LEARN DSA WITH C++

WEEK :: 06

LEARN DSA WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

My profile

LEARN DSA WITH C++

WEEK :: 06 **DAY: 01** **DATE: 22-05-2023**

MERGE SORT & QUICK SORT

RECURSION

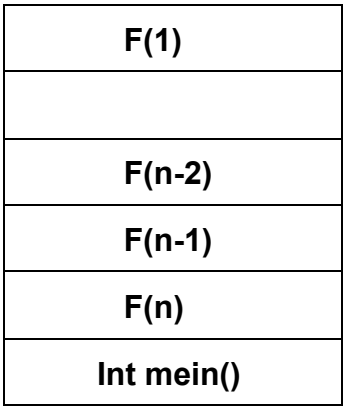
GYM MAN => EAT, GYM, SLEEP & REPEAT :: Actually it is also Recursion.
Because we repeat the same pattern every day.

Here Base case :: 10kg weight loss/gain
Injury
Death

Simple meaning of Recursion :: We just handle present case/present day, other cases automatically handle it.

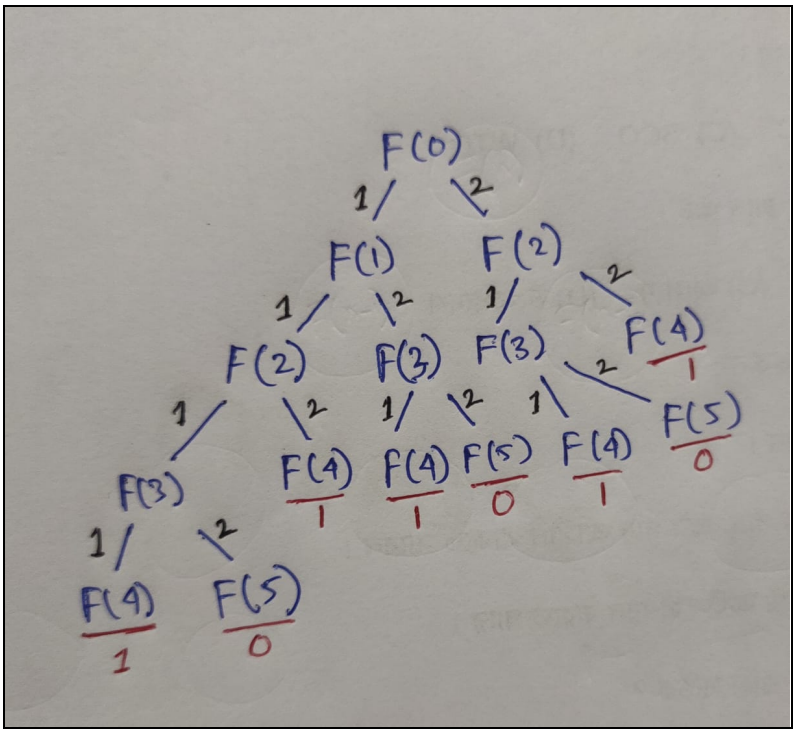
FIBONACCI SERIES :: RECURSION :: TIME COMPLEXITY

Highest number of branch in tree it's consider total number TC



Example:: Reach nth position, take step at a time 1 & 2.

Consider n = 4;
How many way reach ::
f(1) = 1 + 1 + 1 + 1 =4
f(2) = 1 + 2 + 1 =4
f(3) = 2 + 1 + 1 =4
f(4) = 1 + 1 + 2 =4
f(5) = 2 + 2 =4



When We reach 4 positions then we return 1 and other times return 0.
Time Complexity :: highest length of tree = O(n+1)

