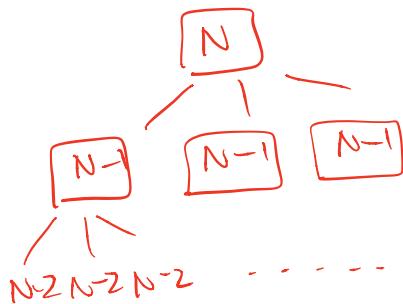


1 Runtime Analysis

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);
}
```



Height	No. of node(s)
0	1 (3^0)
1	3 (3^1)
2	9 (3^2)
3	3^3
4	3^4
5	3^5
6	3^6
7	3^7
8	3^8
9	3^9
10	3^{10}
11	3^{11}
12	3^{12}
13	3^{13}
14	3^{14}
15	3^{15}
16	3^{16}
17	3^{17}
18	3^{18}
19	3^{19}
20	3^{20}
21	3^{21}
22	3^{22}
23	3^{23}
24	3^{24}
25	3^{25}
26	3^{26}
27	3^{27}
28	3^{28}
29	3^{29}
30	3^{30}
31	3^{31}
32	3^{32}
33	3^{33}
34	3^{34}
35	3^{35}
36	3^{36}
37	3^{37}
38	3^{38}
39	3^{39}
40	3^{40}
41	3^{41}
42	3^{42}
43	3^{43}
44	3^{44}
45	3^{45}
46	3^{46}
47	3^{47}
48	3^{48}
49	3^{49}
50	3^{50}
51	3^{51}
52	3^{52}
53	3^{53}
54	3^{54}
55	3^{55}
56	3^{56}
57	3^{57}
58	3^{58}
59	3^{59}
60	3^{60}
61	3^{61}
62	3^{62}
63	3^{63}
64	3^{64}
65	3^{65}
66	3^{66}
67	3^{67}
68	3^{68}
69	3^{69}
70	3^{70}
71	3^{71}
72	3^{72}
73	3^{73}
74	3^{74}
75	3^{75}
76	3^{76}
77	3^{77}
78	3^{78}
79	3^{79}
80	3^{80}
81	3^{81}
82	3^{82}
83	3^{83}
84	3^{84}
85	3^{85}
86	3^{86}
87	3^{87}
88	3^{88}
89	3^{89}
90	3^{90}
91	3^{91}
92	3^{92}
93	3^{93}
94	3^{94}
95	3^{95}
96	3^{96}
97	3^{97}
98	3^{98}
99	3^{99}
100	3^{100}

(start with 0)

- When $N=3$, the last layer has height = $3-1=2$, The last layer has total of $3^2=9$ nodes.

$$(1+3) < 9$$

$$\text{diff} = 9 - 4 = 5$$

- When $N=4$, $n=3$, last-layer = 27

$$(1+3+9) < 27$$

$$\text{diff} = 27 - 13 = 14$$

- $N=5$, $n=4$, last-layer = 81

$$(1+3+9+27) < 81$$

$$\text{diff} = 41$$

- Conclusion:

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

Therefore:

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

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 : $\Theta(1 \cdot 3^N)$

$$= \Theta(3^N)$$

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

```
2. public void f2(int N, i < N) {
    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 $\times 2$ operation will get to N so the for loop stops?

if we execute h time and we stop,

$$N = \underbrace{1 \times 2 \times 2 \times \dots \times 2}_{\times 2 h \text{ times.}}$$

$$= 2^h$$

$$h = \log N \leftarrow \text{so we have execute } \log N \text{ times.}$$

- Result: they're nested.

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

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

373

```
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

Ans :

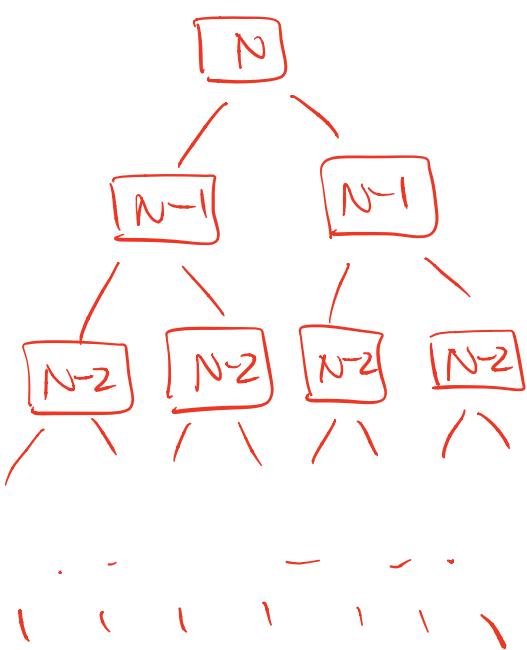
$$\Theta(3^N \cdot M) = \Theta(M3^N)$$

```
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

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	$N-2$	$4(N-2)$
\vdots	\vdots	\vdots	\vdots
$N-1$	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.

