CLASS 47

01 April 2023 11:07

Class-47

* Print all subsequences of an array
* Print all subsequence with a sum K

```
vector<vector<int>> ans;
void subsequence(vector<int> a, int idx, vector<int> output,
    vector<vector<int>> &ans)
{
    int n = a.size();
    if (idx == n)                    // Base Case
    {
        // Print subsequence
        ans.push_back(output);
        return;
    }
    // include the element
    output.push_back(a[idx]);
    subsequence(a, idx+1, output);
    // Don't include
    output.pop_back();
    subsequence(a, idx+1, output);
}
```

```
int sum = 0
for(auto i: output)
{
    sum += i;
}
if(sum == K)
{ ans.push_back(output) }
```

Sum = K

```
vector<vector<int>> ans
output;
void f(vector<int>a, int idx, vector<int> output, int sum)
{
    int n = a.size();
    if(idx == n)
    { int sum = 0;
      for(auto i: output)
      { sum += i;
```

→ $2^n * K$

Base Case

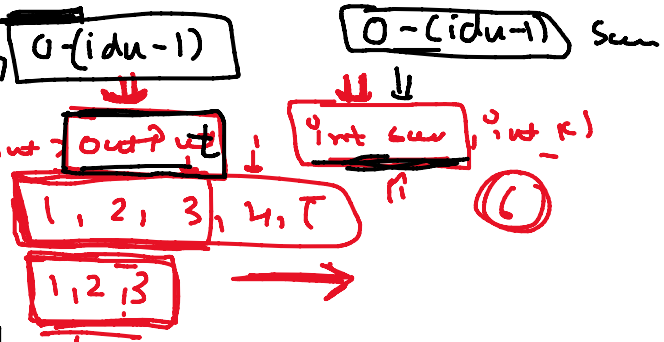
```
For (auto i: output)
{ sum += i;
}
if (sum == K)
{ ans.push_back(output);
}
}
output.push_back(a[i]);
s(a, idx+1, output);
output.pop_back();
s(a, idx+1, output);
}
```

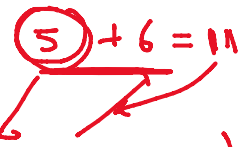
Base case

n = 10, 20

2nd version


```
void s(vector<int> a, int idx, vector<int> output)
{ int n = a.size();
  if (idx == n)
  { if (sum == K)
      ans.push_back(output);
  }
  output.push_back(a[idx]);
  [sum += a[idx];]
  ⇒ s(a, idx+1, output, sum, K)
  output.pop_back(a[idx]);
  sum -= a[idx];
  s(a, idx+1, output, sum, K)
}
```





sum = sum + a[idx];

O(n)

(6), 0, 0

T.C = 2^n

2n + n*4

```
void s(vector<int> a, int idx, vector<int> output, int sum, int K)
{ int n = a.size();
  if (sum == K) { ans.push_back(output); return; }
```

```
{  int n = a.size();
   if(idu==n)
   {
      if(sum == K)
      {ans.push(output);}
      return;
   }
// element include
   if(sum+a[idu] <= K)
   {  output.push_back(a[idu]);
      sum += a[idu];
      [s(a,idu+1, output, sum, K);]
      output.pop_back();   sum -= a[idu];
   }
// don't include
   [s(a, idu+1, output, sum, K);]
}
```

if(sum==K) { ans.push_back(output); return; }

1,2 ... idu==n

(3)

(6) ← Positive No

1,2 (4)

(7)

sum (3)

1,2,4 ..... K=6

(1,2,3,4)

1 2 - -

(3)

K = 100000

s(a,0,ot,0,6)

ot → [1]

s(a,1,ot,1,6)

