OntarioTech Science

Kourosh Davoudi
kourosh@ontariotechu.ca

Lecture 2: Divide and Conquer

CSCI 3070U: Design and Analysis of Algorithms

Learning Outcomes

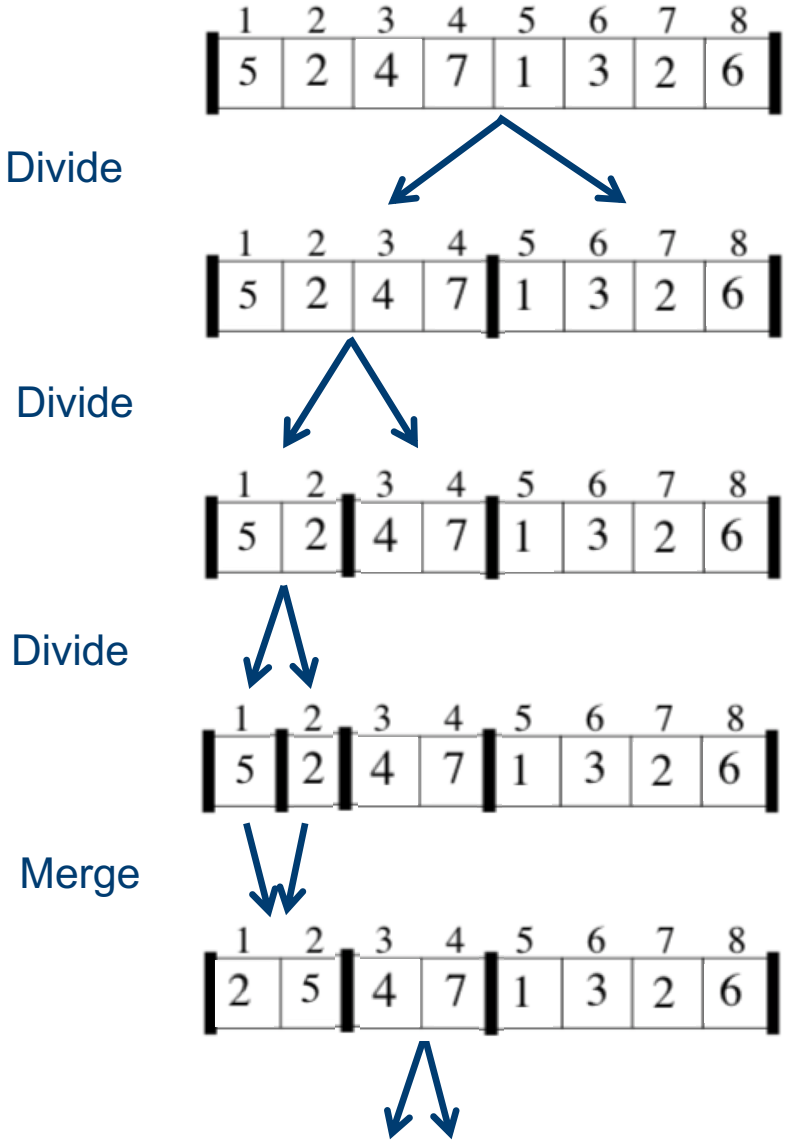
- Designing Algorithm by divide and conquer
  - Case Study: Merge Sort
  - Case Study: Binary Search
- Solving Recurrence Equations
  - Substitution
  - Recursion Tree
  - Master Theorem

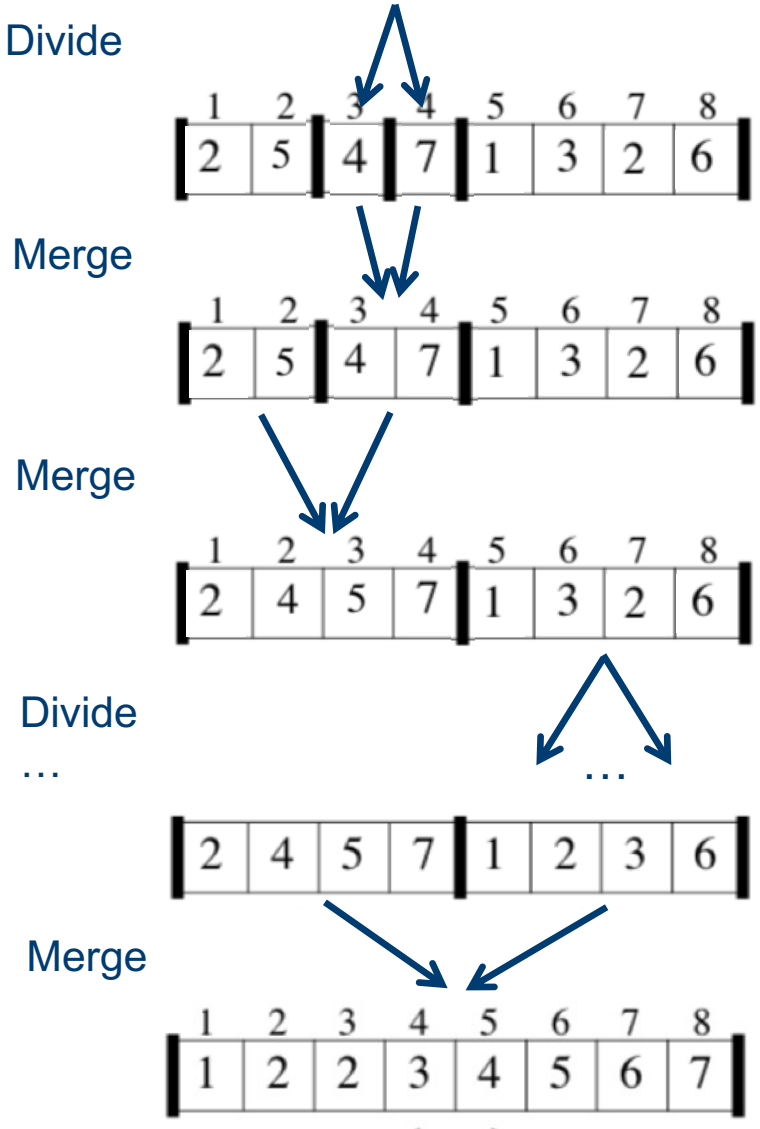
Divide and Conquer

- What is divide and conquer?
  - An approach for designing algorithms
- What is the general divide and conquer scheme?
  - Divide:
    - Divide the problem into sub-problems
  - Conquer
    - Solve these sub-problems recursively
    - Idea: Smaller problems are easier to solve!
  - Combine
    - Combine the subproblems results

Case Study: Merge Sort

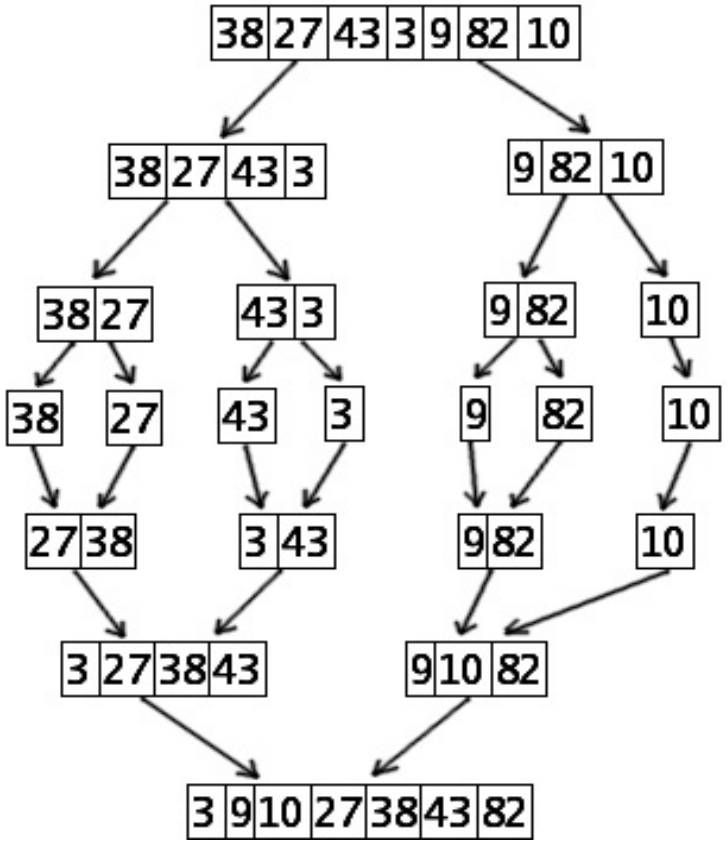
- General Idea:
  - **Divide** by splitting into two subarrays $A[p..q]$ and $A[q+1..r]$, where q is the halfway point of $A[p..r]$
  - **Conquer** by recursively sorting the two subarrays $A[p..q]$ and $A[q+1..r]$
  - **Combine** by merging the two sorted subarrays subarrays $A[p..q]$ and $A[q+1..r]$ to produce a single sorted subarray .