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

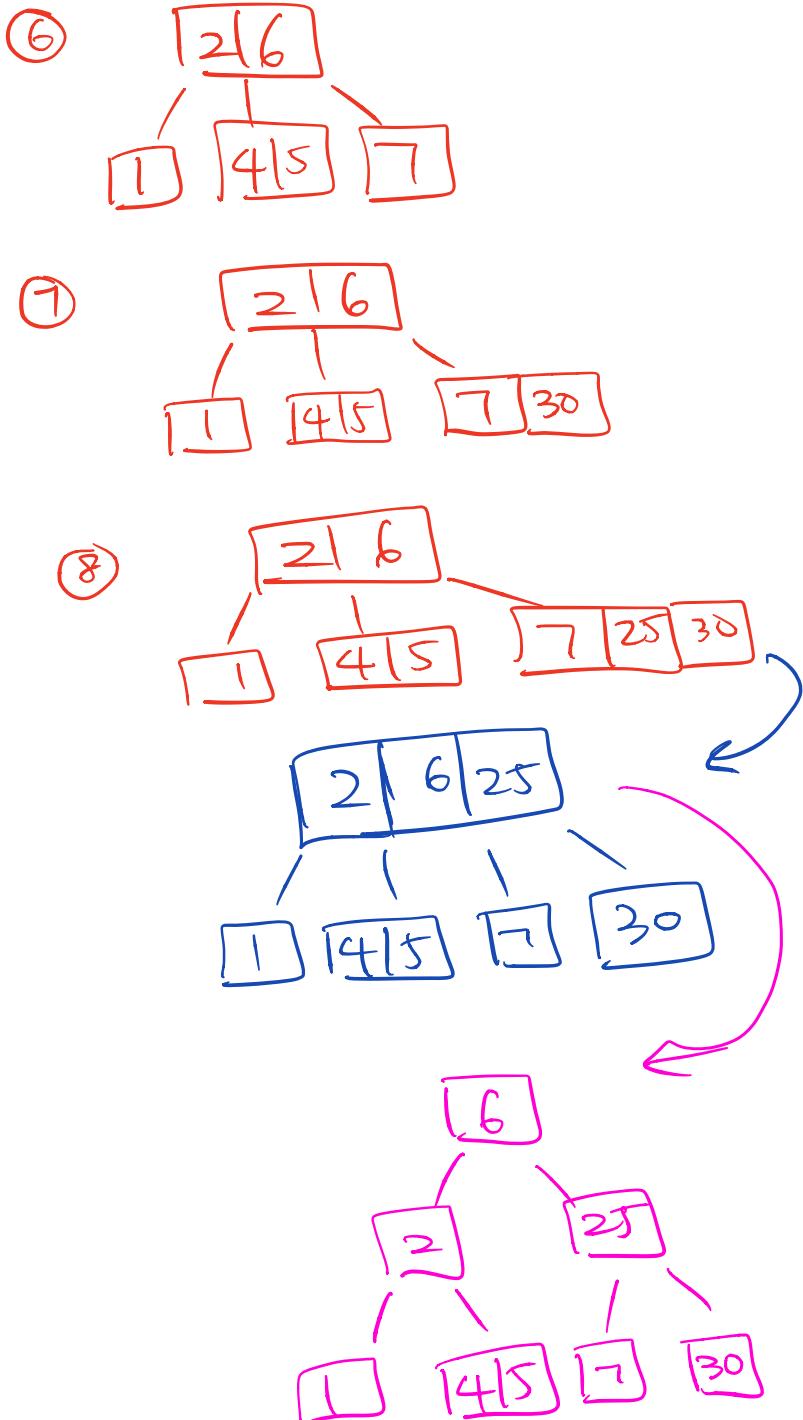
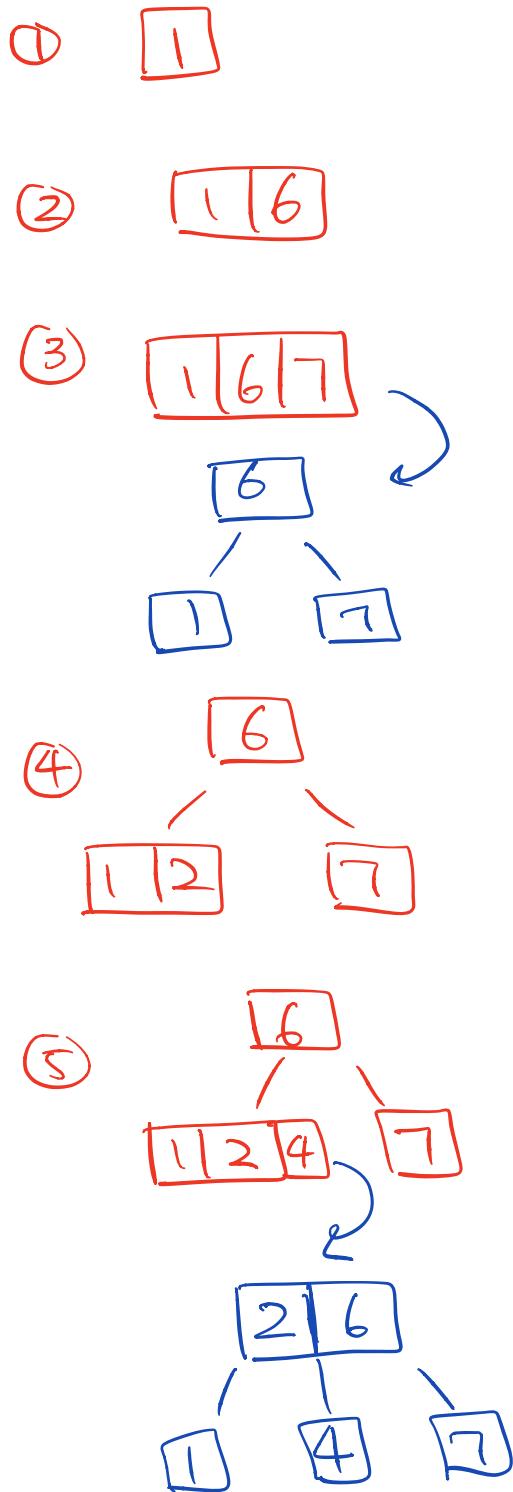
remove the constants

