Basic keywords & concepts

Page No.
Date

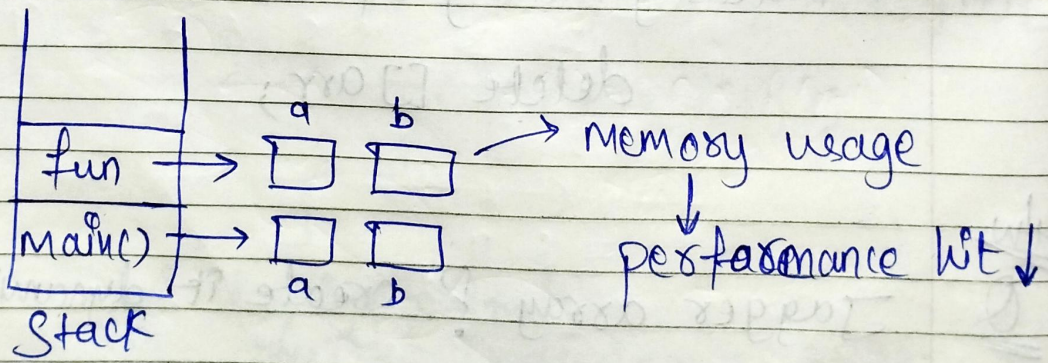
(*) **Macros:** A piece of code in a program that is replaced by value of macro

Eg:- #define Pi 3.14

Macros are used to reduced the space complexity and to maintain the accessibility in program.

(*) **Note:** Never use global variable, instead use reference variable because global variable can be changed easily.

(*) **Inline function:** are used to reduce the function call overhead.



to reduce performance bit, we use getline fun.

Page No.
Date

Note: Inline function only works if the function body ≤ 1.

Eg:-
```
inline int getMax(int& a, int &b){
    return (a>b)? a:b;
}
int main(){
    int a=1, b=2;
    int ans=0;

    ans = getMax(a,b);
    cout << ans << endl;

    a=a+3;
    b= b+1;

    ans = getMax(a,b);
    cout<< ans << endl;

    return 0;
}
```

Note: getMax is Exchanging with (a>b)? a:b if it is inline function.

Note: reduce time complexity.

Phase-2

Page No.
Date

(#) **Recursion**: When a function call itself either directly or indirectly.

→ **Base Case** : ✓ **Note**: Return is mandatory

→ **Recursive Relation**: **Note**: Mandatory.

Eg:- $5! = 5 \times 4!$

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1!$

$1! = 1 \times 0!$

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

Recursive relation

```
if(n==0){ return 1;}
return n * fact(n-1);
```

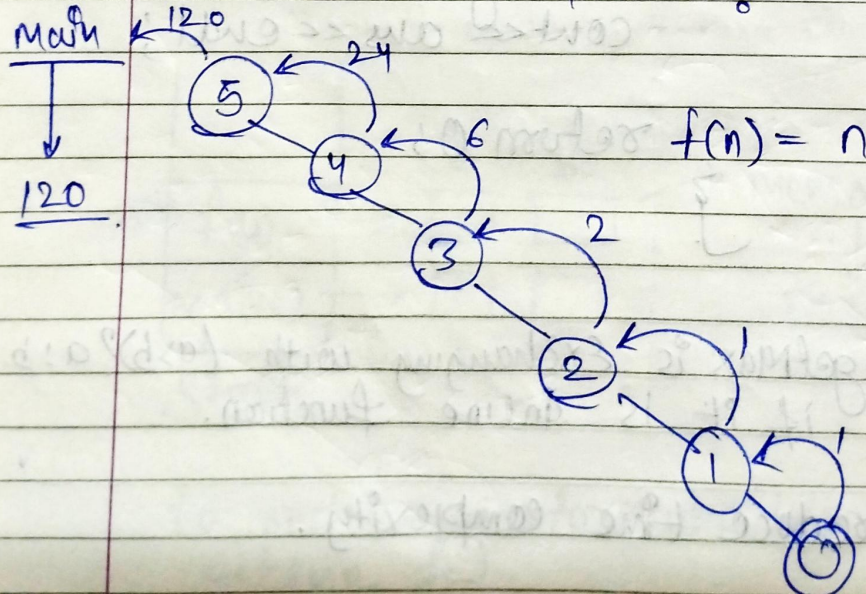
Base case

Time complexity = O(n)

Space complexity = O(n)

Eg:

Recursion tree for 5!