Divide
Divide
Divide
Merge
1 2 3 4 5 6 7 8
5 2 4 7 1 3 2 6
1 2 3 4 5 6 7 8
5 2 4 7 1 3 2 6
1 2 3 4 5 6 7 8
5 2 4 7 1 3 2 6
1 2 3 4 5 6 7 8
5 2 4 7 1 3 2 6
1 2 3 4 5 6 7 8
2 5 4 7 1 3 2 6


Divide
Merge
Merge
Divide
…
…
Merge
1 2 3 4 5 6 7 8
2 5 4 7 1 3 2 6
1 2 3 4 5 6 7 8
2 5 4 7 1 3 2 6
1 2 3 4 5 6 7 8
2 4 5 7 1 3 2 6
2 4 5 7 1 2 3 6
1 2 3 4 5 6 7 8
1 2 2 3 4 5 6 7

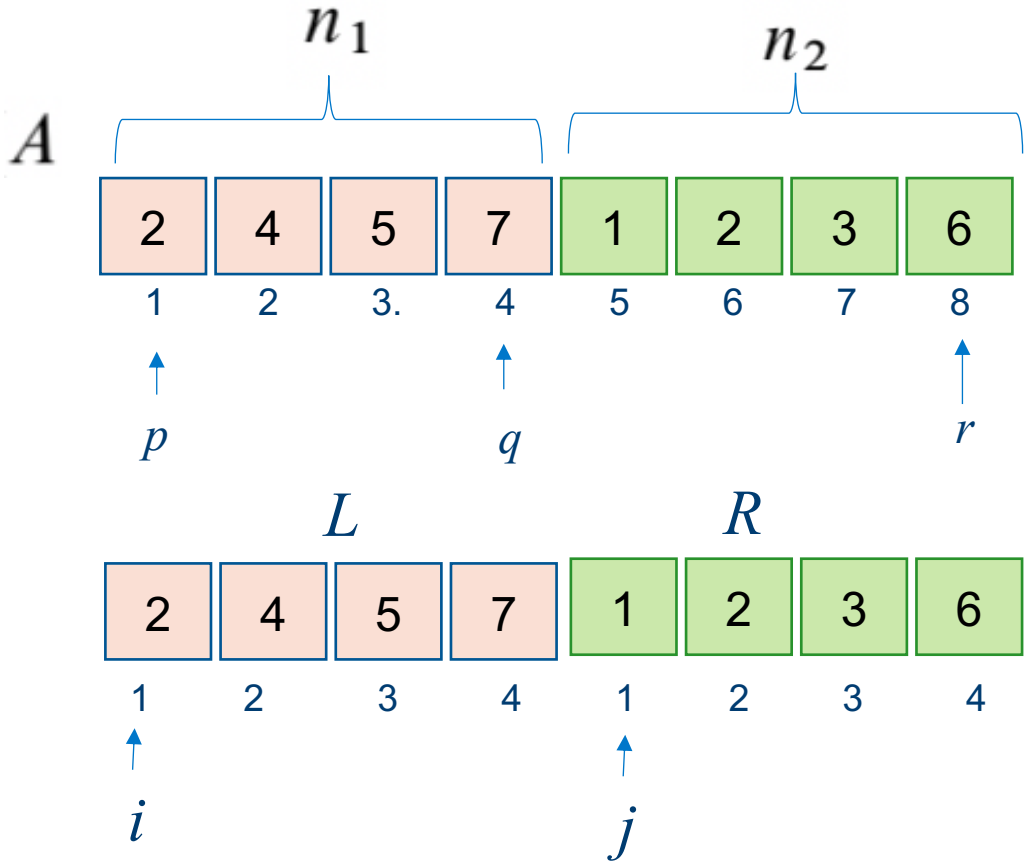
Case Study: Merge Sort

```
MERGE-SORT(A, p, r)
  if p < r                          // check for base case
      q = ⌊(p + r)/2⌋               // divide
      MERGE-SORT(A, p, q)           // conquer
      MERGE-SORT(A, q + 1, r)       // conquer
      MERGE(A, p, q, r)             // combine
```