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

MERGE SORT

4	3	7	2	8	1	9	5

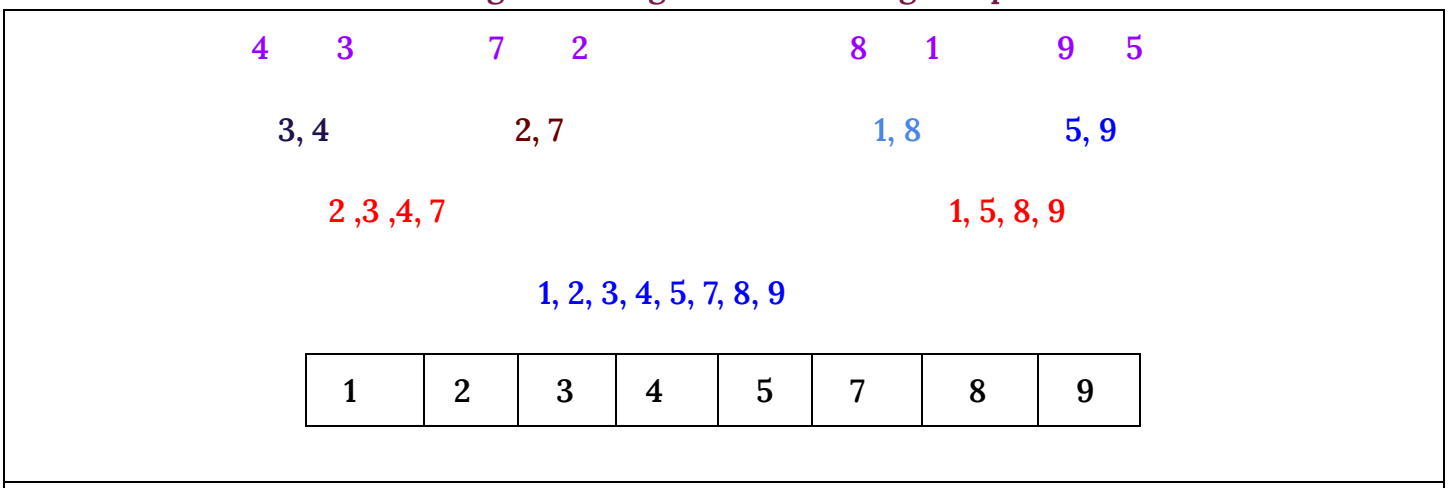
Divided two part

4, 3, 7, 2 **8, 1, 9, 5**

4, 3 **7,2** **8, 1** **9, 5**

4 **3** **7** **2** **8** **1** **9** **5**

Sorting ascending order according two part



Divide Array in 2 parts.
Divide it, Until its size becomes 1.
Then merge the both parts in sorted order.

#Code ::

MERGE SORT

```
class Solution
{
    public:
    void merge(int arr[], int l, int m, int r)
    {
        // Your code here
        int *ans = new int[r-l+1];
        int first = l, second = m+1, pos = 0;
        while(first <=m && second <=r)
        {
            if(arr[first]<=arr[second])
            ans[pos++] = arr[first++];
            else
            ans[pos++] = arr[second++];
        }

        while(first<=m)
        ans[pos++]= arr[first++];

        while(second<=r)
        ans[pos++] = arr[second++];

        pos =0, m=l;
        while(m<=r)
        arr[m++] = ans[pos++];

        delete []ans;
    }
    public:
    void mergeSort(int arr[], int l, int r)
```

```
    {
        //code here
        if(l==r)
        return ;
        int mid = l +(r-l)/2;
        mergeSort(arr, l, mid);
        mergeSort(arr,mid+1, r);
        merge(arr,l, mid, r);
    }
};
```

QUICK SORT

3	1	5	6	2	4

Take from last

Which one biggest compere me, Ceme behind me And Which smallest compare me, come in front me

small<Me<Big

3	1	2	4	5	6

Repeat same process but do individually

1<2<3 **5<6**

1	2	3	4	5	6

Last element of the array will be put in its correct position.
Values less than pivot go left and others go right.
Then visit left part and right part , then again repeat

LEARN DSA WITH C++

WEEK :: 06 **DAY: 02** **DATE: 23-05-2023**

QUICK SORT + Recursion

Example:- 01

Array :	5	3	1	2	4	
Index :	0	1	2	3	4	First choose Pivot element : 4 (last Element)
	3	1	2	4	5	4<5
	1	2	3	4	5	1<2 single elem all ready shorted [5]