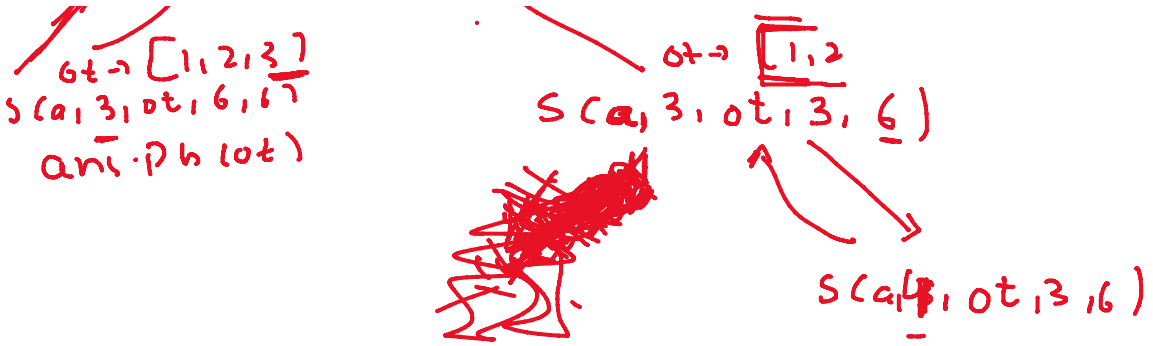
Subset == K

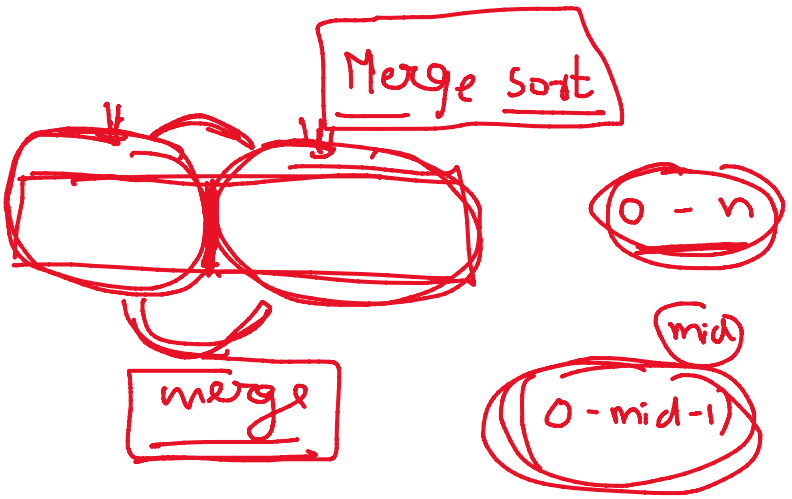
ot → [1,2]

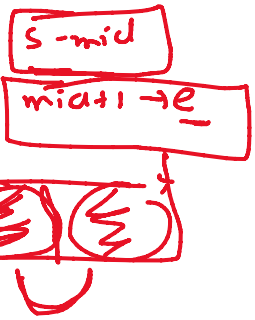
s(a,2,ot,3,6)

ot → [1,2,3]

ot → [1,2]

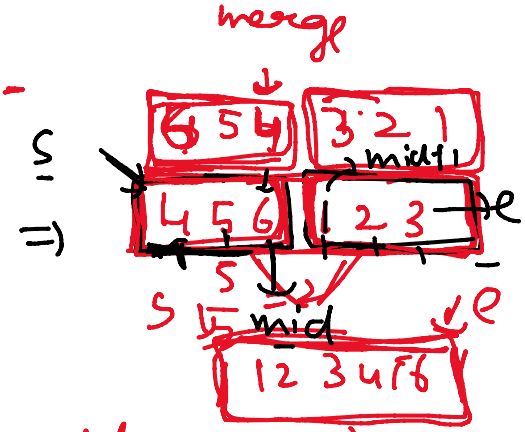







```
void mergeSort (int ar[], int s, int e)
{   if (s >= e) return;
    int mid = (s+e)/2;
    mergeSort (a, s, mid);
    mergeSort (a, mid+1, e);
    merge (a, s, mid, e);
}
```