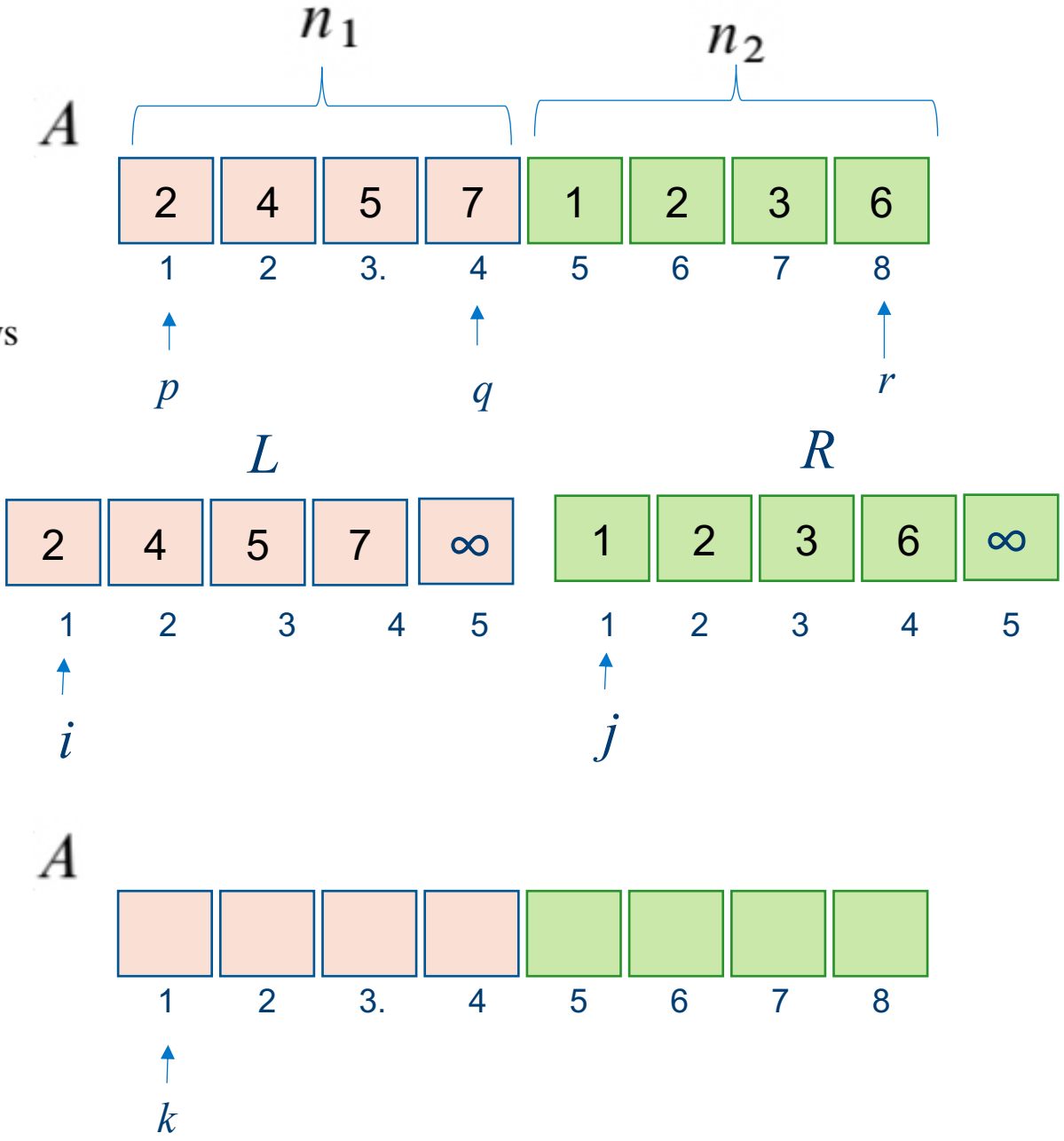
Case Study: Merge Sort

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```



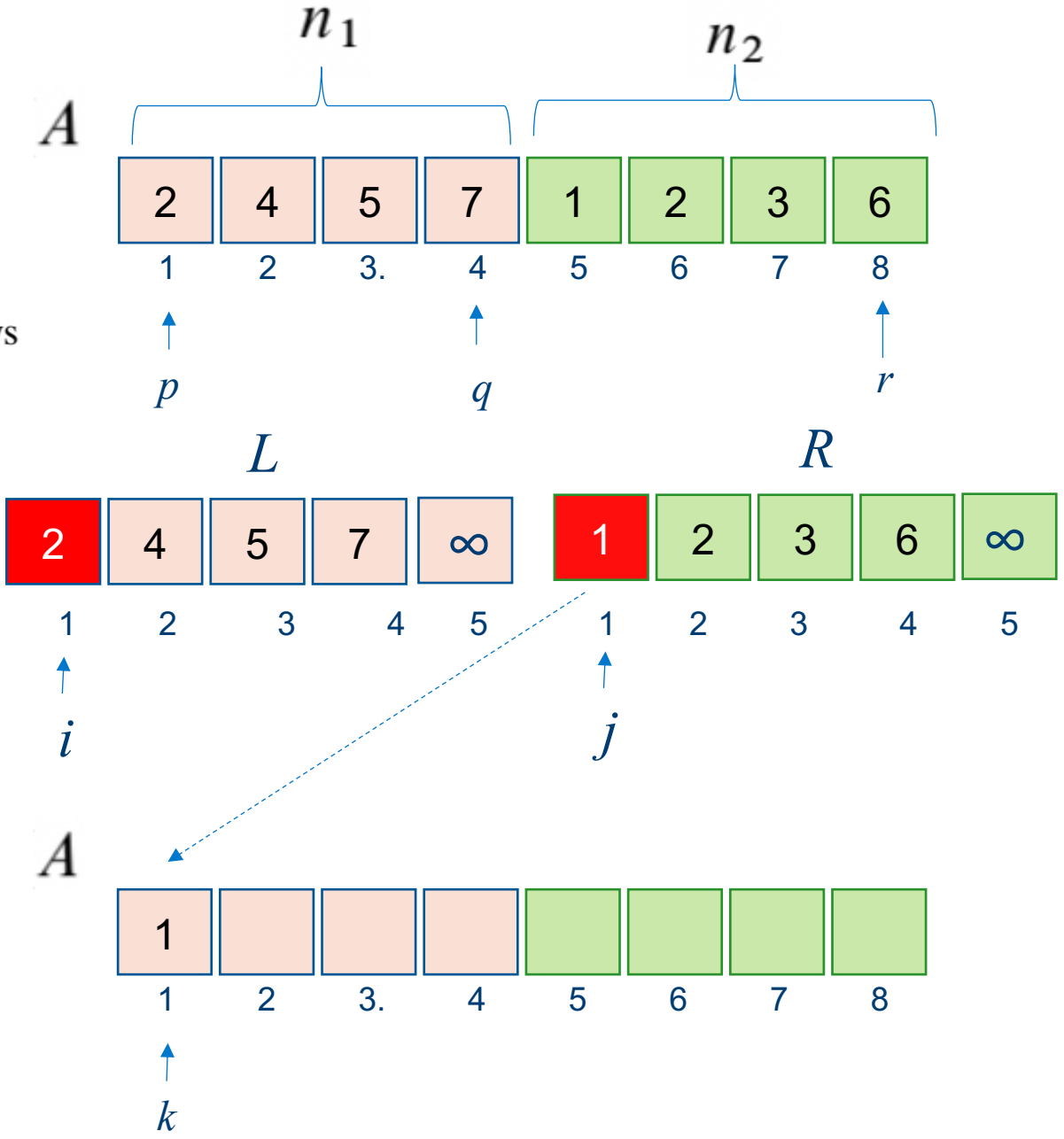
Case Study: Merge Sort

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```



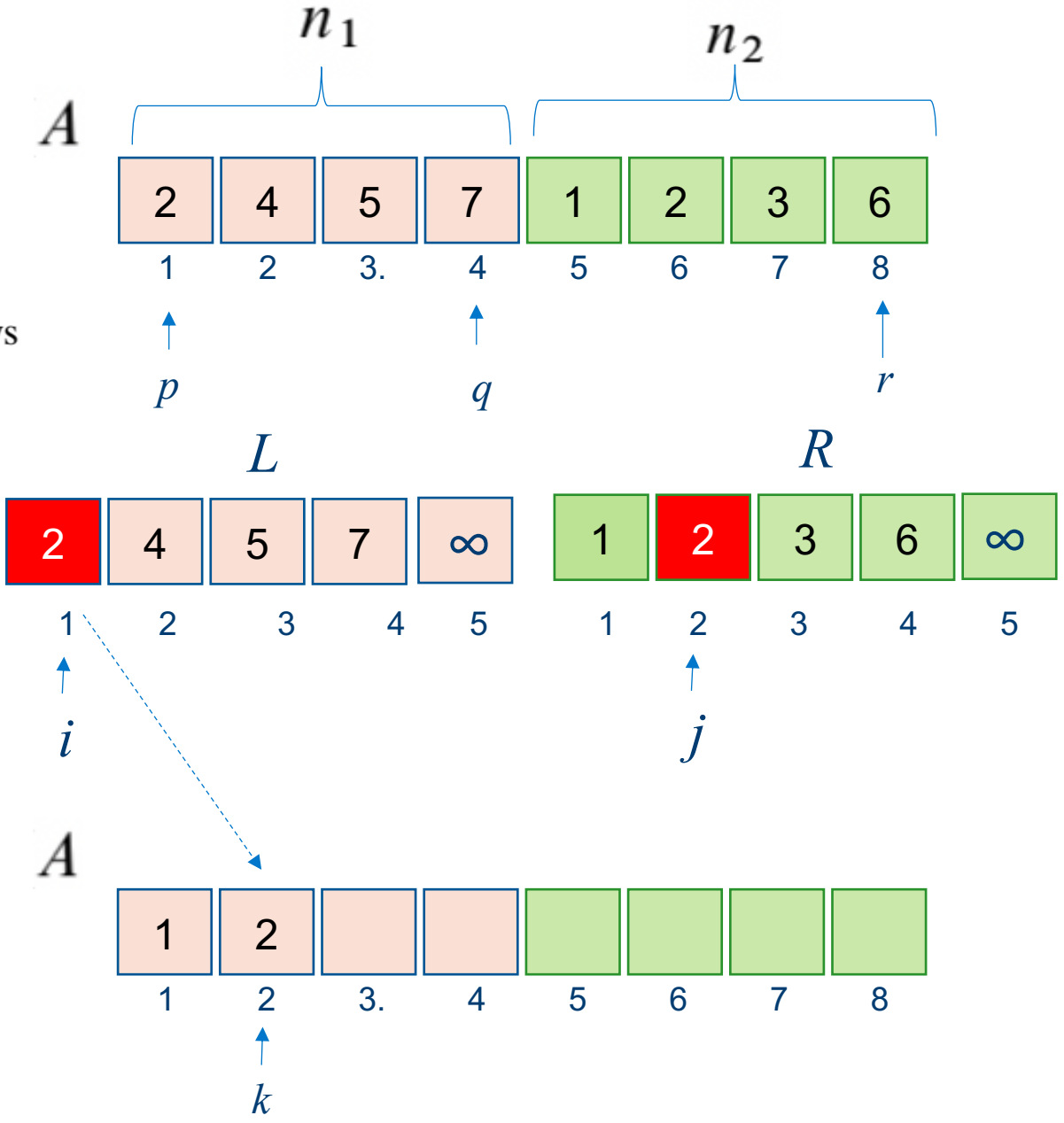
Case Study: Merge Sort

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```



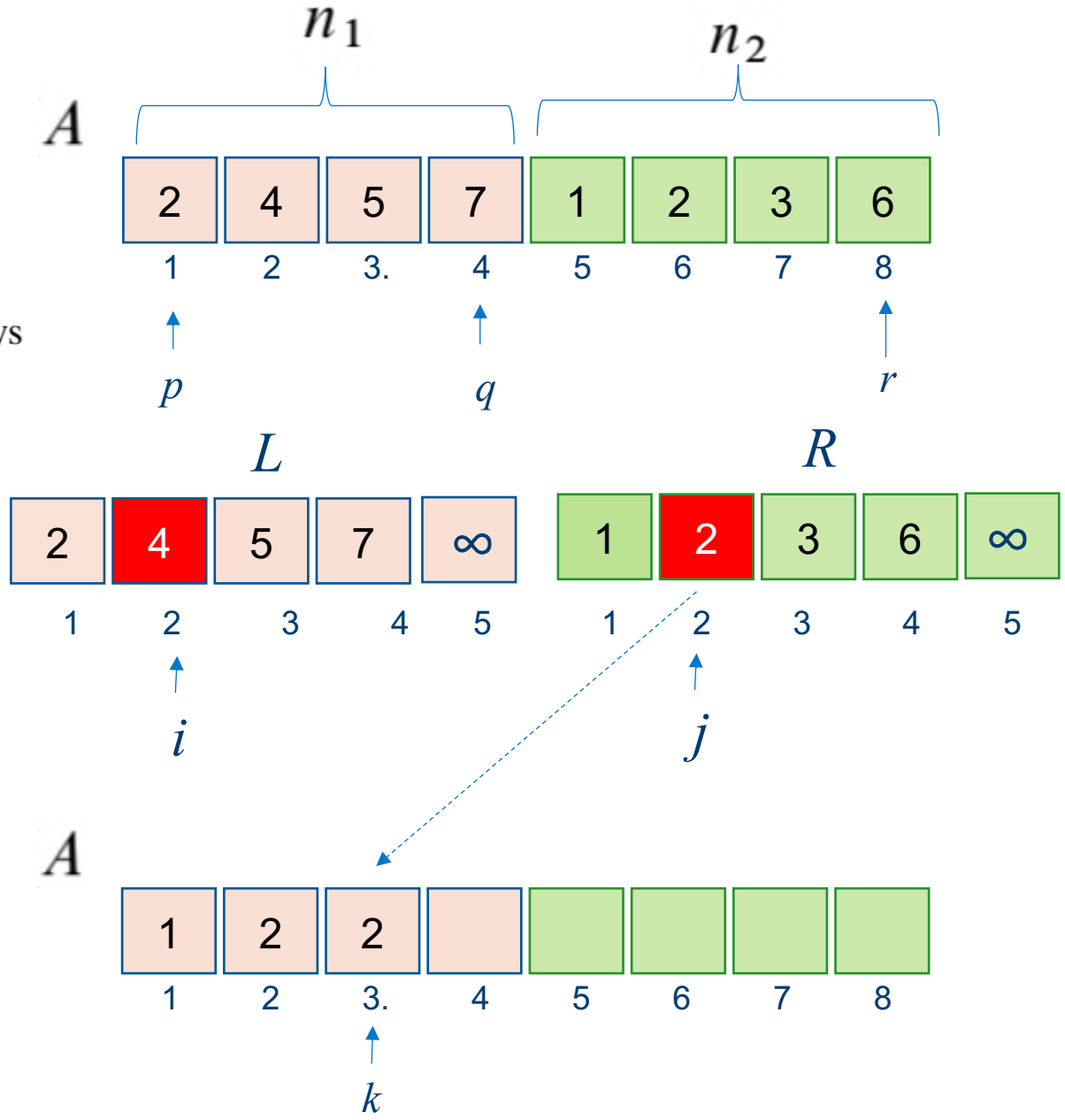
Case Study: Merge Sort

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```



Case Study: Merge Sort

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
  for i = 1 to n1                                   Θ(n1)
      L[i] = A[p + i − 1]
  for j = 1 to n2                                   Θ(n2)