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

Page No.
Date

Structure:

```
fun () {
    Base case;
    Processing;
    Recursive Relation;
}
```

→ Tail Recursion
(R.R at last)

```
fun() {
    Base case;
    Recursive Relation;
    Processing;
}
```

→ Head Recursion
(R.R before processing)

Eg:- find 2^n

$2^5 = 2 * 2^4$

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

BP SP

```
int power(int n){
    // base case
    if(n==0){
        return 1;
    }

    // recursive relation
    int smallproblem = power(n-1);
    int bigproblem = 2 * smallproblem;

    return bigproblem;
}
```

Page No.
Date

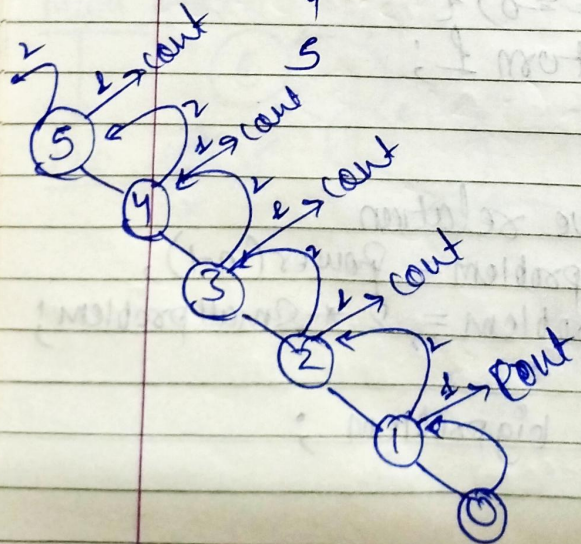
Eg!

Head Recursion

```
void print(int n){

  // base case
    if(n==0){
      return;
    }

  // Recursive relation
    print(n-1);

  cout<<n<<endl;
}
```

O/P ⇒ n=5

1
2
3
4
5

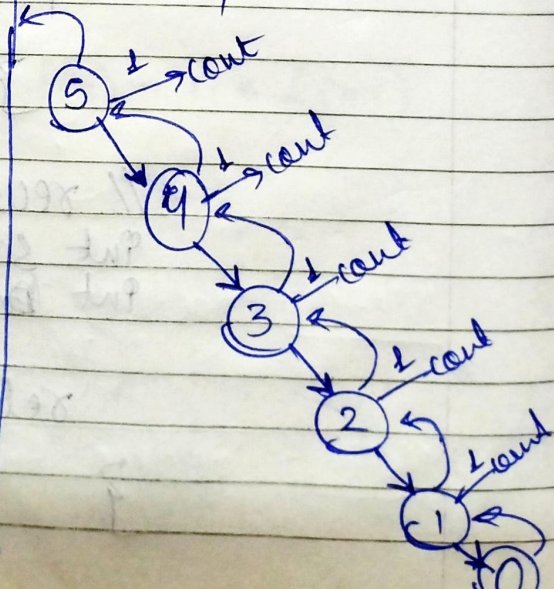


tail Recursion

```
void print(int n){

  // base case
    if(n==0){
      return;
    }

  cout<<n<<endl;
  // recursive rel"
  Print(n-1);
}
```

