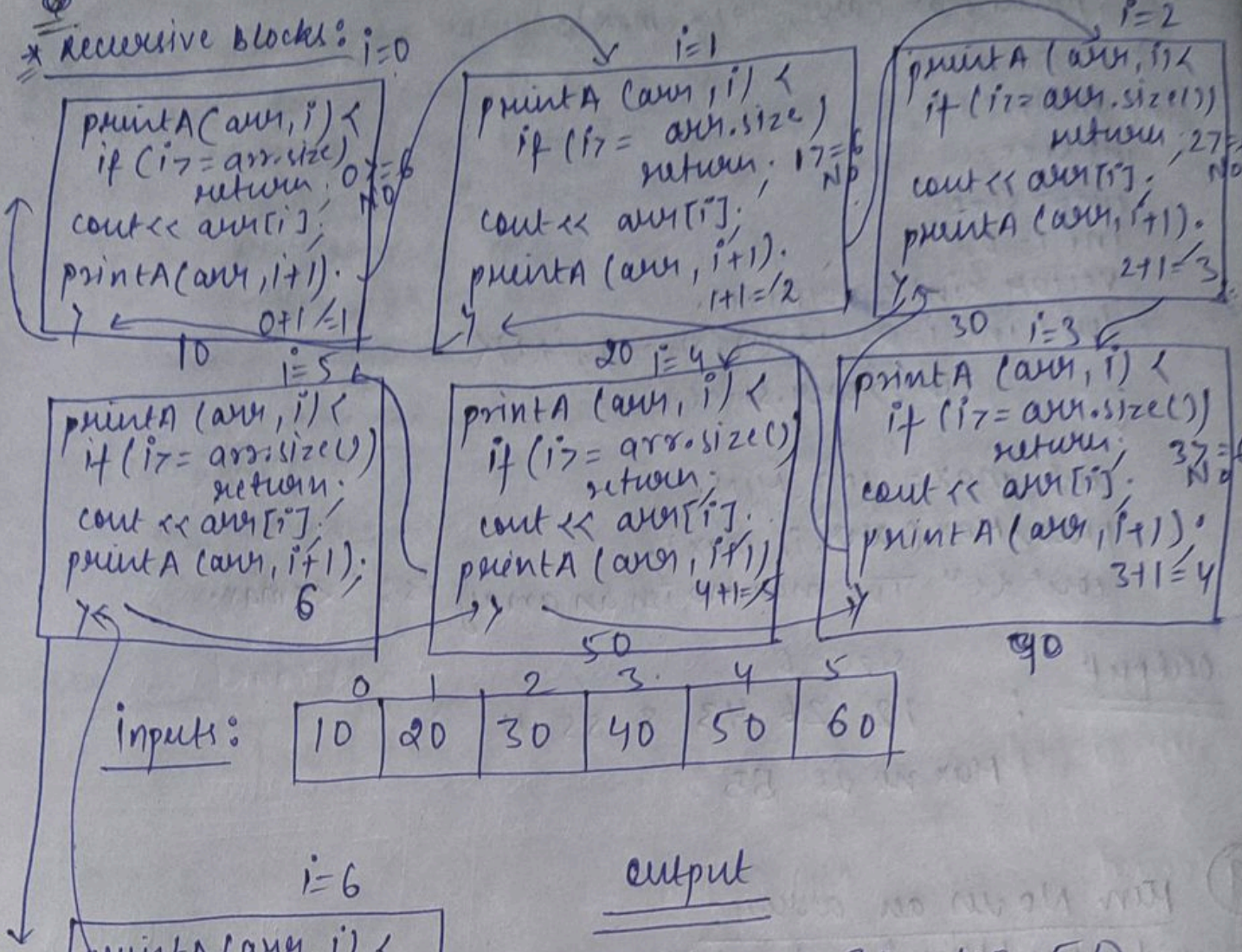
```
int main(){
  //same
}
```

Output: 10 20 30 40 50 Input
50 40 30 20 10 → Reverse Array.

Recursive tree:



* Recursive blocks:



Inputs:

0	1	2	3	4	5
10	20	30	40	50	60

Output

10 20 30 40 50 60

Array same as of Reverse Array.

(8) MAX no in an Array

```
void printMax(vector<int> arr, int i, int &max){
    // base case
    if (i >= arr.size()){
        return;
    }
    // processing
    if (arr[i] > max){
        max = arr[i];
    }
```

```
    //recursive relation.
    printMax(arr, i+1, max);
}
int main(){
    int size.
    cin>>size.
    int i=0.
    vector<int> arr(size).
    for(int i=0. i<arr.size(); i++){
        cin>>arr.size().
}
    int max = INT_MIN;
    printMax(arr, i, max).
    cout<<" The max no in an array is: "<<max;
```

Output : size = 6
12 26 43 8 55 11
Max no is 55.