      R[j] = A[q + j]                                          Θ(n)
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r                                    Θ(n)
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```



Merge Sort Time Complexity

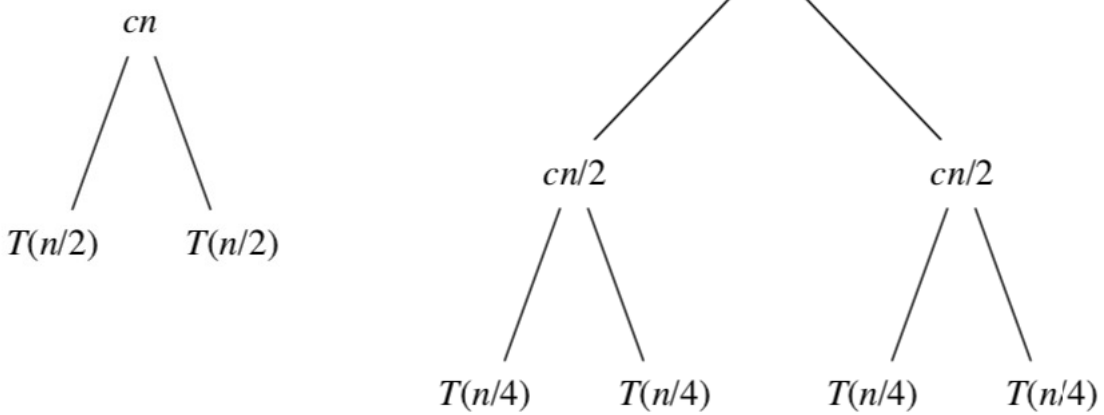
```
MERGE-SORT(A, p, r)
 if p < r
     q = ⌊(p + r)/2⌋                    Θ(1)
     MERGE-SORT(A, p, q)                T(n/2)
     MERGE-SORT(A, q + 1, r)            T(n/2)
     MERGE(A, p, q, r)                  Θ(n)
```

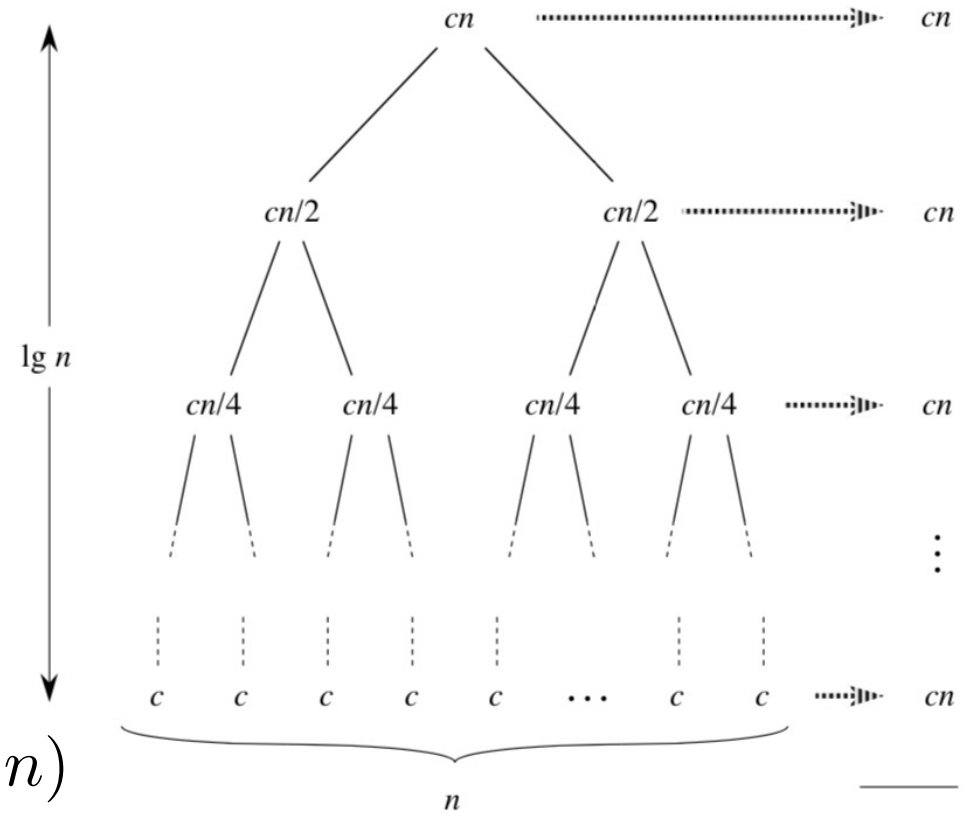
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Solving Merge Sort Recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$





