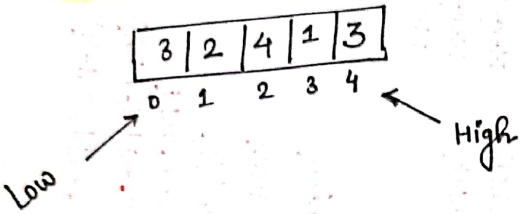
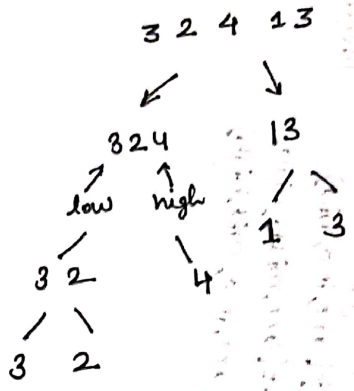
Data Structure and Algorithms

By

Malay Tripathi

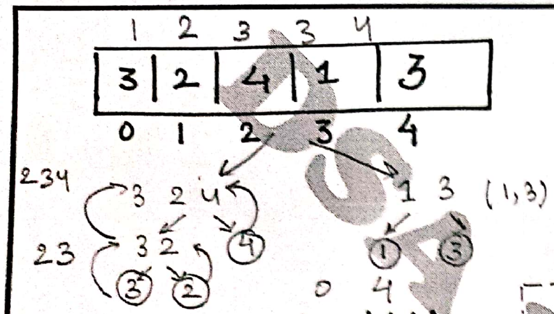
MERGE SORT






```
Merge Sort (arr, low, high)
{  if (low >= high) return ; ] Base condition.
 mid = (low + high) / 2
 merge sort (arr, low, mid)
 merge sort (arr, mid+1, high)
 merge ( arr, low, mid, high)
```

RECURSION OF MERGE SORT




```
mergesort (arr, low, high)
{
  if ( low ≥ high) return;
  mid= ( low+high) /2;    0+4/2=2
  MergeSort (arr, low, mid);
  Merge Sort ( arr, mid+1, high);
  merge( arr, low, mid, high);
}
```

```
0 2
mergesort (arr,low,high)
{
  if ( low ≥ high) return;
  mid= (low+high)/2;  -1
  Mergesort (arr, low, mid)
  Mergesort (arr, mid+1, high)
  Merge(arr, low, mid, high)
}
```

2,3   4

2,3,4

```
0 1
merge sort (arr, low, high)
{
  if(low ≥ high) return;
  mid=(low+high)/2;
  MergeSort(arr, low,mid)
  MergeSort(arr, mid+1, high)
  Merge(arr, low, mid, high)
}
```

3   2

2,3

```
0 0
merge sort (arr, low, high)
{
  if(low ≥ high) return
  mid = (low+high)/2
  Merge Sort( arr, low, mid)
  Merge Sort(arr, mid+1, high)
  Merge(arr, low, mid, high)
}
```

```
3, 4
merge sort (arr, low, high)
{
  if (low ≥ high) return
  mid= (low+high)/2
  mergesort (arr, low, mid)
  mergesort (arr, mid+1, high)
  Merge (arr, low, mid, high)
}
```

```
2 2
mergesort (arr, low, high)
{
  if ( low ≥ high) return
  mid = (low+high)/2
  mergesort (arr, low, mid)
  MergeSort ( arr, mid+1 , high)
  Merge (arr, low, mid, high)
}
```

```
1
mergesort (arr, low, high)
{
  if (low ≥ high) return
  mid = (low+high)/2
  MergeSort (arr, low, mid)
  MergeSort (arr, mid+1, high)
  Merge (arr, low, mid, high)
}
```

```
4 4
merge Sort(arr, low, high)
{
  if(low ≥ high) return
  mid=(low+high)/2
  merge sort (arr, low, mid)
  mergesort (arr, mid+1, high)
  Merge (arr, low, mid, high)
}
```

```
3 3
merge sort (arr, low, high)
{ if ( low ≥ high) return
  mid = (low+high)/2
  mergesort (arr, low, mid)
  mergesort (arr, mid+1, high)
  Merge (arr, low, mid, high)
}
```

temp= 1,1 ,2,2,3,4,4,5,6.

```
* merge (arr, low, mid, high)
    ~~temp→[]~~ temp []
    left= low;
    right= mid+1;
  while (left ≤ mid) && right<= high)
  {
    if (arr[left] ≤ arr[right] )
      temp.add ( arr[left])
      left ++;
  else
    temp.add (arr[right])
    right++;
  }
```

Data Structure and Algorithms
By
MALAY TRIPATHI

MULTIPLE RECURSION CALL:

```
f()                    f()
{                      {

                       f()   ] 2 Times.
                       f()
f()
{                      }
```

FIBONACCI SERIES.