$2^{N-1}N > 2^{N-1} \cdot 1$,
so the constants could be ignored.

$\Theta(2^N)$

2 B-Trees

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



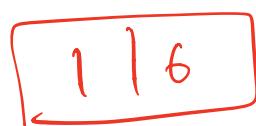
373
2. Construct a 2-3-4 B-Tree with the same keys above.

1, 6, 7, 2, 4, 5, 30, 25

①



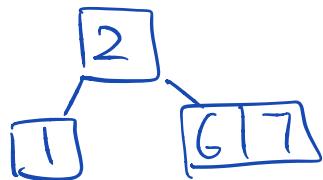
②



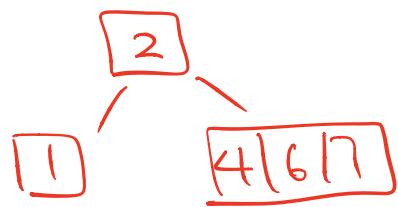
③



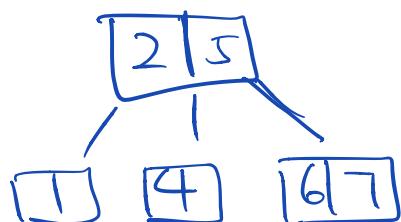
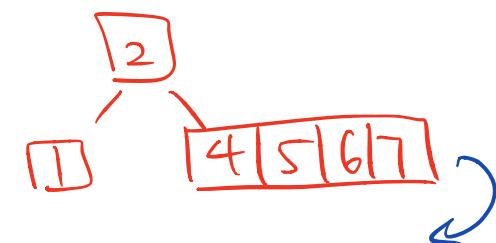
④



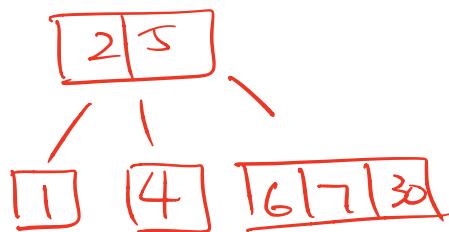
⑤



⑥



⑦



⑧

