

$$T(n) = \begin{cases} d_0 & \text{if } n = 0 \\ c_0 + c_1 n + T(n-1) & \text{otherwise} \end{cases}$$

$$T(n) = (c_0 + c_1 n) + T(n-1)$$

$$= (c_0 + c_1 n) + (c_0 + c_1(n-1) + T(n-2))$$

$$= \sum + (c_0 + c_1(n-2) + T(n-3))$$

$$\left( \sum_{i=0}^{n-1} (c_0 + c_1(n-i)) \right) + d_0$$

## Merge Sort

```

1  sort(L) {
2      if (L.size() < 2) {
3          return L;
4      }
5      else {
6          int mid = L.size() / 2;
7          return merge(
8              sort(L.subList(0, mid)),
9              sort(L.subList(mid, L.size()))
10         );
11     }
12 }
    
```

Handwritten annotations on the code:

- Red bracket on lines 2-4 labeled "BC".
- Blue circles around lines 6 and 7 labeled "NR".
- Green circles around lines 8 and 9 labeled "R".
- Yellow circles around the recursive calls on lines 8 and 9.
- Yellow "n/2" labels under the recursive calls.

First, we need to find the recurrence:

$$T(n) = 2T(n/2) + \cancel{K}$$

$$NR = O(1) + O(n) = O(n)$$

$$R = 2T(n/2)$$

$$BC = O(1)$$

# Merge Sort: Solving the Recurrence

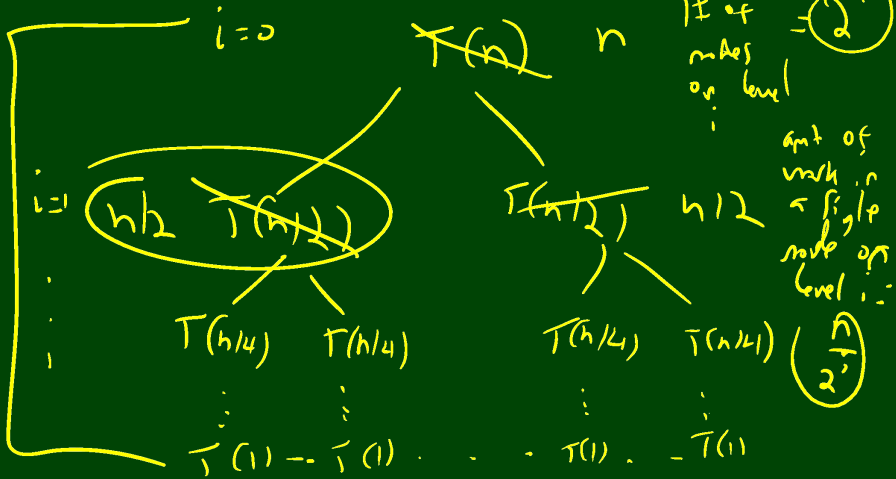
height =  $\lg(n)$

$$T(n) = \begin{cases} d_0 & \text{if } n = 0 \\ d_1 & \text{if } n = 1 \\ c_0 + c_1 n + 2T(n/2) & \text{otherwise} \end{cases}$$

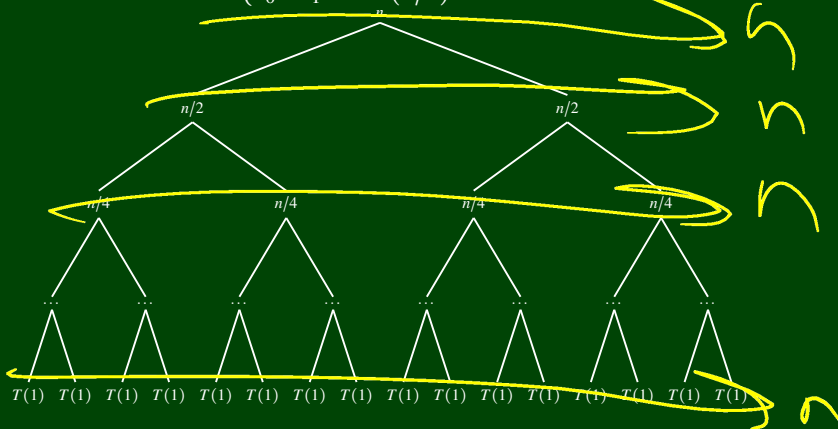
$$\sum_{i=0}^{\lg(n)-1} 2^i \cdot \frac{n}{2^i}$$

# of nodes on level  $i = 2^i$

amt of work in single node on level  $i = \frac{n}{2^i}$



$$T(n) = \begin{cases} d_0 & \text{if } n = 0 \\ d_1 & \text{if } n = 1 \\ c_0 + c_1 n + 2T(n/2) & \text{otherwise} \end{cases}$$



## Find A Big-Oh Bound For The Worst Case Runtime

```

1 sum(n) {
2   if (n < 2) {
3     return n;
4   }
5   return 2 + sum(n - 2);
6 }
    
```

Handwritten annotations:   
 - A green circle around the `if (n < 2)` block with "BC" written next to it.   
 - A red circle around `sum(n - 2)` with an arrow pointing to it.   
 - Blue brackets under `2 +` with "NR" written below each.   
 - A yellow circle around the `NR` annotations.

$$T(n) = \begin{cases} 1 & \text{if } n < 2 \\ 1 + T(n-2) & \text{otherwise} \end{cases}$$

$$\underbrace{1 + 1 + 1 + \dots + 1}_{n/2}$$

## Find A Big-Oh Bound For The Worst Case Runtime

```
1 binarysearch(L, value) {
2   if (L.size() == 0) {
3     return false;
4   }
5   else if (L.size() == 1) {
6     return L[0] == value;
7   }
8   else {
9     int mid = L.size() / 2;
10    if (L[mid] < value) {
11      return binarysearch(L.subList(mid + 1, L.size()), value);
12    }
13    else {
14      return binarysearch(L.subList(0, mid), value);
15    }
16  }
17 }
```

$$T(n) = 1 + \max(T(n/2), T(n/2))$$

Consider a recurrence of the form:

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + n^c & \text{otherwise} \end{cases}$$

Then,

- If  $\log_b(a) < c$ , then  $T(n) = \Theta(n^c)$ .
- If  $\log_b(a) = c$ , then  $T(n) = \Theta(n^c \lg(n))$ .
- If  $\log_b(a) > c$ , then  $T(n) = \Theta(n^{\log_b(a)})$ .