Lecture 9

Merge Sort

HW 9 - Checkins July 22

Insertion Sort

Selection Sort

Time Complexity of Insertion Sort
→ $O(n^2)$

Time Complexity of Selection Sort
→ $O(n^2)$

$O(n^2)$ - not very big.

We will see merge-sort which takes $O(n \log n)$

$n \log n$ is better than n^2

$n \cdot \log n$ $\qquad$ $n \cdot n$

$\log n$ grows slower than n.

$n \log n \in O(n^2)$

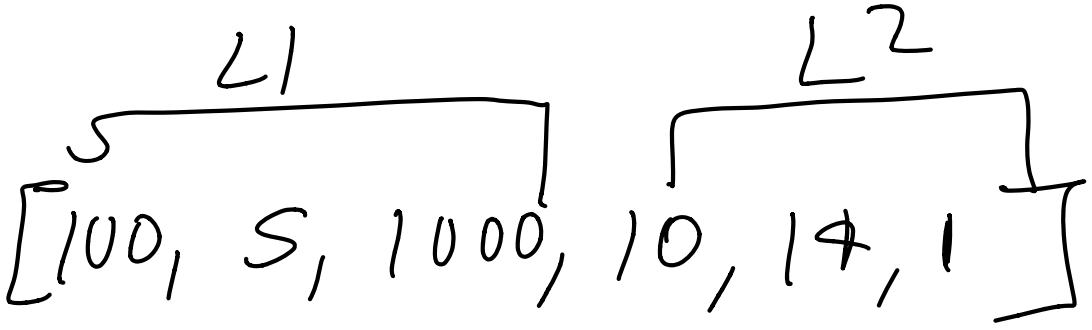
But

$n^2 \notin O(n \log n)$

n^2 grows faster than $n \log n$

Suppose list L.

1. Divide the list into two halves.
   L1, L2
2. Recursively sort them (L1, L2)
3. Merge them into a sorted list.



L1: [100, 5, 1000]

L2: [10, 14, 1]

(merge-sort L1) (merge-sort L2)

(merge-sort l1) → [5, 100, 1000] m1

(merge-sort l2) → [1, 10, 14] m2

(merge m1 m2)

→ [1, 5, 10, 14, 100, 1000]

(merge → a function that takes two sorted lists, and combines them into one sorted list.

Step 1:

How do you divide a list into two halves?

first-half which will give me the first-half of the list.

Define a helper function "head"
Input 1 → mylist
Input 2 → integer (K)

Output → A new list containing first K elements.

Example

[2, 13, 4, 7, 8, 5]

(head (list 2 13 4 7 8 5)
3)
⇒ (2 13 4)

(head my-list (quotient (length my-list)
2))

How to define head.

[2, 13, 4, 6, 100, 200] ← my-list
4

First extract the first element

2 (head (rest my-list) 3)

(cons (first my-list) (head (rest my-list) 3))

Recursive extract k-1 elements
(rest list)

Then combine using cons.

Base case: Want zero elements from the list.

```
(: first-half : (Listof Integer) -> (Listof
                 Integer))

(define (first-half my-list)
  (local
    {(: head : (Listof Integer) Integer
        -> (Listof Integer))
     (define (head mlist k)
       (cond -> op1 [(empty? mlist) empty]
             [(= k 0) empty] -> op1
             [else (cons (first mlist)
                         (head (rest mlist) (- k 1)))]))}
    (head my-list (quotient (length my-list) 2))))
```

input list

number of elements to extract

n-1

k-1

n

n.

Remark

(1 2 3) = 2/3.

(quotient 7 2) ⇒ 3.

Odd number of elements.
We are just rounding down.

Time complexity

Let T(n) be time-complexity of first half.

We need to compute the time complexity of head

Head has two inputs

length and K. → how many elements you want.

Time complexity of head is $T(n, k) = T(n-1, k-1)$

+ constant.

$T(n, k) = T(n-1, k-1)$

+ constant.

$k = 0$

$T(n, 0) =$ constant.

$$T(n, k) = T(n-1, k-1) + C$$

$$T(n, 0) = C.$$

$$T(n, k) = T(n-1, k-1) + C$$

$$= 2C + T(n-2, k-2)$$

$$= 3C + T(n-3, k-3)$$

$$= 4C + T(n-4, k-4)$$

$$= \underline{5}C + T(n-5, k-5)$$

$\downarrow$

$n \to 0 \quad k \to 0.$

$\min(n, k)$

$= \min(n, k) \cdot C$ ←

Time-complexity of first-half

is $\min\left(n, \frac{n}{2}\right) \cdot C = \frac{n}{2} C$

$= O(n)$

Now, we need to extract the second half

Helper function

tail

Instead of extracting K elements, we will delete K elements.