There are $\lg n + 1$ levels (height is $\lg n$).

$$T(n) = cn\log(n) + cn \in \Theta(n \log n)$$

Case Study: Binary Search

- Background
  - Given a sorted array of numbers, and a number, how do you find the location of this number in the array?

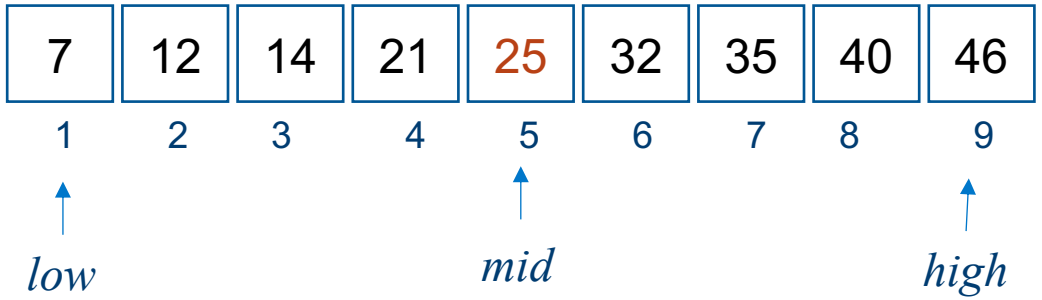
  - I don't care, I will check numbers in the array till find (if any) the location of that number !
    - The worst case complexity of your algorithm is: $\Theta(n)$

Case Study: Binary Search

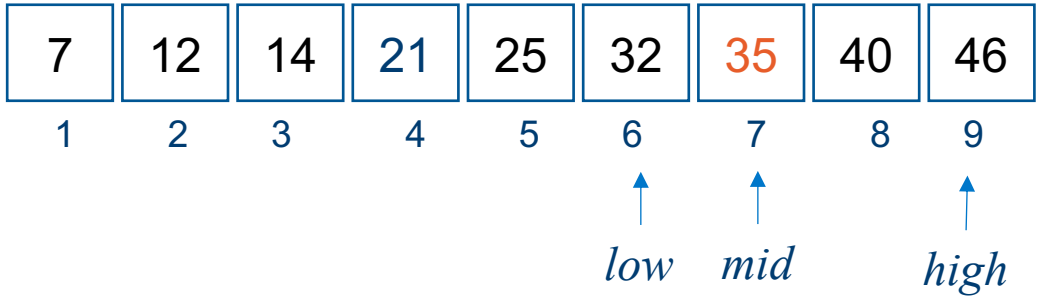
- General Idea:
  - Takes a sorted array A, a value v, and a range $[low\ ..\ high]$ of the array, in which we search for the value v
  - check the midpoint of the sequence against v and eliminate half of the sequence from further consideration.

Case Study: Binary Search

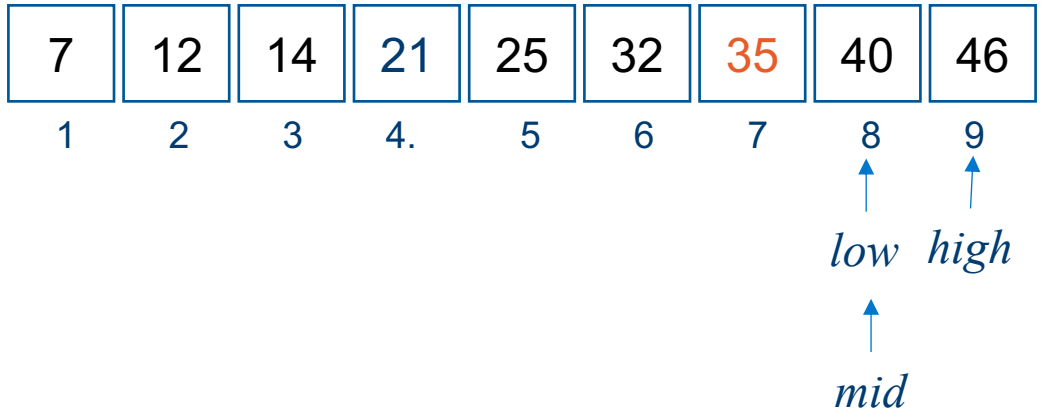
$v = 40$



$$mid = \lfloor \frac{9+1}{2} \rfloor = 5$$



$$mid = \lfloor \frac{9+6}{2} \rfloor = 7$$



$$mid = \lfloor \frac{9+8}{2} \rfloor = 8$$

Binary Search Time Complexity

```
RECURSIVE-BINARY-SEARCH(A, v, low, high)
  if low > high                                                          c
      return NIL
  mid = ⌊(low + high)/2⌋
  if v == A[mid]                                                         c
      return mid
  elseif v > A[mid]                                                      c
      return RECURSIVE-BINARY-SEARCH(A, v, mid + 1, high)                T(n/2)
  else return RECURSIVE-BINARY-SEARCH(A, v, low, mid − 1)                T(n/2)
```

$$T(n) = T(n/2) + c$$

Binary Search Time Complexity

$$
\begin{aligned}
T(n) &= T(n/2) + c \\
&= T(n/4) + c + c \\
&= T(n/8) + c + c + c \\
&\cdots \\
&= T(n/2^k) + kc
\end{aligned}
$$

$$k = \log_2(n)$$

$$
\begin{aligned}
T(n) &= T(1) + c\log_2(n) \\
&= c + c\log_2(n) \in \Theta(\log n)
\end{aligned}
$$

How to Solve Recurrence Equation?

- Substitution Method
  - Guess, then use induction to prove it
- Recursion Tree
  - Draw the recurrence as a tree and use geometry (i.e. tree height and width) to estimate total cost
- Master Theorem
  - Applies to recurrences of the form: $T(n) = aT(n/b) + f(n)$

Substitution Method

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

(1) Guess: $T(n) = n \lg n + n$

(2) Induction:

Basis: $n = 1 \Rightarrow n \lg n + n = 1 = T(n)$

Inductive step: Inductive hypothesis is that $T(k) = k \lg k + k$ for all $k < n$

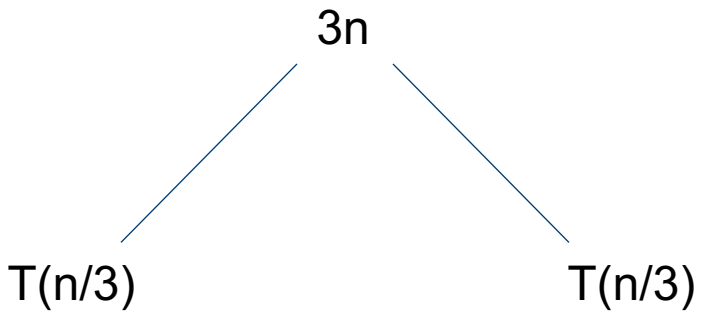
$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \\ &= 2\left(\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}\right) + n \\ &\quad \text{(by inductive hypothesis)} \\ &= n\lg\frac{n}{2} + n + n \\ &= n(\lg n - \lg 2) + n + n \\ &= n\lg n - n + n + n \\ &= n\lg n + n\ . \end{aligned}$$