Example:-02

4	7	3	8	1	2	5	9	6	choose pivot element = 6
4	3	1	2	5	6	7	8	9	smallest elem < 6 < Biggest elem two part mid = 6
4	3	1	2	5	6	7	8	9	pivot = 5; || similarly pivot = 9
1	2	4	3	5	6	7	8	9	1 single elem; || pivot = 8
1	2	3	4	5	6	7	8	9	pivot elem = 3 || 7 = single elem

** NOTES

1. Put the pivot element in its correct position.
2. Visit left of the array.
3. Visit the right part of the array.

HOW IT'S WORK :

4	7	1	8	1	7	5	9	6

Take two pointer :
First pointer : where value Fill
Second pointer : Traverse the whole array

Condition ::

```
        (Arr[second point ] <= Pivot)
        {
            Swap (arr[first], arr[second];
                  First ++; second ++;
        }
        Else
                Second ++;
```

Quicksort Code

```
Quicksort(int arr[ ],  int start, int end)
{
       if(start>= end)
          return;

       Int index = Partition(arr, start, end);
       Quicksort (arr, start, index-1);
       Quicksort (arr, index+1, end)
}
```

Int index = Partition(arr, start, end)

```
Int  partition (int *arr, int start, int end)
{
    Firstpointer = start;
    Secondpointer =start;

    while(second pointer <=end)
      {
          if(arr[end] >= arr[second])
            swap(arr[second++], arr[first++])
          else
             second ++;
}
    return  firstpointer -1;
```

 # Time Complexity :

1. **O(N^2) = when the array has sorted.**
2. **O(NlogN) = when the array has not been sorted.**

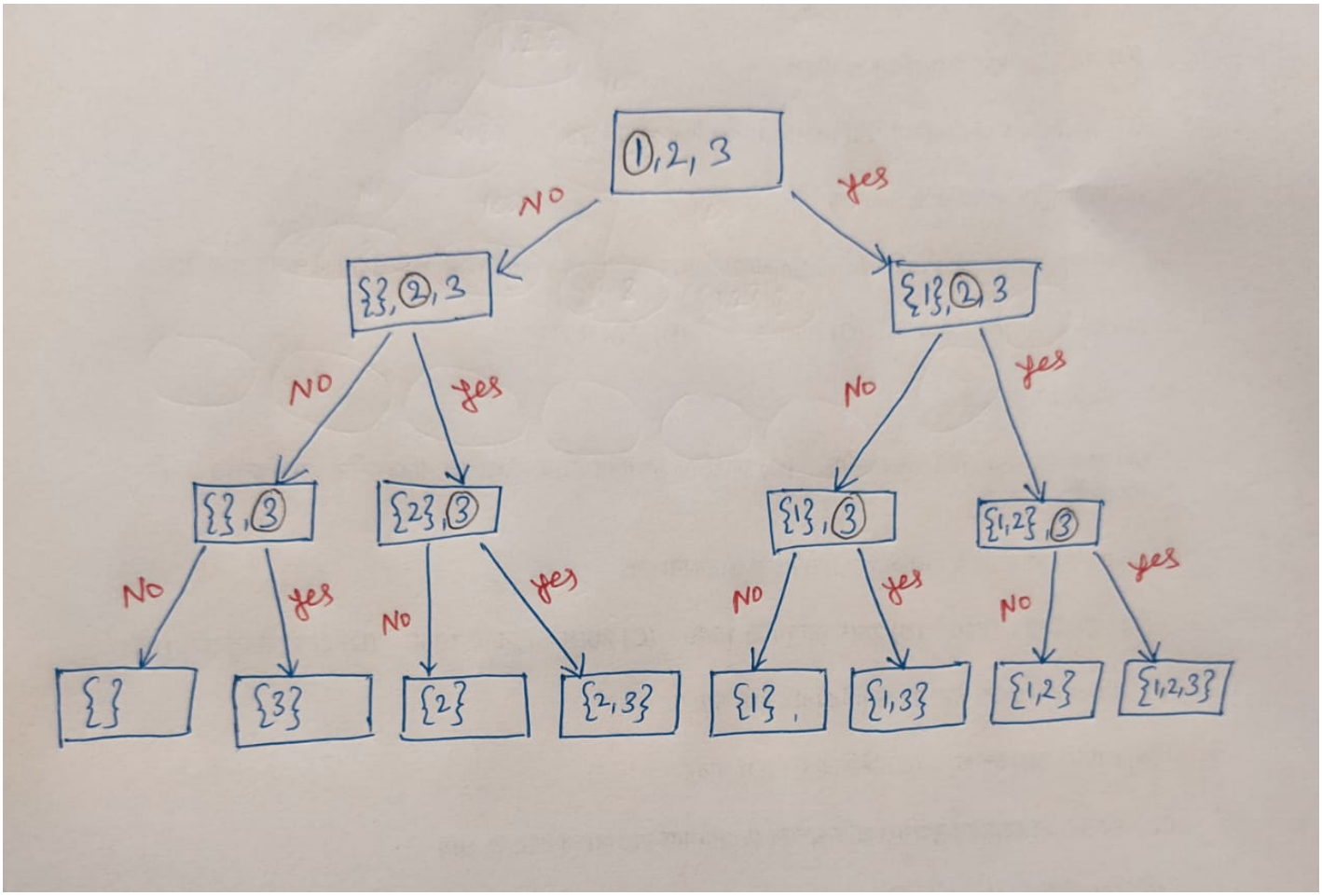
Space Complexity :

