Asymptotic Analysis – measures the order of growth of an algorithm in terms of the input size

$1 < \log \log n < \log n < n^{1/3} < n^{1/2} < n < n^2 < n^3 < n^4 < 2^n < n^n$

Big O Notation – upper/exact bound
Theta Notation – exact bound
Omega Notation – lower/exact bound

```
for (int i = 0; i < n; i = i + c) {
    // θ (1) work
}
```

θ (n)

```
for (int i = 0; i < n; i = i – c) {
    // θ (1) work
}
```

θ (n)

```
for (int i = 0; i < n; i = i * c) {
    // θ (1) work
}
```

θ (log n)

```
for (int i = 0; i < n; i = i / c) {
    // θ (1) work
}
```

θ (log n)

```
for (int i = 0; i < n; i = pow (i, c)) {
    // θ (1) work
}
```

θ (log log n)

add subsequent loop values
multiply nested loop values

```
void fun (int n) {
    if (n <= 0)
        return;
    // θ (1) work
    fun (n / 2);
    fun (n / 2);
}
```

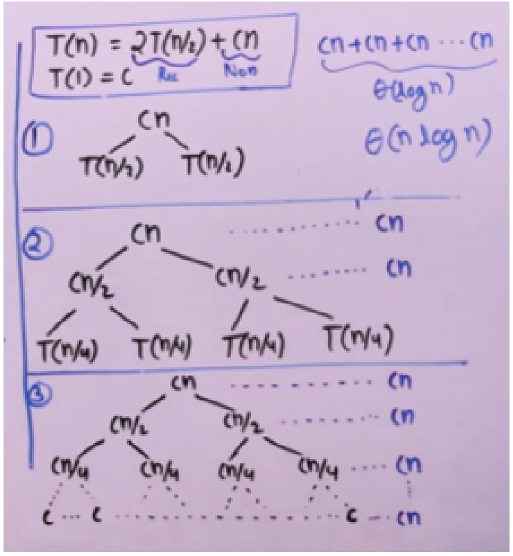
T (n) = 2T (n / 2) + θ (1)
T (0) = θ (1)

```
void fun (int n) {
    if (n <= 0)
        return;
    for (int i = 0; i < n; i = i++) {
        // θ (1) work
    }
    fun (n / 2);
    fun (n / 3);
}
```

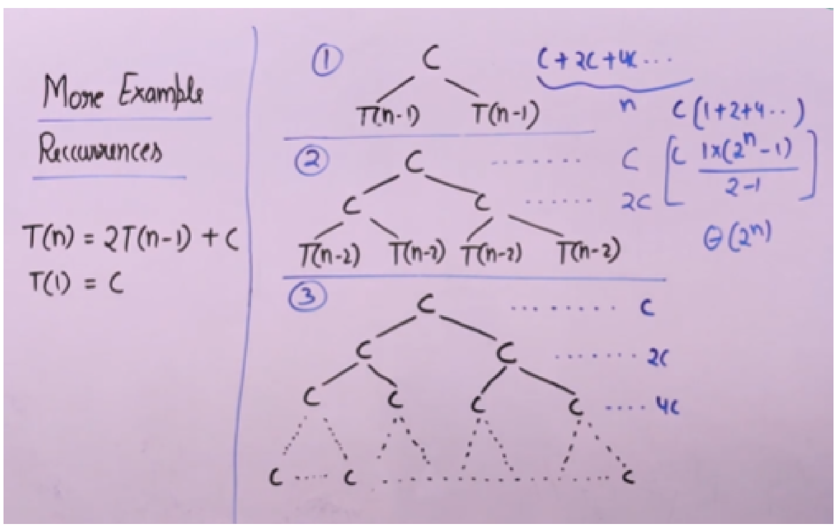
T (n) = T (n / 2) + T (n / 3) + θ (n)
T (0) = θ (1)

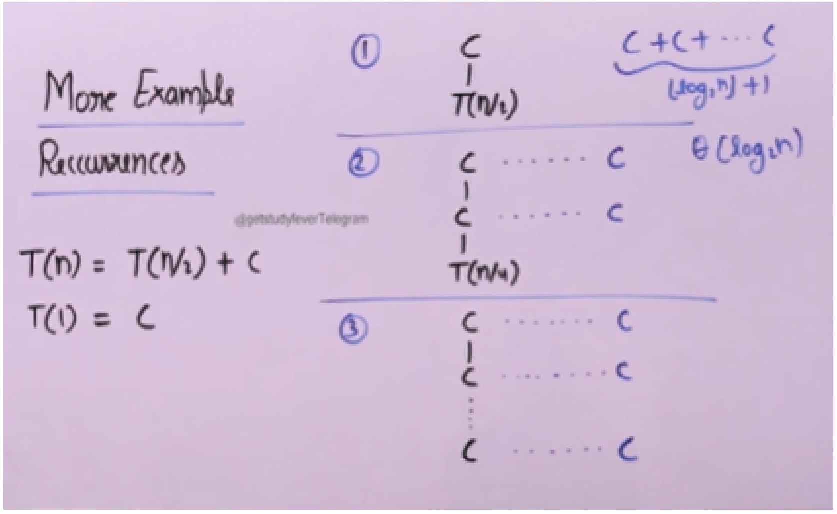
```
void fun (int n) {
    if (n <= 1)
        return;
    // θ (1) work
    fun (n  - 1);
}
```