Recursion Tree

$$T(n) = 2T(n/3) + 3n$$
$$T(1) = 1$$

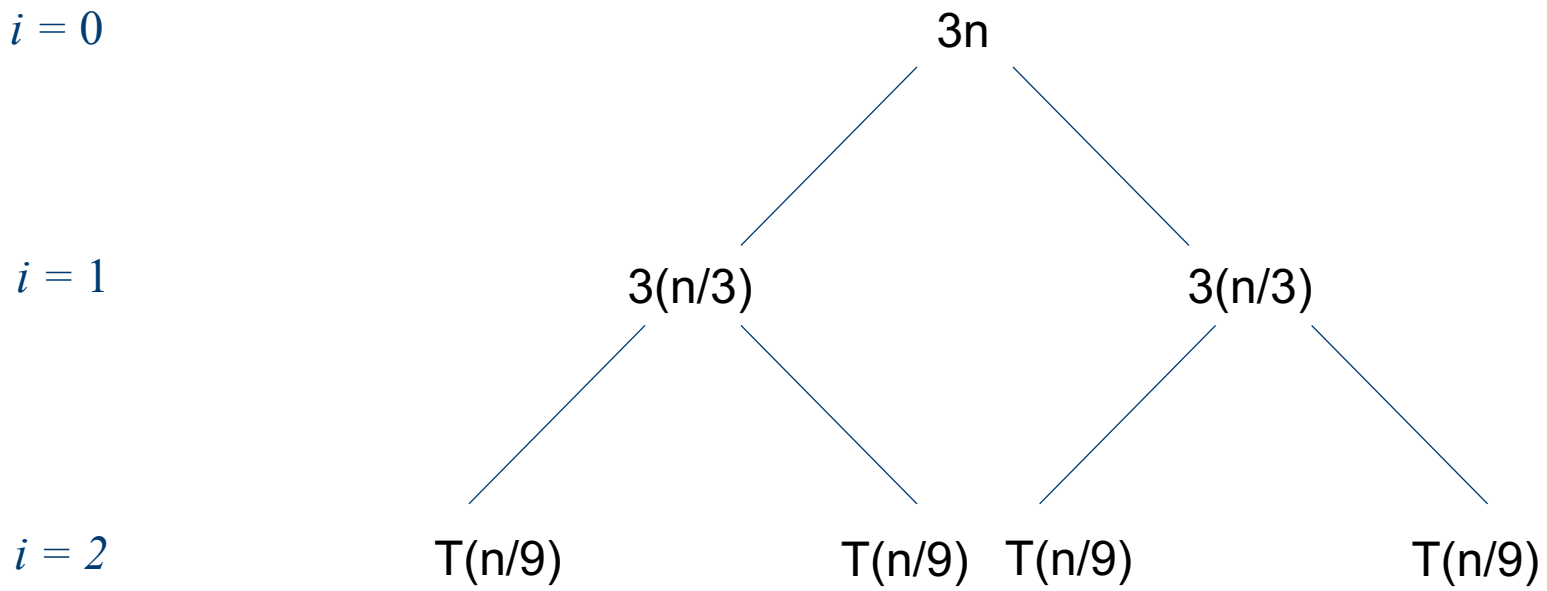
$i = 0$

$i = 1$



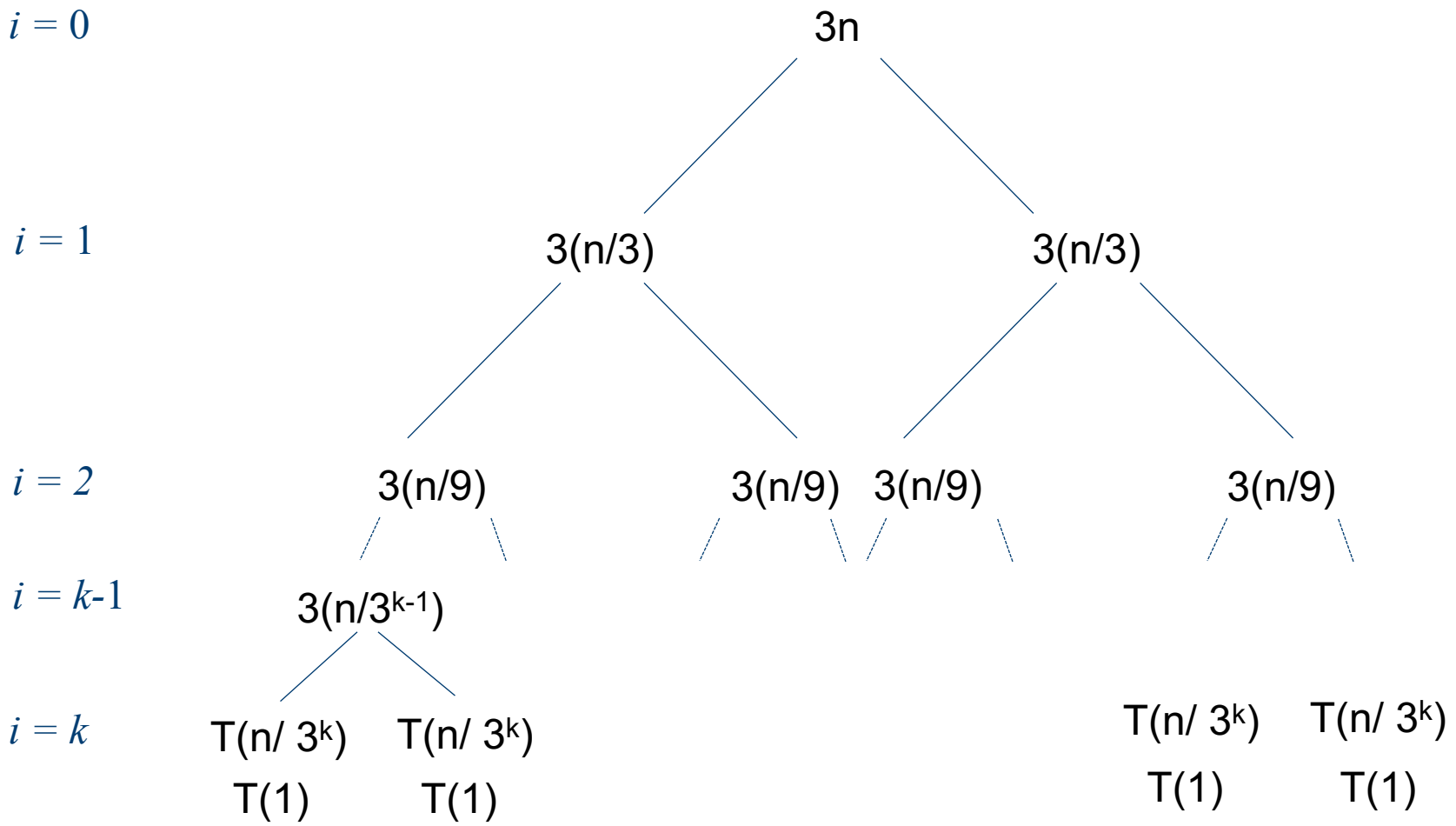
Recursion Tree

$$T(n) = 2T(n/3) + 3n$$
$$T(1) = 1$$



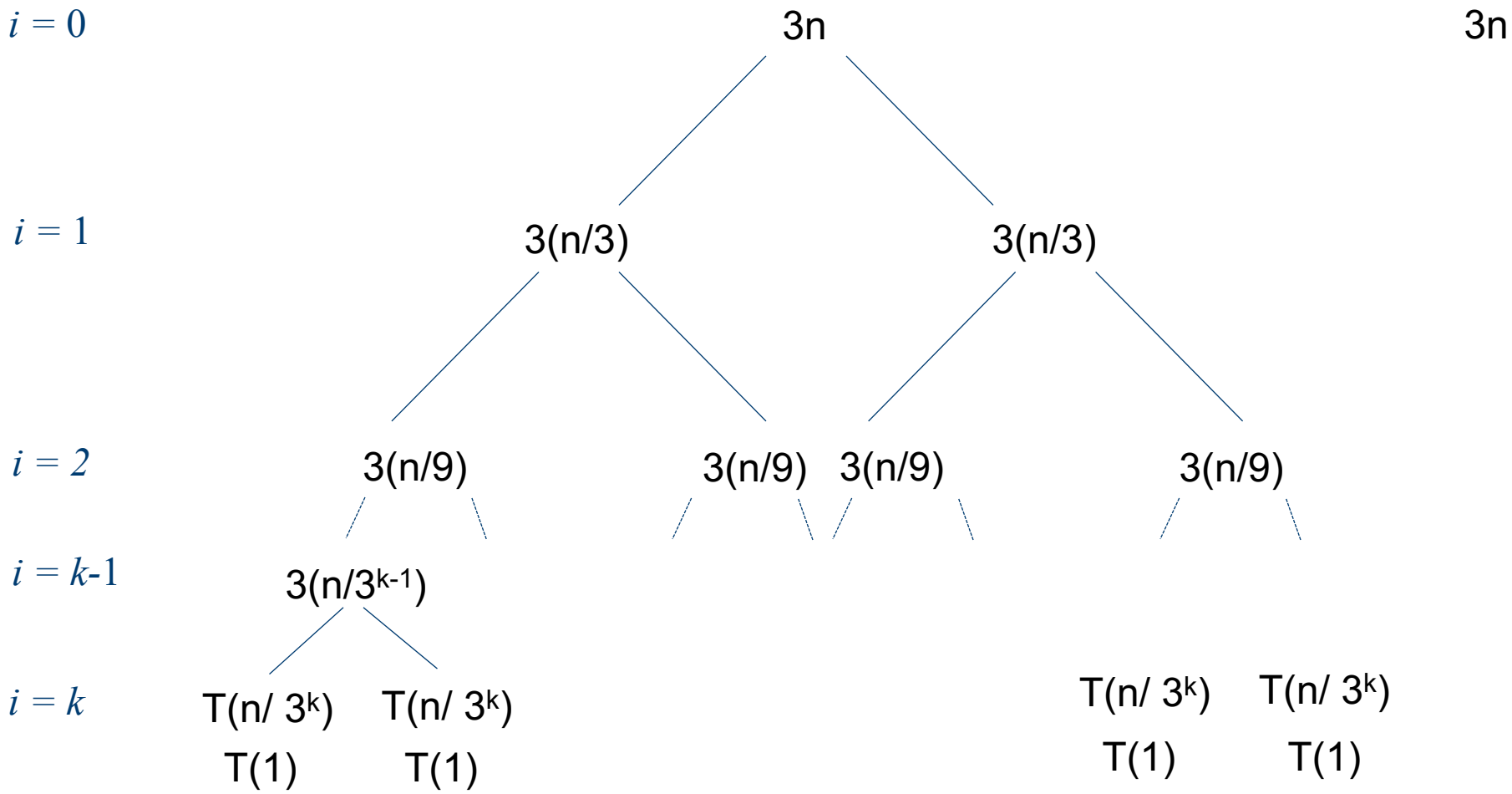
Recursion Tree

$$T(n) = 2T(n/3) + 3n$$
$$T(1) = 1$$



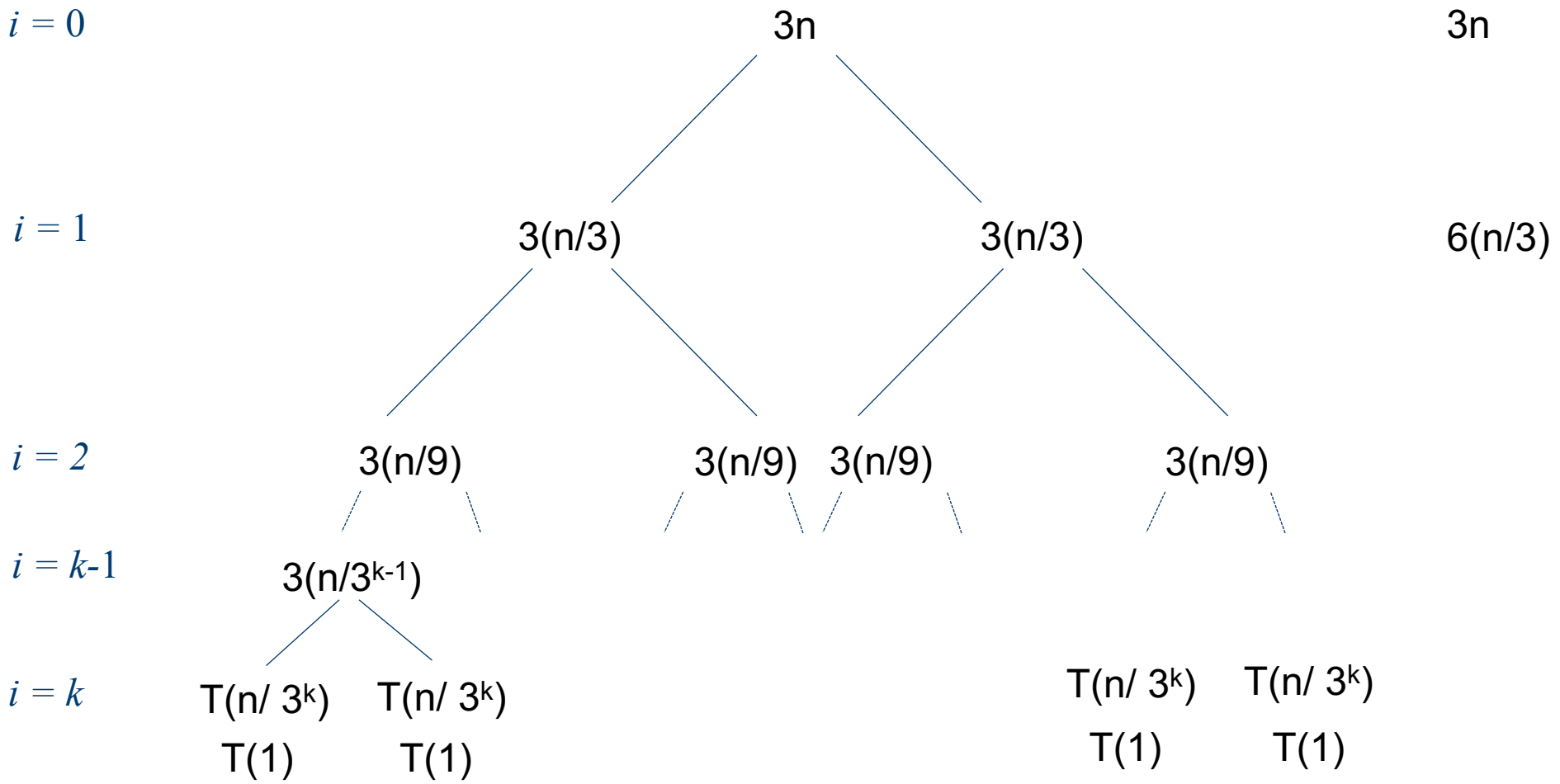
Recursion Tree

$$T(n) = 2T(n/3) + 3n$$
$$T(1) = 1$$



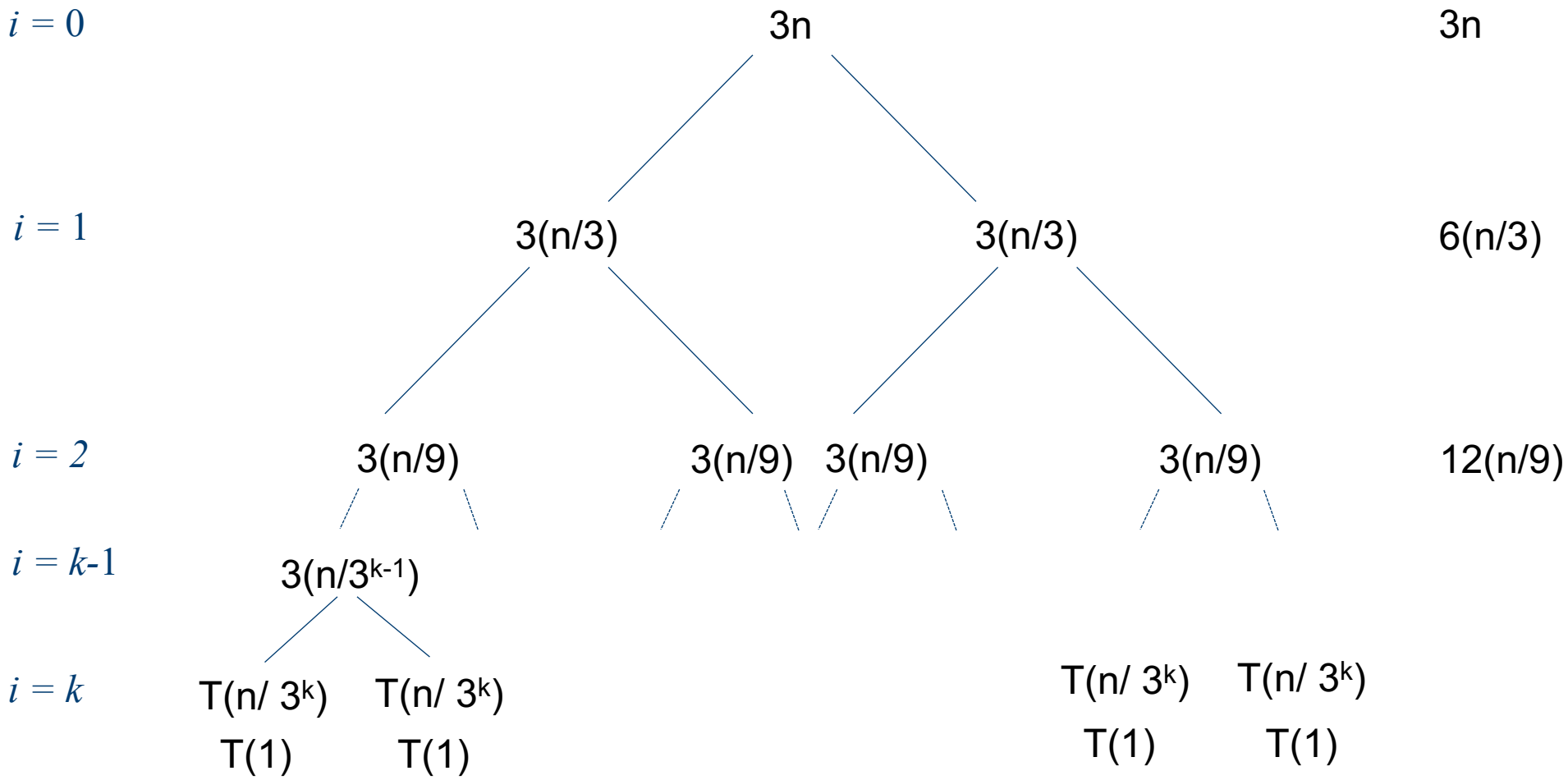
Recursion Tree

$$T(n) = 2T(n/3) + 3n$$
$$T(1) = 1$$



Recursion Tree

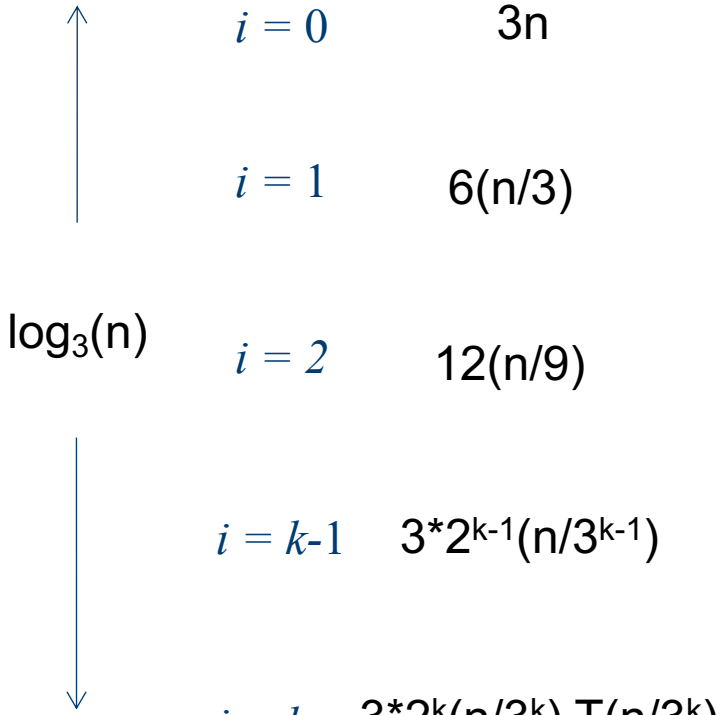
$$T(n) = 2T(n/3) + 3n$$
$$T(1) = 1$$



Recursion Tree

$$T(n) = 2T(n/3) + 3n$$

$$1 + q + q^2 + \cdots + q^k = \frac{1 - q^{k+1}}{1 - q}$$



$$n/3^k = 1 \quad \Rightarrow \quad k = log_3(n)$$

$$T(n) = 3n(1 + 2/3 + 4/9 + \cdots + (2/3)^{\log_3(n)})$$

$$T(n) = 3n\frac{1 - (2/3)^{\log_3(n)+1}}{1 - 2/3}$$

$$= 9n(1 - (2/3)(2/3)^{\log_3(n)})$$

$$a^{\log_b(n)} = n^{\log_b(a)}$$