1. O(N) = array sorted.

**Important ::

Sorted algorithm	Insert	Bubble	Selection	Merge	Quick
Time Complexity	O(N^2)	O(N^2)	O(N^2)	O(NlogN)	O(N^2)
Space Complexity					

Possible combination of element : 2^3=8



Find sum = 20; given array; take any Combination

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

bool sum_pos(int arr[], int size, int sum, int index, int total)
```

```
{
    if(index==size)
    {
        if(sum==total)
        return 1;
        else
        return 0;
    }

    return sum_pos(arr, size, sum, index+1, total) || sum_pos(arr, size, sum, index+1,
total+ arr[index]);
};
int main()
{
    int arr[6] = {2, 5, 1, 6, 7, 11};
    int sum = 20;
    int total = 0;
    cout<<sum_pos(arr, 6, sum, 0, total);
    return 0;

    // 1 means yes it is possible
}
```

LEARN DSA WITH C++

WEEK :: 06 **DAY**: 03 **DATE**: 24-05-2023

Recursion + Robber House

House Robber <<LeeTCode>>

```
Chori(int arr[], int n, int & sum, int total, int index)
{
     if(index>=n)
     {
        Sum = max(sum, total)
        return ;
      }
    Chori(arr, n, sum, total, index)
    Chori(arr, n, sum, total + arr[index], index + 2);
}
Int main()
{
   Int n;
   Int arr[n];
   Cin >> n;
   Int total =0;
   Chori(arr, n, sum, total, 0(index));
   Cout << sum;
}
```

House Robber II << LeetCode>>

Combination Sum << Leetcode >>

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

void sum_possible(int *arr, int size, int index, int sum)
{
    if(index == size)
    {
        cout<<sum<<" ";
```

```
        return;
    }


    sum_possible(arr, size, index+1, sum);
    sum_possible(arr, size, index+1, sum + arr[index]);
}
int main()
{
    int arr[5] = {2, 4, 1, 5, 8};
    int index = 0, sum = 0;
    sum_possible(arr, 5, index, sum);
    return 0;
}
```