[100, 5, 6, 1000, 13, 2]

K = 2.

⇒ [6, 1000, 13, 2].

rest ! element being deleted

(tail (rest mylist) (- K 1))

```
(: second-half: (Listof Integer)
                -> (Listof Integer)

(define (second-half my-list)
  (local
    [(: tail : (Listof Integer) Integer
                -> (Listof Integer))
     (define (tail mlist k)
       (cond
         [(= k 0) mlist]
         [else (tail (rest mlist)
                     (- k 1))]))]
    (tail my-list (quotient (length
                             my-list) 2))))
```

num elements to be deleted

input list

check if mlist is empty

op1

op1

op1

local ends

mlist - size n

(rest mlist) - size n-1

Time complexity of second-half.

Second-half calling the tail function

We need to compute the time complexity of the tail function.

Let's say time complexity of tail is $T(n, k)$

Recurrence:

$$T(n, k) = T(n-1, k-1) + C$$

$$T(n, K) = C \quad \text{if } K = 0$$

$$T(n, k) = \min(n, k) \cdot C.$$

Second-half will call $T(n, k)$

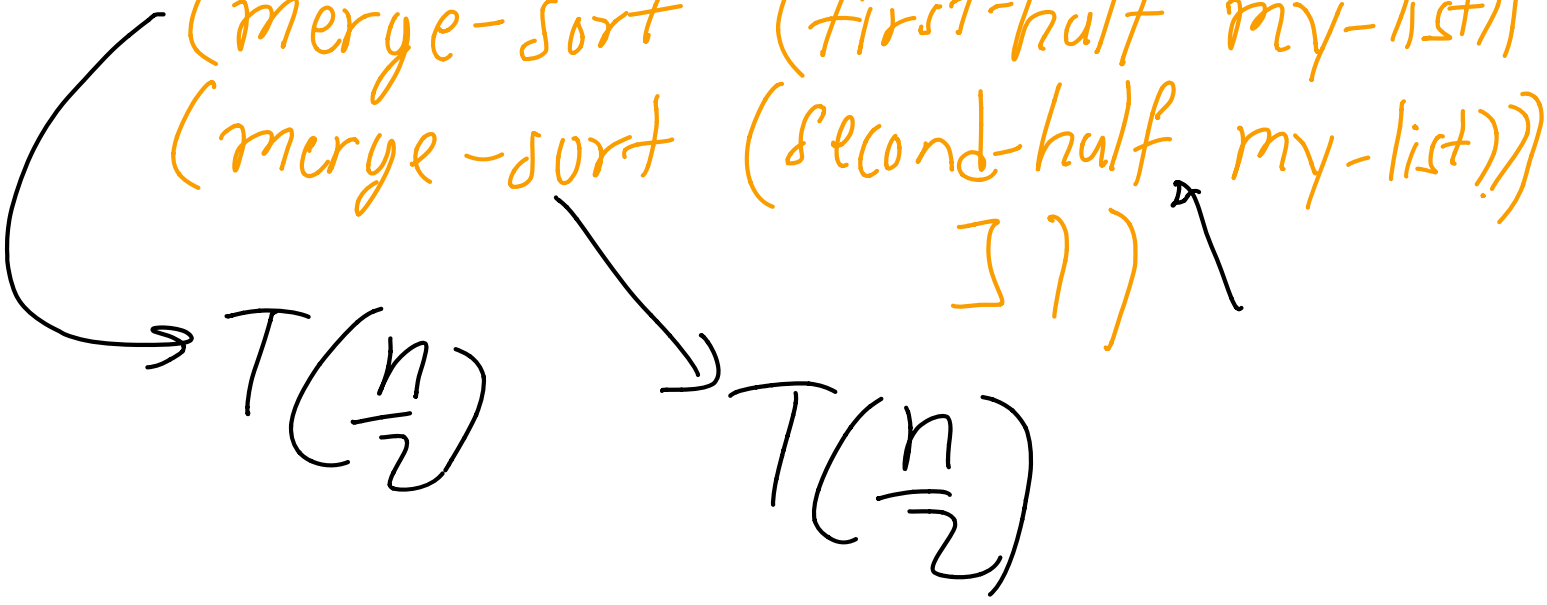
$$T\left(n, \frac{n}{2}\right) = \min\left(n, \frac{n}{2}\right) C$$

$$= \frac{n}{2} \cdot C = O(n)$$

Time complexity of second-half is $O(n)$.

```
(: merge-sort : (List of Integer)
   → (List of Integer))

(define (merge-sort my-list)
  (cond → 1 op
    [(<= (length my-list) 1)
       my-list ] → 1 op
    [else (merge → O(n)
      (merge-sort (first-half my-list))
      (merge-sort (second-half my-list))
                ]))
```



Merge

How to merge two sorted list.