$$= 9n(1 - (2/3)(n)^{\log_3(2/3)})$$

$$\log(a/b) = \log(a) - \log(b)$$

$$= 9n - 9(2/3)\ n^{\log_3(2)} \in \Theta(n)$$

Master Theorem

$$T(n) = aT(n/b) + f(n) \qquad \text{where } a \geq 1, b > 1, \text{ and } f(n) > 0$$

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.
($f(n)$ is polynomially smaller than $n^{\log_b a}$.)

Solution: $T(n) = \Theta(n^{\log_b a})$

Case 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k \geq 0$.

Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$

Simple case: $k = 0 \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$

$af(n/b) \leq cf(n)$ for some constant $c < 1$

($f(n)$ is polynomially greater than $n^{\log_b a}$.)

Solution: $T(n) = \Theta(f(n))$

Master Theorem

- **Example 1:** $T(n) = 4T(n/2) + n$

$$T(n) = aT(n/b) + f(n)$$

  - a = 4
  - b = 2

$$\log_b(a) = 2$$

  - $f(n) = n = O(n^{2-\epsilon})$? for some $\epsilon > 0$?
    - Yes! Specifically, for $\epsilon = 1$
  - Thus, case 1 applies and $T(n) = \Theta(n^2)$

Master Theorem

- Example 2: $T(n) = 4T(n/2) + n^2$

$$T(n) = aT(n/b) + f(n)$$

  - a = 4
  - b = 2

$$\log_b(a) = 2$$

  - Is $f(n) = n^2 = O(n^{2-\epsilon})$, for some $\epsilon > 0$?
    - No
  - Is $f(n) = n^2 = \Theta(n^2)$?
    - Yes!
  - Thus, case 2 applies and $T(n) = \Theta(n^2 \log(n))$

Master Theorem

- Example 3: $T(n) = 4T(n/2) + n^4$

$$T(n) = aT(n/b) + f(n)$$

  - a = 4
  - b = 2

$$\log_b(a) = 2$$

  - $f(n) = n^4 = O(n^{2-\epsilon})$, for some $\epsilon > 0$? No
  - $f(n) = n^4 = \Theta(n^2)$? No
  - $f(n) = n^4 = \Omega(n^{2+\epsilon})$, for some $\epsilon > 0$? Yes
  - $af(n/b) = 4(n/2)^4 \leq cf(n) = cn^4$, for some c < 1 ? Yes $c \geq 1/4$

  - Thus, case 3 applies and $T(n) = \Theta(n^4)$

Wrap-up

- We learned
  - Foundation of divide and conquer
    - Merge sort
    - Binary search
  - How to solve the recurrent equation:
    - Substitution
    - Recursion Tree
    - Master Theorem