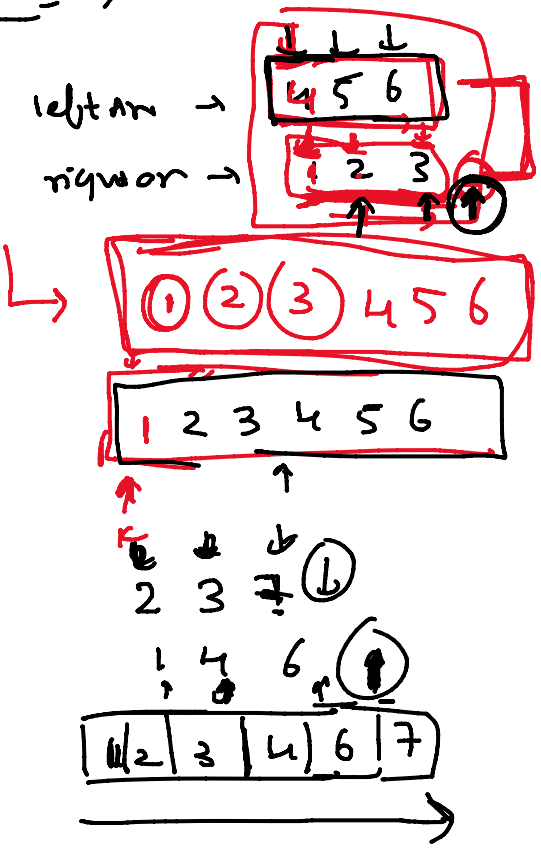
n/2



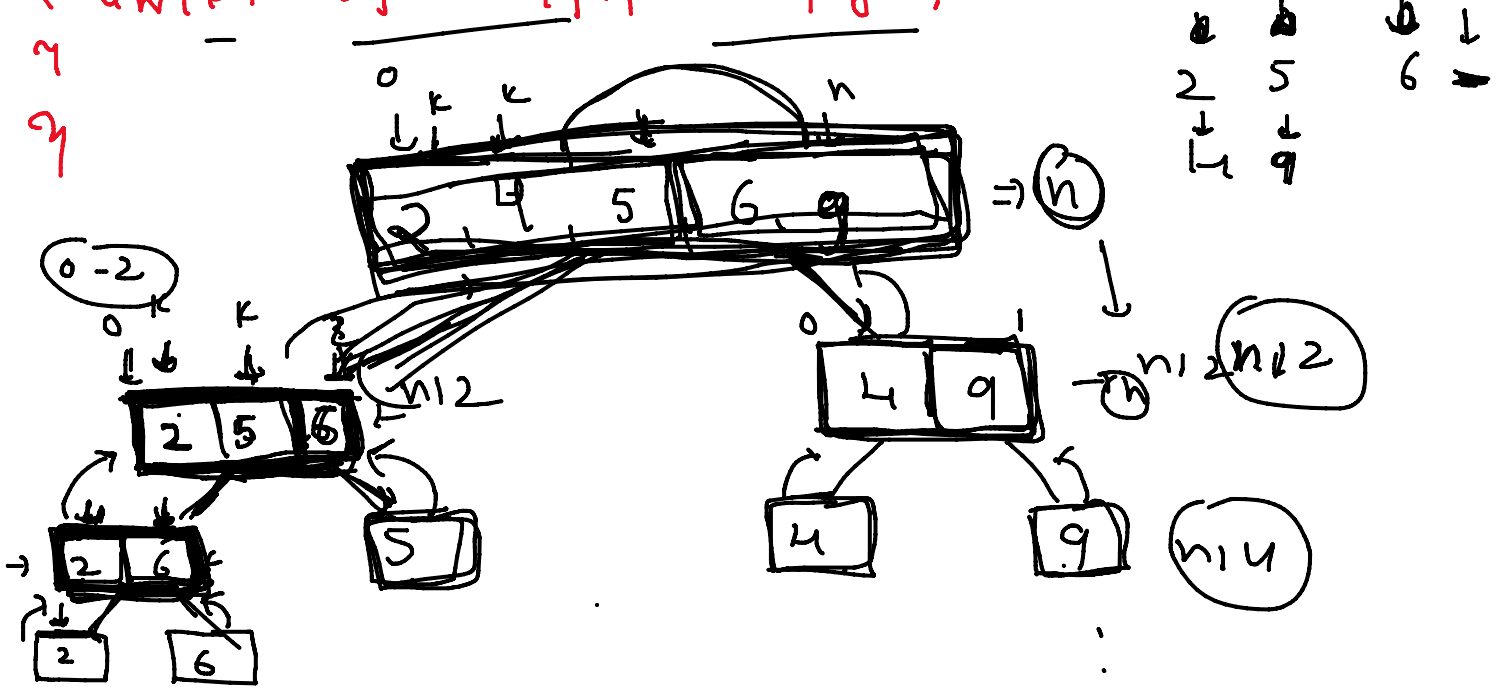

```
void merge ( int ar[], int start, int mid, int end)
{   int l1 = mid - start + 1;
    int l2 = end - mid;
```

```
int len2 = end - mid;
int leftarr[len1], rightArr[len1];
for (i=0; i<len1; i++)
{ leftarr[i] = ar[start + i];
}
for (j=0; j<len2; j++)
{ rightarr[j] = ar[mid + 1 + j];
}
int i,j,k;
i=0; j=0; k=0
while (i<len1 and j<len2) =
{ if (leftarr[i] <= rightArr[j])
  { ar[k] = leftarr[i]; i++;
  }
  else
  { arr[k] = rightArr[j]; j++;
  }
  k++;
}
while (i<len1)
{ arr[k] = left Arr[i];
  i++; k++;
```

