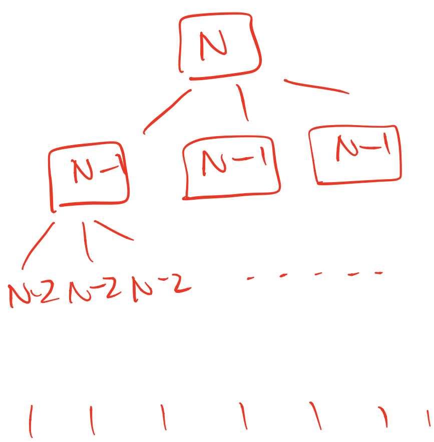
1 Runtime Analysis

Find out the Big-Θ bound of the following functions:

1.
```
public void f1(int N) {
    if (N <= 0) {
        return;
    }
    System.out.println("working");
    return f1(N-1) + f1(N-1) + f1(N-1);
}
```

Each node has a constant runtime as $\Theta(1)$, so we could count the total amount of nodes and it will be our answer.



Height	No. of node(s)
0	1 (3^0)
1	3 (3^1)
⋮	9 (3^2)
	⋮
$N-1$	3^{N-1}

(start with 0)

Count Nodes.

$1+3+9+\dots+3^{N-1}$

$< 2 * 3^{N-1}$

- Remove the constants
  We have approx 3^N nodes.
  For each node the runtime is $\Theta(1)$.

- Result: $\Theta(1 \cdot 3^N)$
  $= \Theta(3^N)$

- when $N=3$, the last layer has height $= 3-1=2$, The last layer has total of $3^2=9$ nodes.
  $(1+3)<9$ $\boxed{\text{diff}: 9-4=5}$
- when $N=4$, $h=3$, last-layer $=27$
  $(1+3+9)<27$ $\boxed{\text{diff}: 27-13=14}$
- $N=5$, $h=4$, last-layer $=81$
  $(1+3+9+27)<81$ $\boxed{\text{diff}: 41}$
- Conclusion:
  $(1+3+9+\dots+3^{N-2}) < 3^{N-1}$

Therefore:

$$\boxed{(1+3+9+\dots+3^{N-2})+3^{N-1} < 2 * 3^{N-1}}$$

2.
```
public void f2(int N, ~~int M~~) {
    int count = 0;
    for(int i = N/2; i < N; i++) {
        for(int j = 1; j <= N; j = 2 * j) {
            for(int k = 1; k <= N; k = k * 2) {
                count++;
            }
        }
    }
}
```

execute $n/2$ times $= \Theta(n)$

execute $\log n$ times

execute $\log n$ times

$\log(N)$:

- How many $*2$ operation will get to N so the for loop stops?

if we execute h time and we stop,

$$N = 1 * \underbrace{2 * 2 * \ldots * 2}_{*2 \ h \text{ times}}$$

$$= 2^h$$

$h = \log N$ ← so we have execute $\log N$ times.

- Result: they're nested.

$$\Theta(N \log N \cdot \log N)$$

$$= \Theta(N \log^2 N)$$

3.
```
public void f3(int N, int M) {
    if (N <= 0) {
        return;
    }
    for(int i = 0; i < M; i++) {
        System.out.println("working");
    }
    return f3(N-1, M) + f3(N-1, M) + f3(N-1, M);
}
```

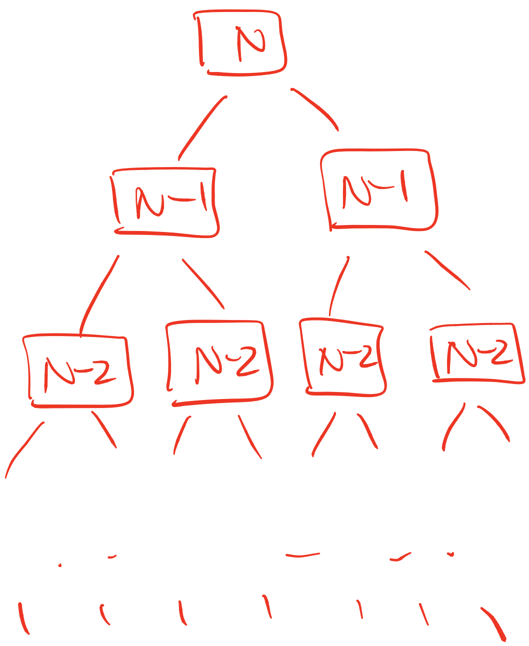
The only difference between 1 and 3 is that each node execute a for loop with $\Theta(M)$. The runtime is the same for all nodes, so we keep the same total amount of nodes from 1 : 3^N

Res :

$\Theta(3^N \cdot M) = \Theta(M3^N)$

4.
```
public void f4(int N) {
    if (N <= 0) {
        return;
    }
    for(int i = 0; i < N; i++) {
        System.out.println("working");
    }
    return f4(N-1) + f4(N-1);
}
```

- This time, different from section Question 3, the runtime of each level is not the same.
- Can't use counting method as ① since nodes have different runtime.