0 1 1 2 3 5 8 13 21 34

0th 1st 2nd 3rd 4th ← ---------- numbers of Fibonacci.

Write Recursive function to tell you the N^{th} Fibonacci Number.

$f(3) = 2$

$f(4) = 3$

$f(5) = f(4) + f(3)$

SIMPLE:

```
f[0] = 0
f[1] = 1
for (i=2 → n)
    f[i] = f[i-1] + f[i-2]
```

Data Structure and Algorithms
By
MALAY TRIPATHI

NOW WITH THE HELP OF RECURSION.

```
f(n)
{
 if (n<=1)
    return n;

 return f(n-1)+f(n-2)
}
```

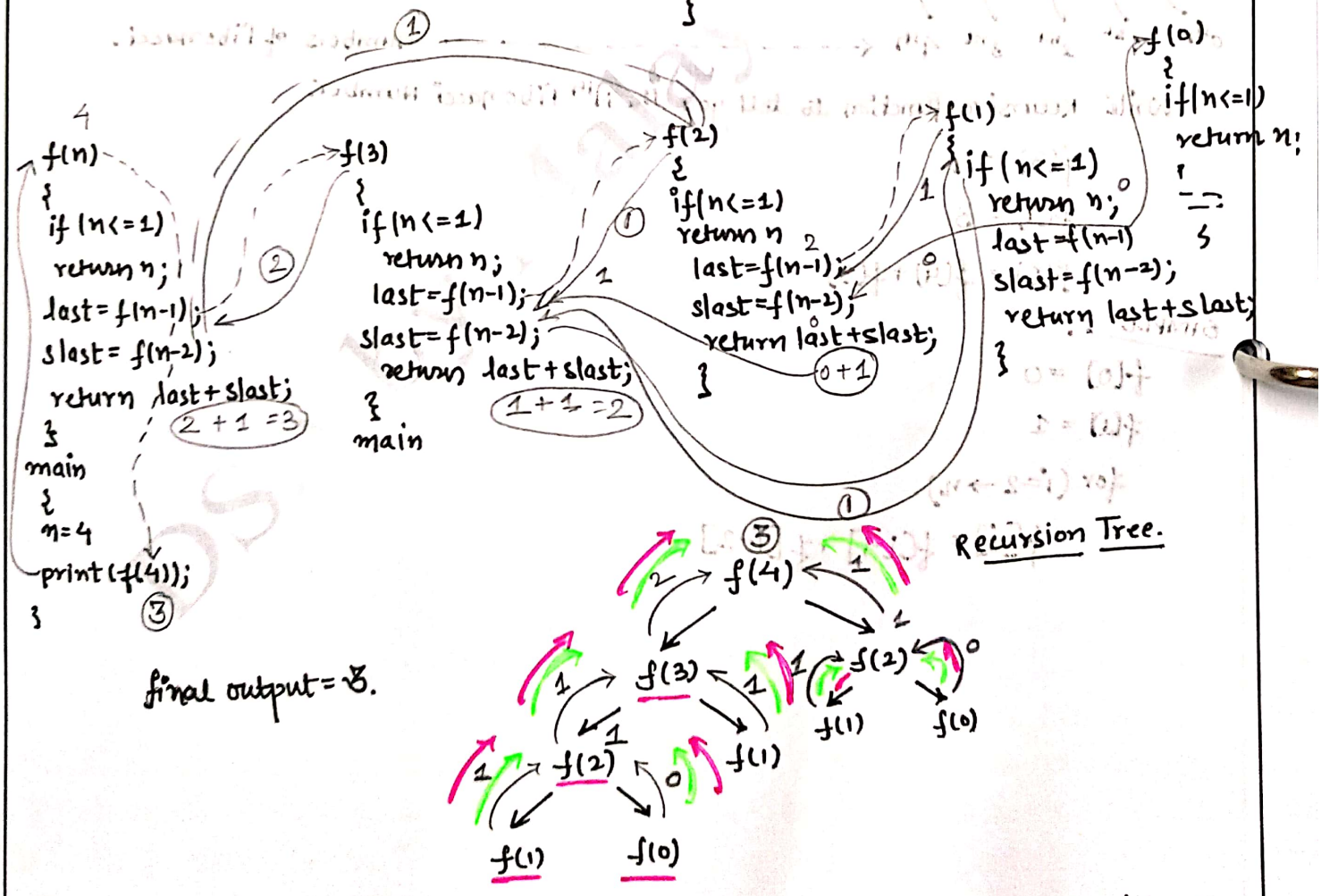
```
f(n)
{
 if (n<=1)
    return n;
 last= f(n-1);
 secondlast = f(n-2);
 return last+ secondlast;
}
main
{
 n=4
 print (f(4));
}
```

