Divide & Conquer Runtime Analysis

General Workflow

- Divide the original problem into x pieces of subproblems with **equal size**
- Solve the subproblems recursively (until we hit the base case)
- After getting answers for all subproblems, combine their result to get final solution
- When calculating the runtime, need to consider time for splitting and combining
  - Remember splitting can take more than O(1) time

```
/** Sort b[h..k]. */
public static void mergeSort(int[] b, int h, int k) {
    if (h >= k)  return;  // if b[h..k] has size 0 or 1

    int e= (h + k) / 2;
    mergeSort(b, h, e);          // Sort b[h..e]
    mergeSort(b, e + 1, k);   // Sort b[e+1..k]
    merge(b, h, e, k);           // Merge the 2 segments
}
```

Equation for the runtime

- General form: $T(n) \le qT(n/p) + O(\text{combine})$, or $T(n) \le qT(n/p) + O(\text{combine})$, where
    - q is the number of subproblems we divide the original problem into
    - p is the reciprocal of the size of each subproblem
    - O(combine) is the runtime for the last combine step, can be anything
- Notice this is not the final runtime which in the form of O(·), we need to solve

```
/** Sort b[h..k]. */
public static void mergeSort(int[] b, int h, int k) {
   if (h >= k)  return;  // if b[h..k] has size 0 or 1

   int e= (h + k) / 2;
   mergeSort(b, h, e);         // Sort b[h..e]
   mergeSort(b, e + 1, k);  // Sort b[e+1..k]
   merge(b, h, e, k);          // Merge the 2 segments
}
```

Each subproblem has size of n/2, p = 2

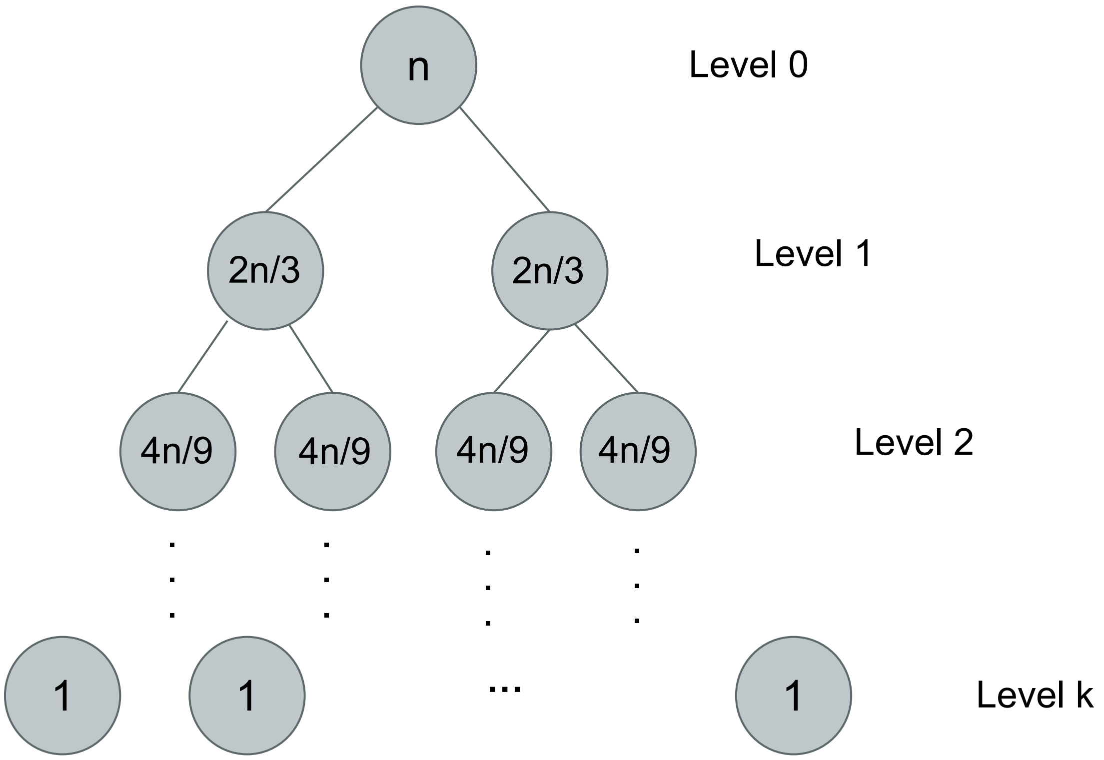
Two subproblems, q=2

Conquer step takes O(n)