height	No. of nodes	Each node runtime	runtime by level
0	1	N	$1 \cdot N = N$
1	2	$N-1$	$2(N-1)$
2	4	$N-2$	$4(N-2)$
⋮	⋮		⋮
$N-1$	2^{N-1}	1	$2^{N-1} \cdot 1$

Sum of runtime by each level

$$N + 2(N-1) + 4(N-2) + 8(N-3) \dots + 2^{N-1}(N-(N-1))$$

$$= N + 2N - 2 + 4N - 8 + 8N - 24 + \dots + 2^{N-1}$$

ignore the constants.

$$\boxed{\leq} (1 + 2 + 4 + 8 + \dots + 2^{N-1}) N - (2 + 8 + 24 + \dots)$$

formula

$2^{N-1} N > 2^{N-1} \cdot 1$

so the constants could be ignored.

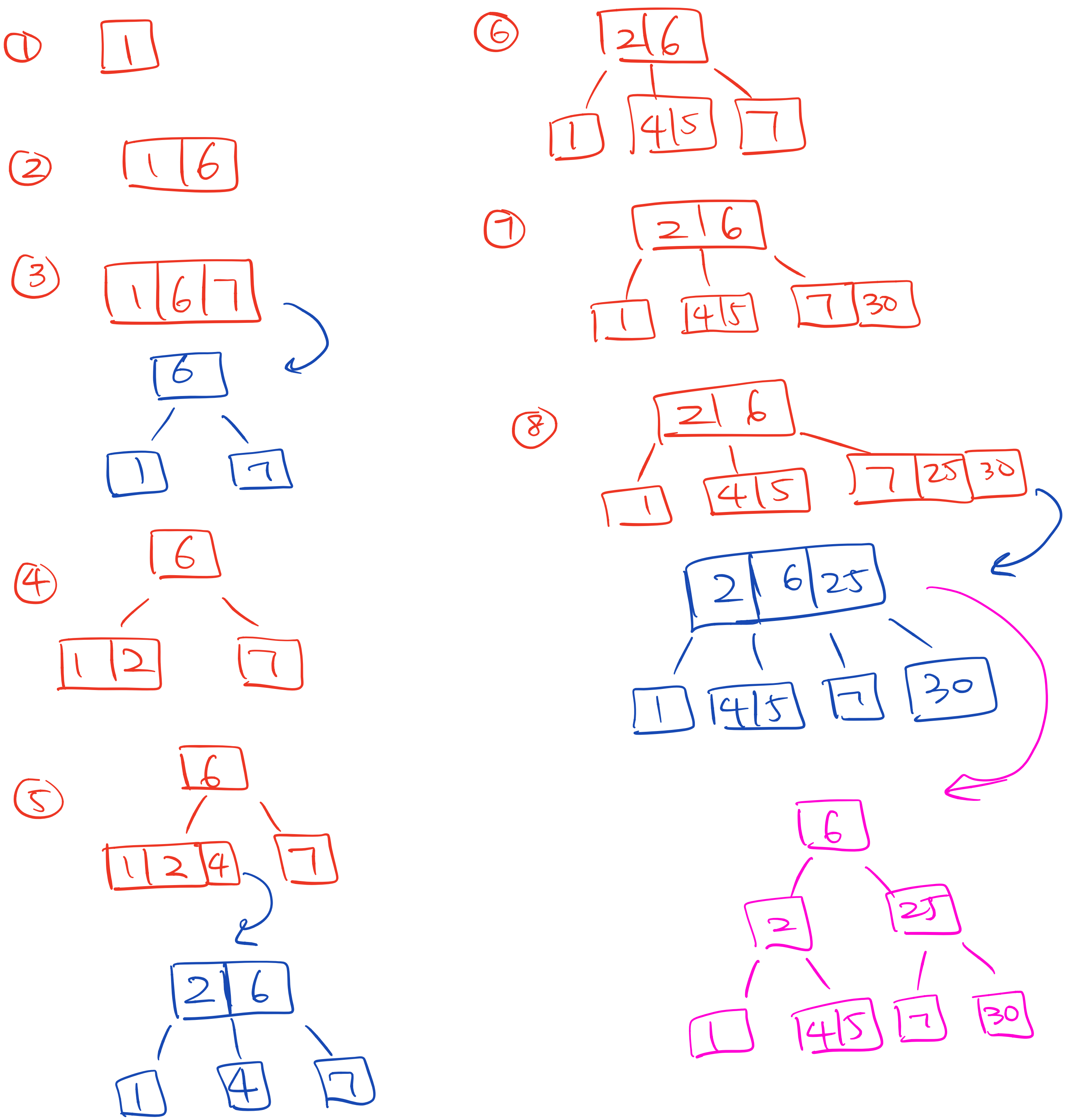
$$= (2 \cdot 2^{N-1} - 1) N$$

$$= 2^N N$$ remove the constants

$$\Theta(2^N N)$$

2 B-Trees

1. Construct a 2-3 B-Tree with keys in order: 1, 6, 7, 2, 4, 5, 30, 25.



2. Construct a 2-3-4 B-Tree with the same keys above. 1, 6, 7, 2, 4, 5, 30, 25