waiting for this line -->

f(3)
{
If ( ) x }
last= f(2)
second last = f(1)
return last+second last
}

f(2)
{
if ( ) x
last =f(1)
secondlast=f(0)
return last + secondlast

f(1)
{
return n.



final output = 3.

Recursion Tree.

TC = (2^n) approx (exponential) → as you are calling 2 func for each function.

Subarray is diff. from subseq.
subsequence is always subarray
But vice-versa is not true.

RECURSION ON SUBSEQUENCES

Subsequence → a contiguous / non-contiguous sequences which (follows the order).
↳ it is necessary that it follows the order of the array.

arr → [3, 1, 2] → subsequences.

{} → null is also subsequence.
3 → contiguous sequence → it also follow the order of the array
1
2
3 1
1 2
3 2 → non-contiguous → it also follows the order of the array. (3 is before 2)
3 1 2
3 2 1 ✗ it is not subsequence as it not follows the order of the array.

[3 1 2] → [3 2] Based on take/not index.

→ 3 x 2 → [1, 2]
→ x 1 2 → [3, 1]
→ 3 1 x → [3, 1, 2]
→ 3 1 2 → { }
→ { } → x x x

```
arr [3,1,2]
f(ind, [])
{
 if (ind>=n)
   print([])
   return;
[].add(arr[i]);
 f(ind+1,[]);
[].remove(arr[i]);
 f(ind+1,[]);
}
main
{
 arr→[312]
 f(0,[]);
}
```

```
f(1,[3])
{
if (ind>=n)
  print([])
  return;
[3].add(arr[1])
 f(2,[3 1])
[31].remove(arr[1])
f(2,[3])
```

```
f(2,[31])
{
 if(ind>=n)
  print([])
  return;
[31].add(arr[2])
 f(3,[312])
[312].remove[arr(2)]
[31]
f(ind+1,[31])
 f(3,[31])
```

```
f(3,[312])
{
 if(ind>=n)
 print([])
  return;
}
{ ---
  ---
  --
}
```

```
f(2,[3])
{
 if(ind>=n)
  print([])
  return;
[3].add(arr[2])
  f(3,[3,2])
[32].remove(arr[2])
   f(3,[3,2]).
```