O/P ⇒ n=5

5
4
3
2
1



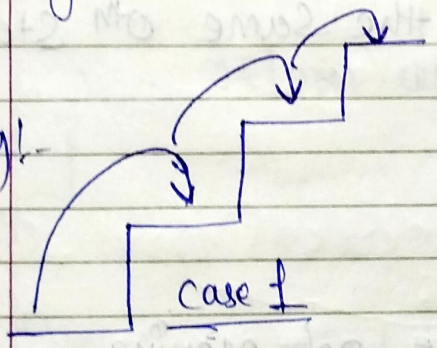
Page No.
Date

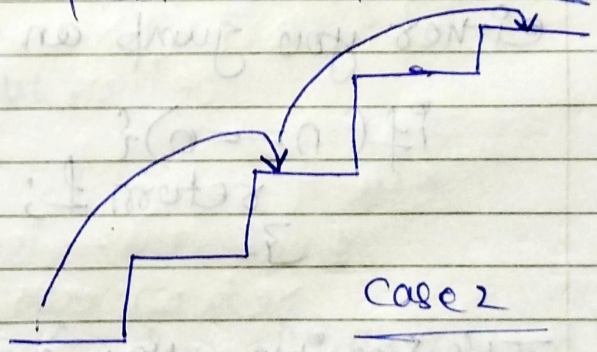
Note: In Recursion, we only need to solve a single case, else case will be solved automatically.

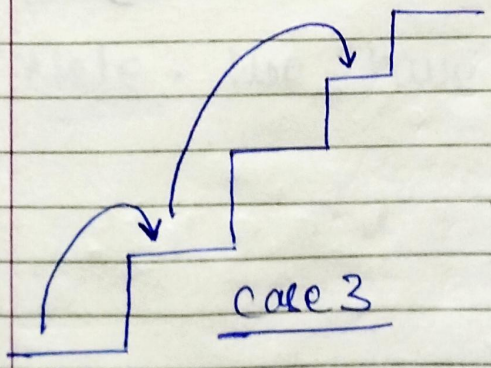
Imp. ****

Q. You have been given a number of stairs. Initially, you are at the 0th stair and you need to reach the Nth stair. Each time you can either climb one step or two steps. You are supposed to return the number of distinct way in which you can climb from 0th to Nth stair.

Eg:-



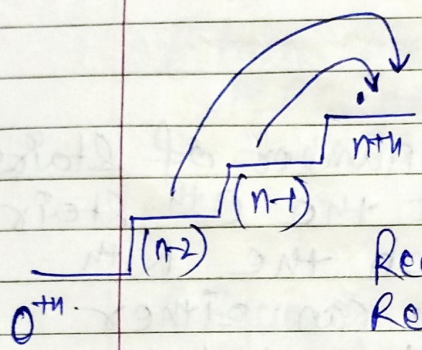




Page No.
Date

Explanation :

There are only 2 possible ways to reach the n^{th} stair.



either from (n-1) or (n-2)

Recursive Relation :- $f(n) = f(n-1) + f(n-2)$

Base case :-

either you jump on the same 0^{th} stair.

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

It's possible, you are not moving

```
if (n<0) {
    return 0;
}
```

Page No.
Date

Code:

```
int countDisplayStair(long long nstairs){

    // base case
    if(nstair<0)
        return 0;

    if(nstairs==0)
        return 1;

    // Recursive relation

    int ans= countDisplaystair(nstair-1) + countDisplaystair(nstair-2);

        return ans;
}
```

Problem: TLE (Time limit exceed)

Note: we have to use dp in this.