O(n)



```
}
while (j < len2)
{ arr[k] = right arr[j]; k++; j++;
}
}
```



let Arr = [2 6]

right Arr = [5]

logn

Recursion Tree

2^n

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

log n

T.C

$T(n) = 2 * T(n/2) + O(n)$

$T(n/2) = 2 * T(n/4) + O(n/2)$

$T(n/4) = 2 * T(n/8) + O(n/4)$

$T(n) = 2 * [\, 2 * T(n/4) + O(n) \,] + O(n)$

$T(n) = [\, T(n/8) + O(n) + O(n) \,] + O(n)$

$T(n) =$ [illegible] $+ 30(n)$

$$T(n) = T(n/8) + 30(n)$$

$$T(n) = T(n/16) + 40(n)$$

$n \to$ 1

$\frac{n}{2}, \frac{n}{4}, \frac{n}{8}$ -- log n

$n, n/2, n/4, \frac{n}{8}, \frac{n}{16}, \ldots$ (1)

$T(n) = T(1) + K * O(n)$ ⇓

$\log n$

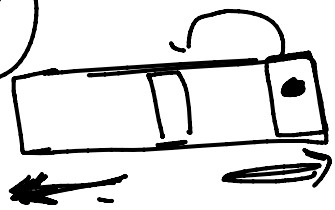
$\log_2 n$

$O(n \log n)$

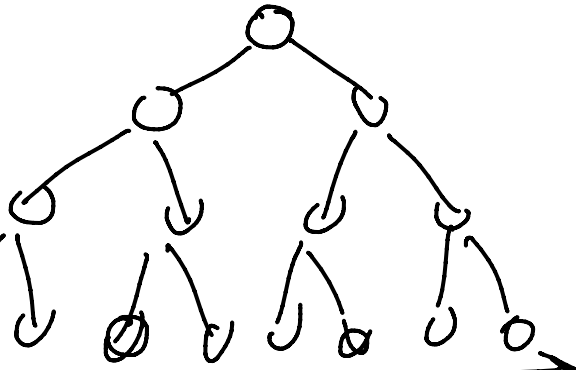
$\frac{n}{2^k} = 1$

$(n = 2^k)$ ⇒ $k = \log_2 n$

Quick sort



Recursion Tree

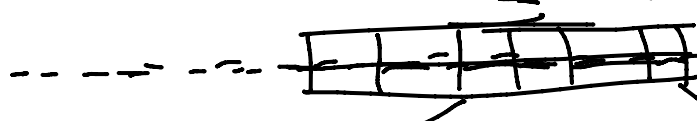


$N_1 + N_2 + N_3 \times (1)$ -- $N_n \times (k)$

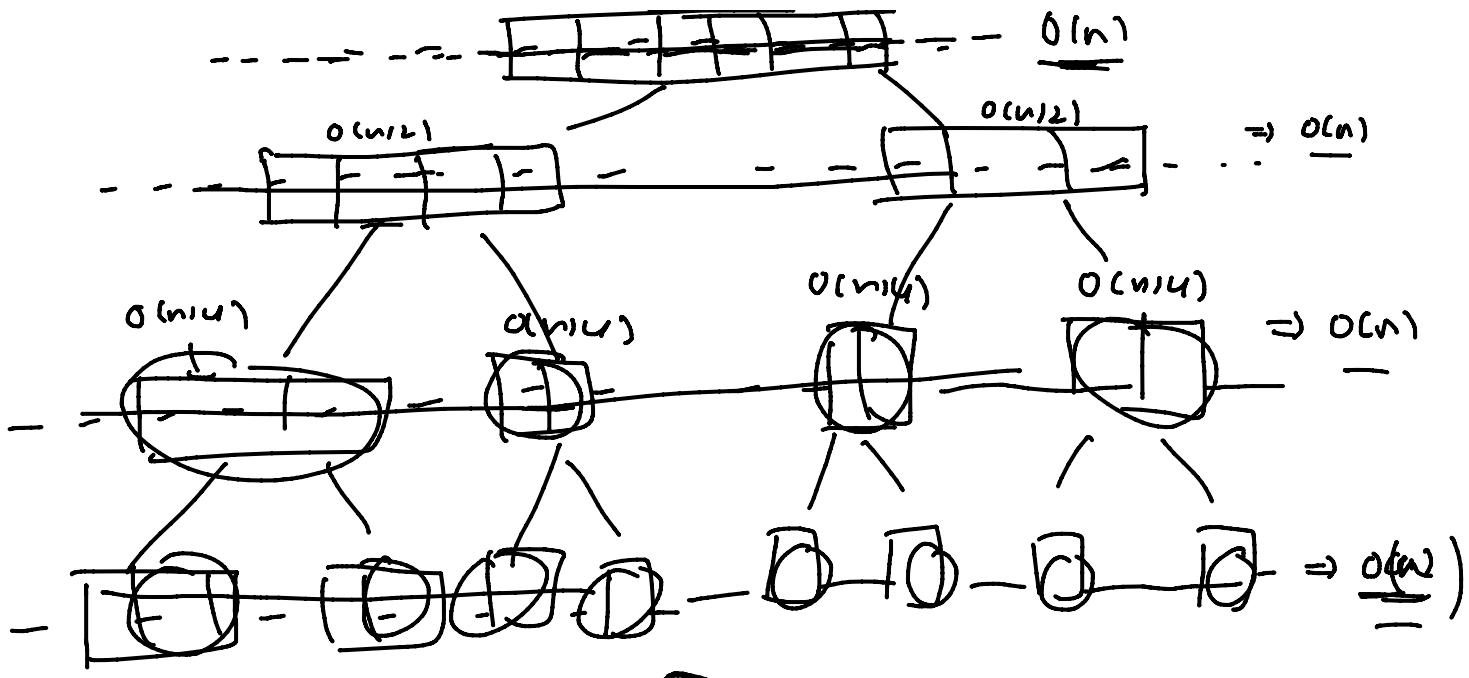
$= k[N_1 + N_2 + N_3 \ldots N]$

T.C = Total Time to execute the code = Total time taken by all the Nodes

Time taken by each Node $\times$ Total No of Node



$O(n)$



logn

Total complexity: Time = No of levels * Time taken ⇒ O(n) * O(logn)

O(n) , O(n/2) O(n/4) O(n/8)