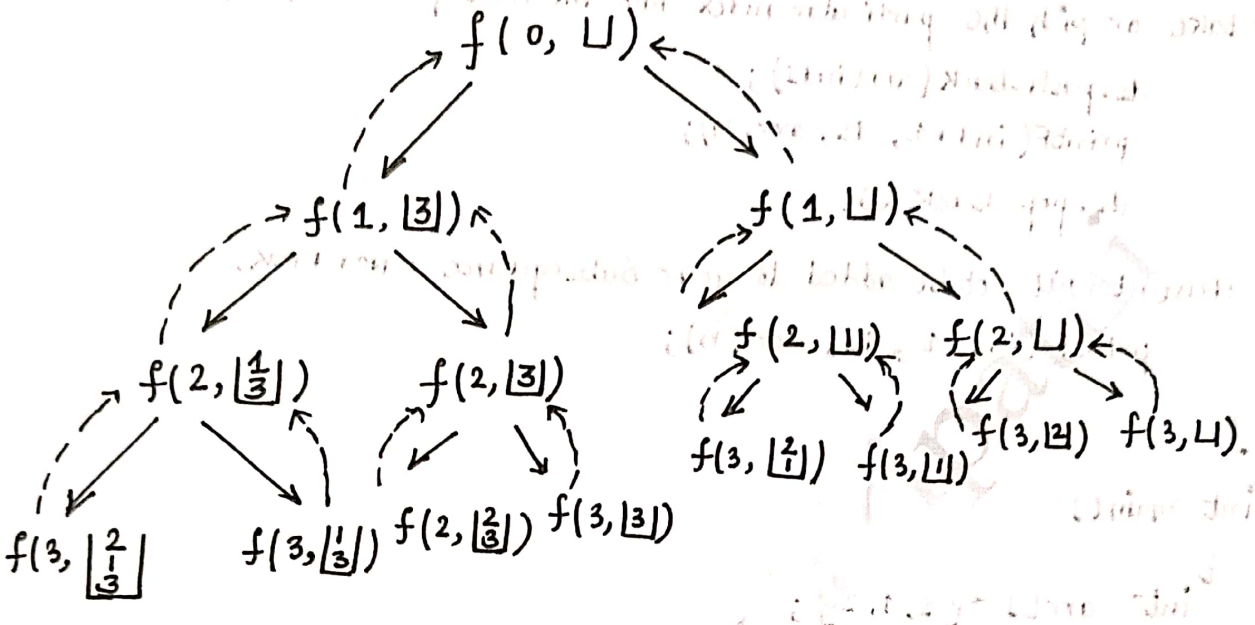
```
f(3,[32])
{ if(ind>=n)
   print([])
   return;
}
```

o/p
312
31
32
3
1 2
1
2
{}

RECURSION TREE

RECURSION ON SUBSEQUENCE.

arr = [3 1 2]



[3 1 2] [3 1] [3 2] [3] [1 2] [1] [2] { }

```
#include <bits/stdc++.h>
using namespace std;
void printf (int ind , vector<int>& ds, int arr[], int n)
    {
      if (ind == n)
        {
          for (auto it : ds)
            {
              cout << it << " ";
            }
          if (ds.size() == 0)
              {
                cout << "{}";
              }
              cout << endl;
              return;
        }
```

```
// take or pick the particular index into the subsequence = PICK
        ds.push_back(arr[ind]);
        printf(ind+1, ds, arr, n);
        ds.pop_back();
// element will not be added to your subsequence = NOT PICK.
        printF(ind+1, ds, arr, n);
}
int main()
{
    int arr[] = {3, 1, 2};
    int n=3;
    vector <int> ds;
    printF(0, ds, arr, n);
```

* TIME COMPLEXITY - for every index there is couple of answer/options

$$O(2^n) * n$$

* AUXILLARY STACK SPACE = not more than 3 Recursion call waiting at the stack space. because the Depth of the Recursion is THREE.

  SPACE COMPLEXITY

  O(N).