Page No.
Date

Sorting

Binary Search using Recursion

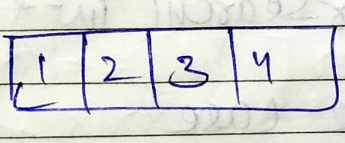
```
#include <iostream>
using namespace std;

bool isSorted(int *arr, int size){

    // base case
    if(size==0 || size==1){
        return true;
    }

    if(arr[0] > arr[1])
        return false;

    else{
        bool ans = isSorted(arr+1, size-1);
        return ans;
    }
}

int main(){

    int arr[5]={2,4,9,9,10};
    int size=5;

    bool temp = isSorted(arr, size);
```

Page No.
Date

```
if(temp){
    cout<<" Array is sorted"<<endl;
  }
else{
    cout<<" Array is not sorted "<<endl;
    }

return 0;
}
```

Ques.

Find the sum of the array elements using the Recursion.

Eg:- 

=> Sum = 10.

Sol^n

```
int getSum( int *arr, int size){

  // base case
  if(size==0){
      return 0;
    }

  if(size==1){
      return arr[0];
    }
```

Page No.
Date

```
int remainingPart = getSum(arr+1, size-1);
int Sum = arr[0] + remainingPart;
return Sum;
}
```

}

Ans. (#)

Q find the linear search using Recursion?

Sol^n

```
#include<iostream>
using namespace std;
bool LinearSearch(int* arr, int size, int k) {
    // base case
    if(size == 0)
        return false;
    if(arr[0] == k)
        return false;
    else{
        bool remainingPart = Linearsearch(arr+1, size
        return remainingPart;
    }
```

Page No.
Date

```
}
int main()
{
    int arr[5] = {3,5,9,2,1};
    int size = 5;
    int key = 2;

    bool ans = Linear Search( arr, size, key);

    if (ans) {

    cout<< "present" <<endl;
      }
    else {
      cout<< " NOT present "<<endl;
      }

{
  return 0;
}
```

```
+1, size-1, k);
```

Page No.
Date

Binary search in Recursion.

```
#include <iostream>
using namespace std;

bool binarySearch(int *arr, int s, int e, int k){
    // base case
    // element not found.
    if(s>e)
        return false;

    // element found
    int mid = s+(e-s)/2;

    if( arr[mid] == k)
        return true;

    if( arr[mid] < k){
    return binarySearch( arr, mid+1, e, k);
    }
    else{
    return binarySearch( arr, s, mid-1, k);
    }
}
```

Page No.
Date

```
int main(){
    int arr[5] = {5,9,14,18,21};
    int s=0;
    int e = arr.size()-1;
    int key = 18;
{
cout<<" present or not"<<endl;

    bool ans = binarySearch(arr,s,e,k);
if(ans){
    cout<<" Element present"<<endl;
    }
else{
    cout<<" Not present"<<endl;
    }
return 0;
}
```

Time complexity : = O(logn)

Space complexity = O(logn)

Page No.
Date

Q. Reverse a String using Recursion

```
#include <iostream>
using namespace std;

void reverse(string& str, int i, int j){

    // base case
    if(i>j)
        return;

    swap(str[i], str[j]);
    i++;
    j--;

    // Recursive call
    reverse(str, i, j);
}

int main(){

    string name = "TUSHAR";

    reverse(name, 0, name.length()-1);

    cout<< name<< endl;

    return 0;
}
```

Page No.
Date

imp ****

Q. Check for the palindrome.

```
#include <iostream>
using namespace std;

bool checkPalindrome (string str, int i, int j)
{
    // base case
    if (i>j)
        return true;

    if ( str[i] != str[j])
        return false;

    else {
        // Recursive call
        return checkPalindrome(str,
                               i+1, j-1);
    }
}

int main() {
    string name = "abbccbba";
    cout<<endl;

bool isPalindrome = checkPalindrome (name,
                    0, name.length() - 1);

if( isPalindrome)
    cout<<" It is palindrome";
else
    cout <<" Not Palindrome"; }
```

Page No.
Date

Q WAP for a^b using Recursion.