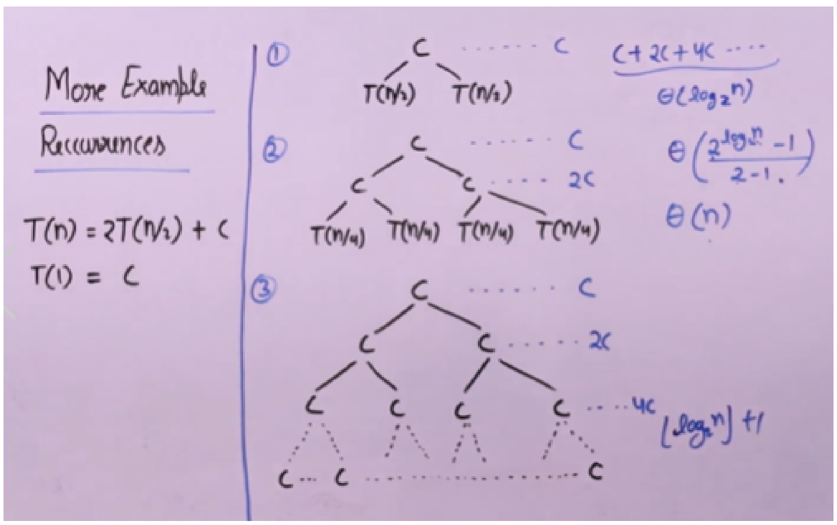
T (n) = T (n – 1) + θ (1)
T (1) = θ (1)

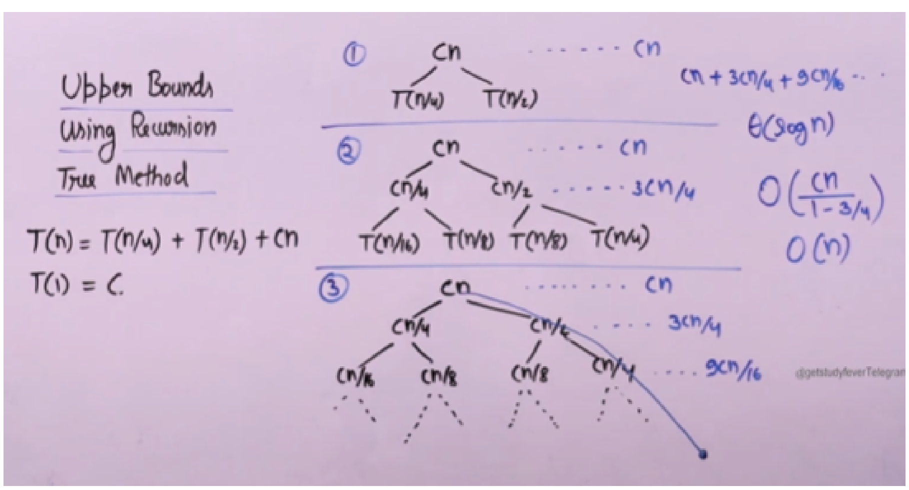


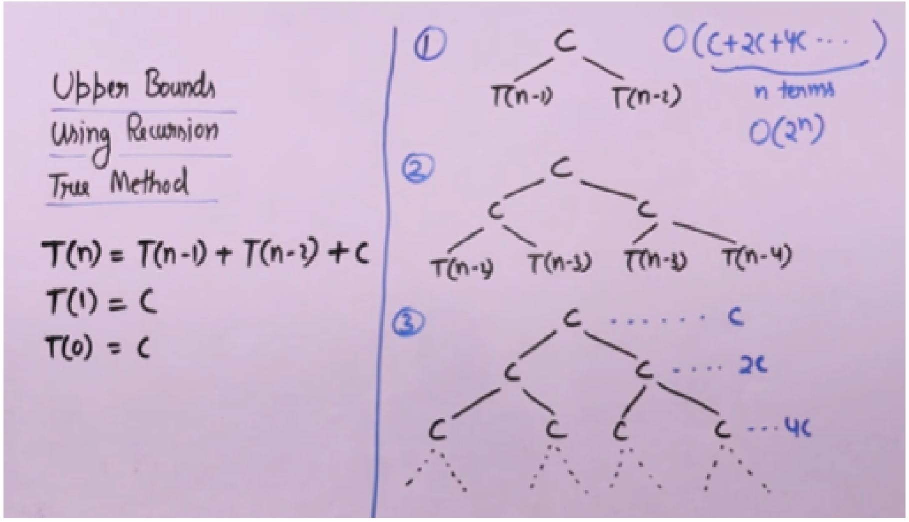
T (n) = 2T (n / 2) + c_1n
T (1) = c_2
θ (nlog n)