Combination Sum with target

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

void sum_possible(int *arr, int size, int index, int sum, int target, int &ans)
{
    if(index == size)
    {
        if(sum==target)
        {
            ans = 1;
        }
        return;
    }


    sum_possible(arr, size, index+1, sum, target, ans);
    sum_possible(arr, size, index+1, sum + arr[index], target, ans);
}
int main()
{
    int arr[5] = {2, 4, 1, 5, 8};
    int index = 0, sum = 0;
    int target = 15;
    int ans = 0;
    sum_possible(arr, 5, index, sum, target, ans);
    cout<<ans;
    return 0;
```

Arr : {3, 2, 7} take target = 8, you can use elem repeatedly || Combination Sum

```
Void sum_possible(int arr, int size, int index, int total, int sum, int & ans)
{
   if(index==size)
    {
      if(sum==total)
         ans = 1;
       Return;
     }
   if(sum > target)
     return;
   sum_possible(arr, size, index+1, total, sum, ans)
   sum_possible(arr, size, index, total, sum + arr[index], ans)
}
```

Target = 100; take 1^2, 2^2, …… , sum of any elem(power) = 100

```
sum_possible(int n, int num, int power, int total, int & cout)
{
   if(total == num)
     {
       cout ++; return;
       if(total> num)
        return;
      }

sum_possible(n+1; num; power, total, cout)
sum_possible(n=1, num, power, total + pow(n, power), count)
}
```

LEARN DSA WITH C++

WEEK :: 06 **DAY: 04** **DATE: 25-05-2023**

Recursion + Palindrome + Permutation

String : {.....}, print all possible combination [continuous way]

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

void print(string s, int index, string ans)
{
    if(ans.size())
    cout<<ans<<" ";

    if(index==s.size())
    return;

    if(ans.size()==0)
    print(s, index+1, ans);

    print(s, index+1, ans + s[index]);
}
int main()
{
    string s;
    cin>>s;
    print(s,0,"");

return 0;
};
```

Combination Sum <<LeetCode>>

```cpp
class Solution {
public:

    void find(vector<int>&candidates,vector<vector<int>>&ans, vector<int>temp, int sum, int target, int index)
    {
        if(index==candidates.size())
        {
            if(sum==target)
            ans.push_back(temp);

            return;
```

```
        }

        if(sum>target)
        return;

         find(candidates, ans, temp, sum, target, index +1);
         sum+=candidates[index];
         temp.push_back(candidates[index]);
         find(candidates, ans, temp, sum, target, index);
    }


    vector<vector<int>> combinationSum(vector<int>& candidates, int target)
    {
        vector<vector<int>>ans;
        vector<int>temp;
        int sum = 0;
        find(candidates, ans, temp, sum, target, 0);
        return ans;
    }
};
```

Print all unique possible combination given string || 'abc'

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

void print(string s, int index)
{
    if(index==s.size()-1)
    {
        cout<<s<<" ";
        return;
    }

    for(int i=index; i<s.size(); i++)
    {
        swap(s[i],s[index]);
        print(s, index+1);
        swap(s[i], s[index]);
    }
}
int main()
{
    string s;
    cin>>s;
    print(s, 0);
return 0;
};
```

Letter Combinations of a Phone Number << LeetCode >>

```
class Solution {
public:

    void fun(string &digits, vector<string>&answer, vector<string>&mapping, string temp,
int index)
   {
       if(index==digits.size())
       {
           answer.push_back(temp);
           return;
       }
       int pos = digits[index] - '2';
       for(int i=0; i<mapping[pos].size(); i++)
       fun(digits, answer, mapping, temp+ mapping[pos][i], index+1);
   }
    vector<string> letterCombinations(string digits) {
        vector<string>answer;
        if(digits.size()==0)
        return answer;
        vector<string>mapping(8);
        mapping[0]="abc";
        mapping[1]="def";
        mapping[2]="ghi";
        mapping[3]="jkl";
        mapping[4]="mno";
        mapping[5]="pqrs";
        mapping[6]="tuv";
        mapping[7]="wxyz";
        fun(digits, answer, mapping, "", 0);
        return answer;
    }
};
```