Hint: a^b → even $(a^{b/2} * a^{b/2})$
a^b → odd $(a * (a^{b/2} * a^{b/2}))$

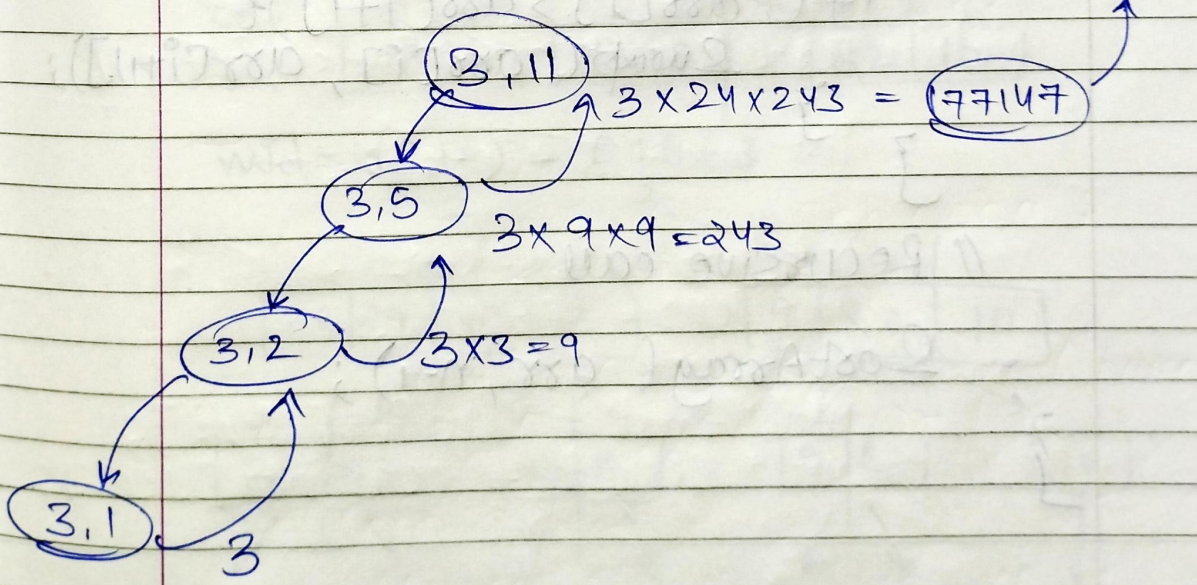
Sol^n

```
#include<iostream>
using namespace std;

int power(int a, int b){

    // base case
    if(b==0)
        return 1;

    if(b==1)
        return a;

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

    // if b is even
    if(b%2==0){
        return ans*ans;
    }

    else{
        // b is odd
        return a*ans*ans;
    }
```

Page No.
Date

```
}
int main()
{
    cout<<" Enter the Number";
    cin>> a;
    cout<<" Enter Power" ;
    cin>> b;

  int ans = Power(a,b);

    cout<<" Answer is" << ans;
    cout<<endl;

    return 0;
}
```

Recursive Tree:- a=3 b=11 3^{11} = ?



Page No.
Date

(11) Bubble Sort using Recursion

```
#include<iostream>
using namespace std;

void SortArray(int *arr, int n){

    // base case - already sorted.
    if(n==0 || n==1){
        return;
    }

    // 1st case - Put larget no. on its place.
    for(int i=0; i<n-1; i++){
        if(arr[i] > arr[i+1]){
            Swap(arr[i], arr[i+1]);
        }
    }

    // Recursive call
    SortArray(arr, n-1);
}
```

Page No.
Date

```
int main()
{
    int arr[5] = { 2,5,1,6,9};
    SortArray(arr, 5);
    for(int i=0; i<5; i++){
        cout<< arr[i]<<"  ";
    }
    return 0;
}
```