More Example
Reccurrences
T(n) = T(n/2) + c
T(1) = c
@getstudyfeverTelegram
①
T(n/2)
②
T(n/4)
③
c + c + ··· c
⌊log₂n⌋ + 1
Θ(log₂n)


More Example
Reccurrences
T(n) = 2T(n/2) + c
T(1) = c
①
T(n/2) T(n/2)
②
2c
T(n/4) T(n/4) T(n/4) T(n/4)
③
2c
4c
⌊log₂n⌋ + 1
c + 2c + 4c ····
Θ(log₂n)
Θ(n)


Upper Bounds
Using Recursion
Tree Method
T(n) = T(n/4) + T(n/2) + cn
T(1) = c
①
cn
T(n/4) T(n/2)
②
cn/4 cn/2
3cn/4
T(n/16) T(n/8) T(n/8) T(n/4)
③
cn/16 cn/8 cn/8 cn/4
9cn/16
cn + 3cn/4 + 9cn/16 ···
Θ(log n)
O(n)
@getstudyfeverTelegram



Space Complexity –

```
int get_sum1 (int n) {
   return n * (n + 1) / 2;
}
```

θ (1)

```
int get_sum2 (int n) {
   int sum = 0;
   for (int i = 1; i <= n; i++)
      sum = sum + i;
   return sum;
}
```

θ (1)

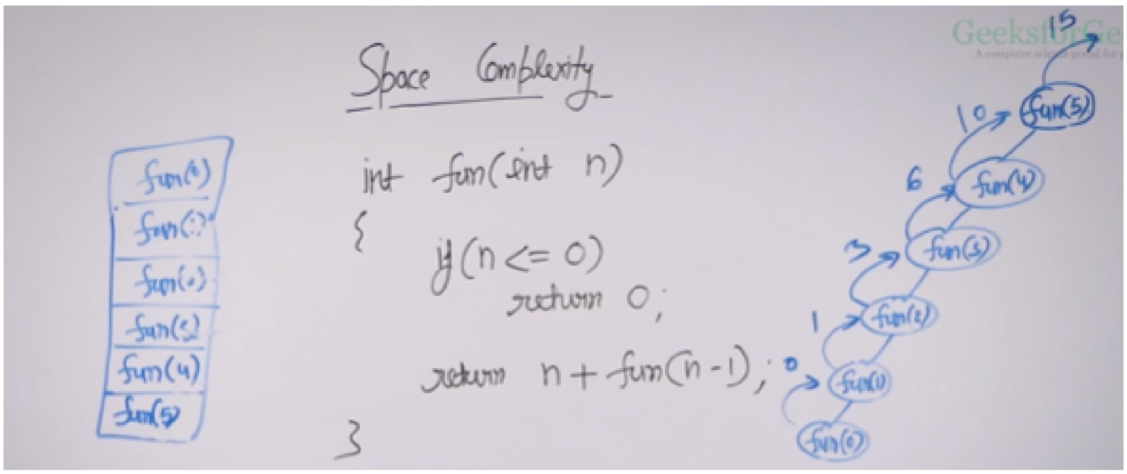
```
int arr_sum (int a [], int n) {
   int sum = 0;
   for (int i = 0; i < n; i++)
      sum += a [i];
   return sum;
}
```

θ (n)

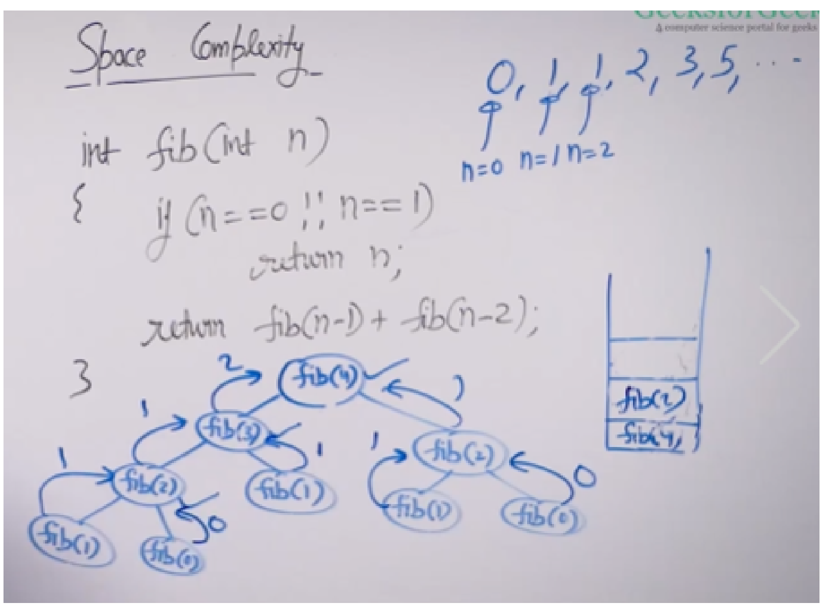
Auxiliary Space – measures the order of growth of extra space

```
int arr_sum (int a [], int n) {
   int sum = 0;
   for (int i = 0; i < n; i++)
      sum += a [i];
   return sum;
}
```

θ (1)



θ (n)



θ (n)