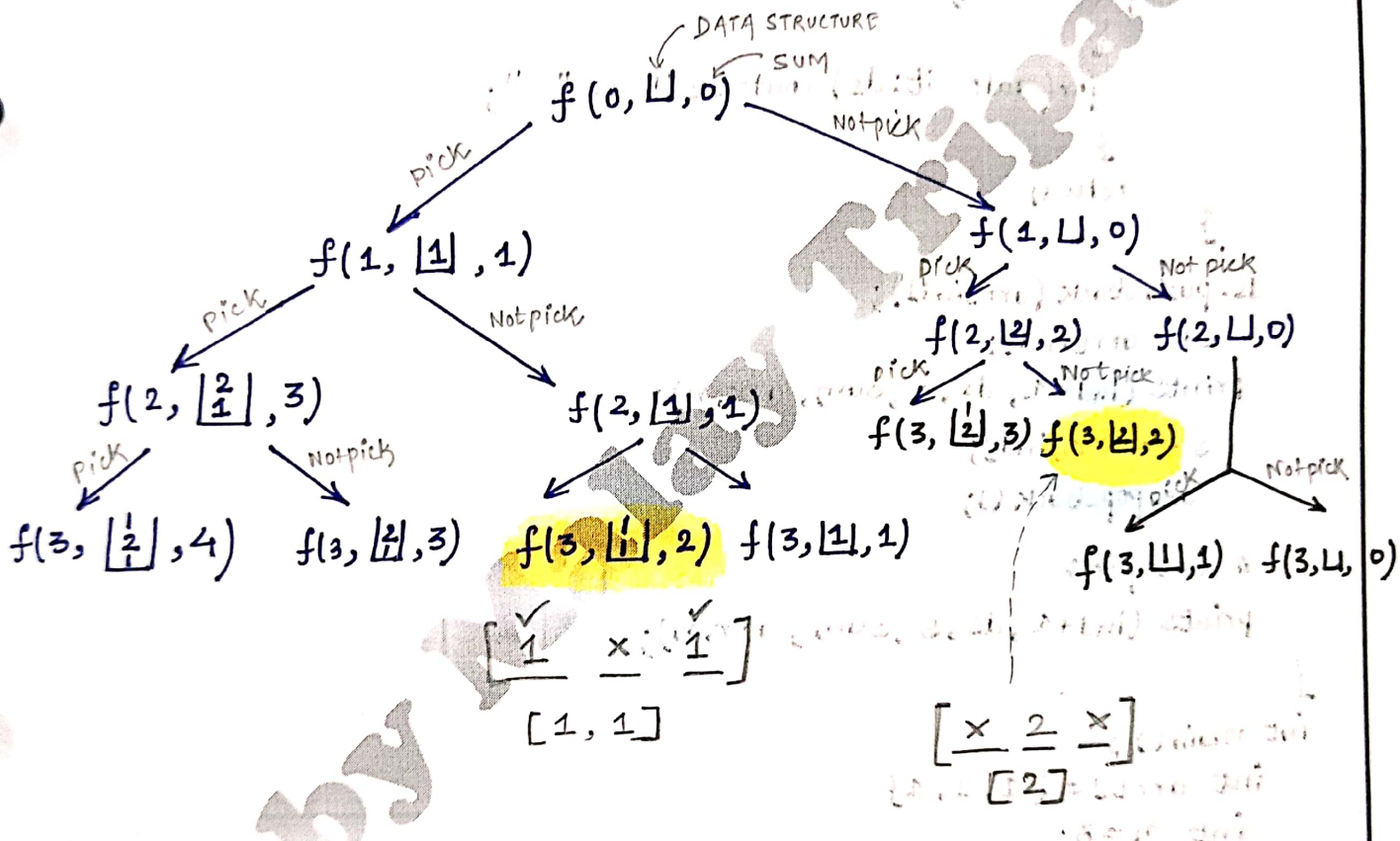
PRINTING SUBSEQUENCE WHOSE SUM IS K

↳ Technique of Take/Not Take.

arr → [1, 2, 1] Sum = 2

[1, 1]

[2]



Pseudo code:-

```
f(i, ⊔, s)
{
  if(i==n)
  {
    if(s==sum)
      print(ds)
    return;
  }
  ds.add(arr[i]);
  s += arr[i]
  f(i+1, ds, s)
```

```
ds.remove(arr[i])
s == arr[i];
```
PICK

```
f(i+1, ds, s)
```
NOT PICK

Data Structure and Algorithms
By
Malay Tripathi

o/p = [1 2]
[2]

Code

```
#include <bits/stdc++.h>
using namespace std;

void printS(int ind, vector<int> &ds, int s, int sum, int arr[], int n)
{
    if (ind == n)
    {
        if (s==sum)
        {
            for(auto it:ds) cout << it <<" ";
        }
        return
    }
    ds.push_back(arr[ind]);
    s+= arr[ind];
    printS(ind+1, ds, s, sum, arr, n);
    s-= arr[ind];
    ds.pop_back();
    // not pick
    printS(ind+1, ds, s, sum, arr, n);
}
int main(){
    int arr[] = {1, 2, 1}
    int n=3;
    int sum=2;
    vector<int> ds;
    printS(0, ds, 0, sum, arr, n);
    return 0;
}
```