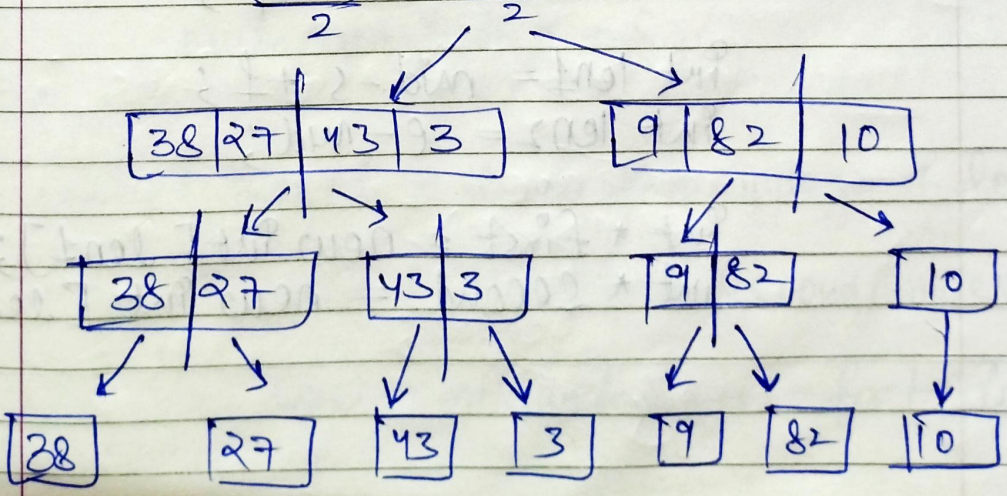
imp.
*****

Merge Sort : used to sort an array.

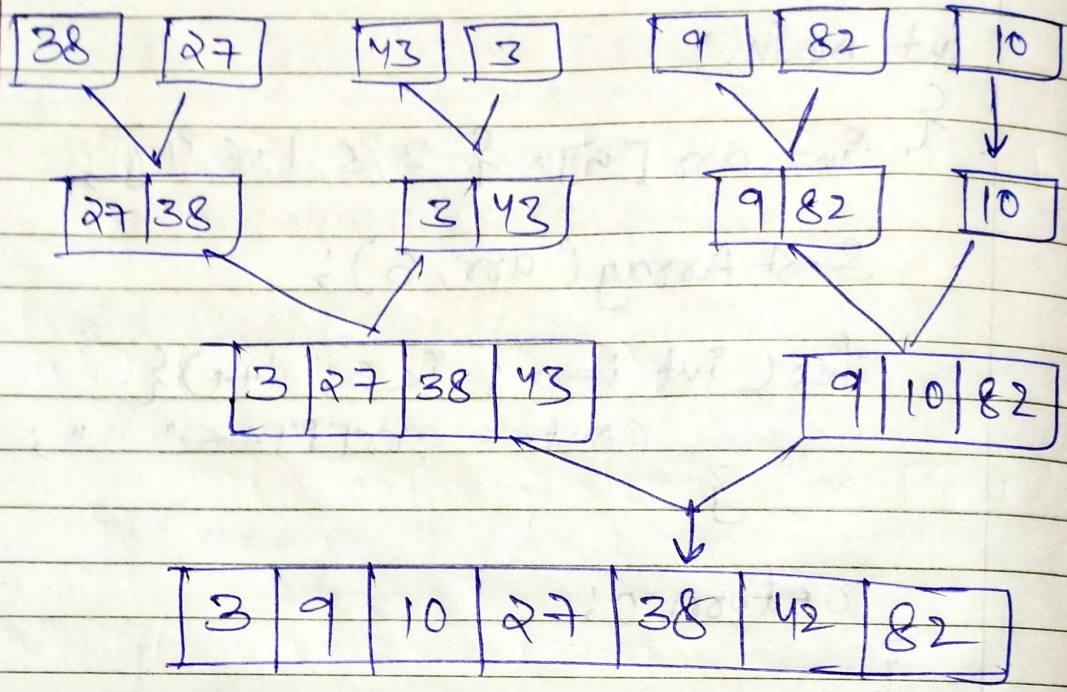
Eg :-

0	1	2	3	4	5	6
38	27	43	3	9	82	10

$mid = \frac{(s+e)}{2} = \frac{0+6}{2} = 3.$



Page No.
Date



Code :

```
#include <iostream>
using namespace std;

void merge( int *arr , int s , int e){
    int mid = s +(e-s)/2;

    int len1 = mid-s +1;
    int len2 = e-mid;

    int * first = new int[len1];
    int * second = new int [len2];
```