(9) Min No in an array:

```
void printMin(vector<int> arr, int i, int& min){
    //base case
    if(i>=arr.size()){
        return;
}
    //processing
    if(arr[i] < min){
        min = arr[i];
}
    // recursive relation.
    printMin(arr, i+1, min).
```

```
int main(){
    // same as above
    int min = INT_MAX.
    printMin(arr, i, min).
    cout<<"The min no in an
        array is: "<<min.
}
```

Output

size = 6
12 26 43 8 55 11
Min no is 8

* Imp → why we are using pass by reference in ques 8 and 9 min & max variable when passing it in function

If we don't pass it by reference then max & min becomes local variable for their respective function and through the recursion the previous value of max & min is lost & generate the new value due to which for max it print minimum value of INT & for min it print maximum value of INT which we don't want. By pass by reference it won't create the copy of max/min variable ∴ whatever the changes will be made inside the original min/max variables.

take an example

0	1	2	3	4	5
12	26	43	8	55	11

To find min value in an array and we don't use pass by reference.

0 7=6 No
arr[0] < INT_MAX
min = 12

f(arr, 0+1, min)
mini = min(mini, arr[0]) = 12

f(arr, 1, 12)
mini = min(mini, arr[1]) = 12

f(arr, 2, 12)
mini = min(mini, arr[2]) = 12

f(arr, 3, 12)
mini = min(mini, arr[3]) = 8

f(arr, 4, 8) mini = min(mini, arr[4]) = 8

f(arr, 5, 8) mini = min(mini, arr[5]) = 8

f(arr, 6, 8)

It will print INT_MAX as minimum value.

All values will be lost

(10) Check key in a string:

```
bool checkKey (string &word, int i, char &key){
  // base case
  if (i >= word.length()){
      return false;
  }
  // processing
  if (word[i] == key){
      return true;
  }
  // recursive relation
  checkKey (word, i+1, key);
}
```

```
int main(){
  string word;
  getline (cin, word);
  char key;
  cin >> key;
  int i = 0;
  if (checkKey (word, i, key)){
      cout << "Found";
  }
  else{
      Not found;
  }
```

Output

Subratsingh	subratsingh	
a	P	
Found	Not Found	

(11) Check key found at what index through Recursion:

```
int checkKeyIndex (string &word, int i, char &key){
  if (i >= word.length()){
      return -1;
  }
  if (word[i] == key){
      return i;
  }
  checkKeyIndex (word, i+1, key);
}
```

```
int main(){
  // same above
  int index =
      checkKeyIndex
          (word, i, key);
  cout << index;
```

Output:

subratsingh
i → key
7 → index

(12) check the no. of occurrence of key in a string

```
int checkkeycount (int &word, int i, char &key, int count){
    if (i >= word.length()){
        return count;
    }
    if (word[i] == key){
        count++;
    }
    checkkeycount (word, i+1, key, count);
}
int main (){
    // same as previous
    int i=0, count=0;
    int freq = checkkeycount (word, i, key, count);
    cout << freq.
```

Output

sssubratsinghs
s → key
5 → count

(13) check the key & store it any vector:

```
void checkkey (string &word, int i, char key, vector<int> &ans){
    if (i >= word.length()){
        return;
    }
    if (word[i] == key){
        ans.push_back(i);
    }
    checkkey (word, i+1, key, ans);
}
int main (){
    // same
    int i=0;
    vector<int> ans;
    checkkey (word, i, key, ans)
    for (int j=0; j < ans.size(); j++){
        cout << ans[j] << " ";
    }
```

Output

sssubratsinghs

0 1 2 8 13

In above ques we are passing vector as pass by reference bcz due to again calling of function the vector becomes empty for every function call due to which it doesn't display during final output ∴ using of reference we'll add elements to vector. i.e original vector during every time call. value will add to original vector. Thatswhy we use pass by references.

(14) Print digit Recursion

```
void printDigits(int num){
    if(num==0){
        return;
    }
    int rem = num%10;
    cout<<rem;
    printDigits(num/10);
}
//TAIL Recursion
```

```
int main(){
    int num;
    cin>>num;
    if(num==0){
        cout<<0;
    }
    printDigits(num);
}
```

Output = 687 → Input

786 → Digits in reverse

(15) Print Digits same Order

```
void printDigits(int num){
    if(num==0){
        return;
    }
    printDigits(num/10);
    //HEAD Recursion
    int rem = num%10;
    cout<<num;
}
```

```
int main(){
    int num;
    cin>>num;
    if(num==0){
        cout<<0;
    }
    printDigits(num);
}
```

Output = 687 → input

687 → digits in same order