Master Theorem

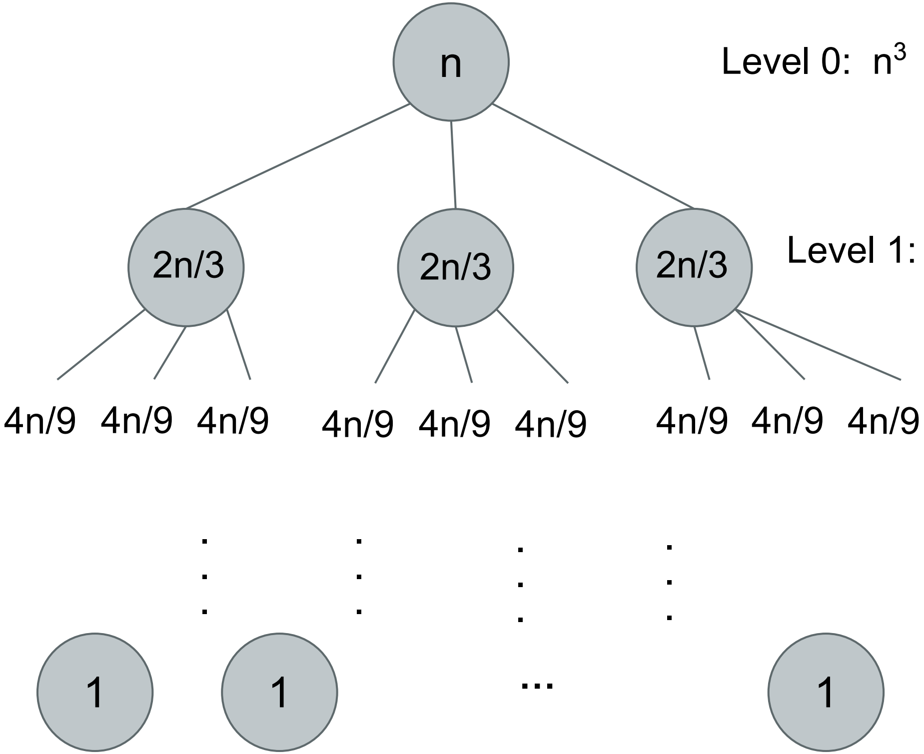
- $T(n) = qT(n/p) + O(n)$
- Three cases:
  - $q < p \rightarrow O(n)$
  - $q = p \rightarrow O(n \log n)$ – merge sort
  - $q > p \rightarrow O(n^{\log_p q})$ – integer multiplication, q=3 and p=2
- Credit to Pete and Sanjit :)

Unrolling Method – calculate the depth



- To calculate size:
  - 0: $n^*(⅔)^0$
  - 1: $n^*(⅔)^1$
  - i: $n^*(⅔)^i$
  - k: $n^*(⅔)^k$
- Then we want $n^*(⅔)^k = 1$
  - $n = (3/2)^k$
  - $k = \log_{3/2} n$
- If we have $(a/b)n$, then we get $k = \log_{b/a} n$
- Notice how this only relates to the size of the subproblem

Unrolling Method – set up equation



Level 0: n^3

Level 1: $3^1(\frac{2}{3}n)^3 \neq 3^1(\frac{2}{3})n^3$

$x = \frac{2}{3}n$ here

Level 2: $3^2[(\frac{2}{3})^2 n)]^3$

Level k: $3^k[(\frac{2}{3})^k n)]^3$

$T(n) \leq 3T(2n/3) + O(n^3)$

$T(x) \leq 3T(2x/3) + O(x^3)$

To get a total runtime complexity, we need to sum over all the runtime for these subproblems

Unrolling cont.

Level k: $3^k[(⅔)^k n)]^3$

$$\sum_{i=0}^{k} 3^i(\tfrac{2}{3})^{3i} n^3 = \sum_{i=0}^{k} 3^i(\tfrac{8}{27})^i n^3 = \sum_{i=0}^{k} (\tfrac{8}{9})^i n^3 = n^3 \sum_{i=0}^{k} (\tfrac{8}{9})^i$$

$$= O(n^3)$$

$$\sum_{i=0}^{k} r^i$$

$$r = 1 \Rightarrow O(k)$$

$$r < 1 \Rightarrow \le \sum_{i=0}^{\infty} r^i = \frac{1}{1-r} = O(1)$$

$$r > 1 \Rightarrow \frac{r^{k+1}-1}{r-1} \le r^{k+1} = O(r^k)$$

$$S = 1 + r + r^2 +\dots + r^k$$
$$r * S = r + r^2 +\dots + r^{k+1}$$
$$rS - S = (r-1)S = r^{k+1} - 1$$
$$S = \frac{r^{k+1}-1}{r-1}$$

Substituting Method

- Guess the final runtime, and plug it back into the recurrence to see if the equation works out
  - It sounds weird, but it's basically using induction
- Useful when we have a general form for the solution, but we want to figure out a more exact value for parameters
  - instead of O(n), we can bounded with 4n
- Read more on textbook!