Page No.
Date

// Copy value to the arrays.

```
int mainArrayIndex = s;
for(int i=0; i<len1; i++){
    first[i] = arr[mainArrayIndex++];
}
mainArrayIndex = mid+1;
for(int i=0; i<len2; i++){
    second[i] = arr[mainArrayIndex++];
}
```

// merge two sorted array

```
int index1 = 0;
int index2 = 0;
mainArrayIndex = s;
while(index1 < len1 && index2 < len2){
    if(first[index1] < second[index2]){
        arr[mainArrayIndex++] = first[index1++];
    }
```

```
        else{
            arr[mainArrayIndex++] = second[index2++];
        }
    }
    while( index1 < len1){
        arr[mainArrayIndex++] = first[index1++];
    }
    while( index2 < len2){
        arr[mainArrayIndex++] = second[index2++];
    }

    delete[] first;
    delete[] second;
}

void mergeSort( int * arr, int s, int e){

    // base case
        if(s>=e){
            return;
        }
```

Page No.
Date

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

// left part sort
mergesort(arr, s, mid);

// Right part sort
mergesort(arr, mid+1, e);

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

int main(){

int arr[5] = {57, 32, 4, 19, 102};

int n=5;

mergeSort(arr, 0, n-1);

for(int i=0; i<n; i++){
    cout<< arr[i] << " ";
}
cout<<endl;

return 0;

}
```

Time complexity : O(nlogn)
Space complexity : O(n)

Page No.
Date

Int

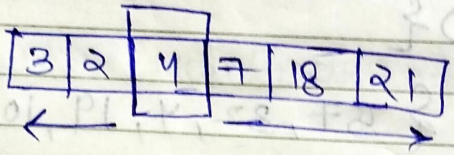
(11) Quick Sort →

Explanation:

Eg:-

0	1	2	3	4	5
4	7	3	18	21	2

Place first element at its right place so, that left side of that element has smaller elements and right side has larger elements.



Now use recursion to solve QuickSort.

(i) Partition
(ii) Recursion.

Code:

```
#include<iostream>
using namespace std;
int partition (int* arr, int s, int e)
{
    int pivot= arr[s];
```

```
int cnt = 0;
for(int i = s+1 ; i<=e ; i++){
    if( arr[i] <= pivot){
        cnt++;
    }
}
```

// place pivot at right position

```
int pivotIndex = s + cnt;
swap( arr[pivotIndex] , arr[s]);
```

// left and right place

```
int i = s, j = e;

while( i < pivotIndex && j > pivotIndex)
{
    while ( arr[i] <= pivot)
        { i++;
        }
    while ( arr[j] >= Pivot)
        {
        j--;
        }
```

Page No.
Date

```
if (i < pivotIndex && j > pivotIndex){
    swap(arr[i++], arr[j--]);
  }
}

  return pivotIndex;
}

void quickSort(int* arr, int s, int e){

    // base case
    if(s >= e){
        return;
    }

  // partition
  int p = partition(arr, s, e);

  // sorting left part
   quickSort(arr, s, p-1);

  // sorting Right part
    quickSort(arr, p+1, e);

}
```

Page No.
Date

```
int main()
{
    int arr[5] = {4,7,18,21,2};
    int n=5;
    quickSort(arr, 0, n-1);
    for(int i=0; i<n; i++){
        cout<< arr[i] <<"  ";
    } cout<<endl;

    return 0;
}
```