[10, 11, 13, 20, 30]

[13, 14, 15, 20, 25]

Remark

my-list

[3, 100, 5, 1000, 6, 7]

(first-half my-list) → [3, 100, 5]

(second-half my-list) → [1000, 6, 7]

(merge-sort (first-half my-list))

[3, 5, 100] ✓

(merge-sort (second-half my-list))

[6, 7, 1000] ✓

merge [3, 5, 6, 7, 100, 1000]

first1 [3, 5, 100] ✓

first2 [6, 7, 1000] ✓

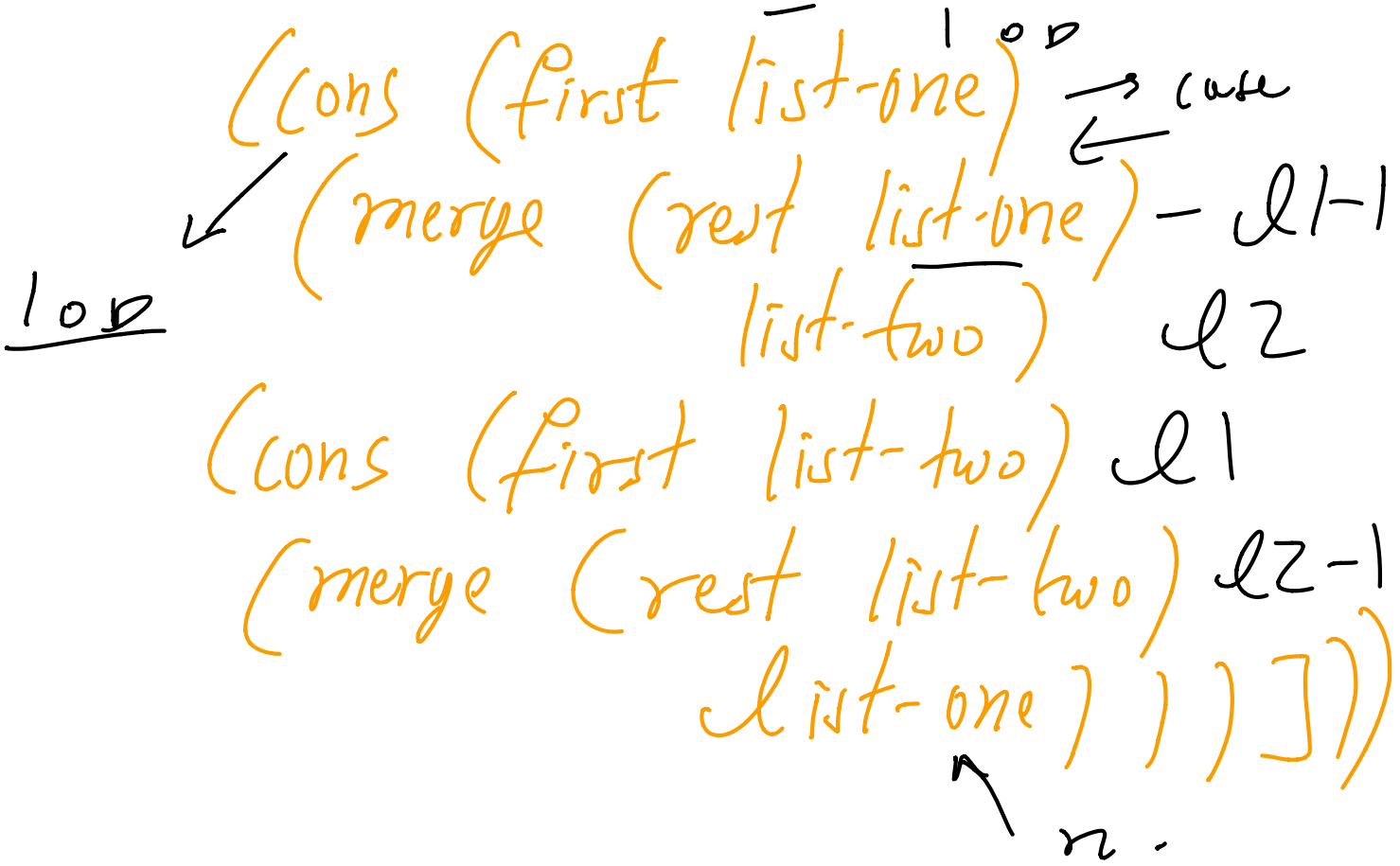
③ [5, 100] [6, 7, 1000]

done by recursion.

3 [5, 6, 7, 100, 1000]

(cons 3 , 5, 6, 7, 100, 1000)

```
(: merge : (Listof Integer)
           (Listof Integer)
           → (Listof Integer))
(define (merge list-one list-two)
  (cond ← 1 op   ↖ l1   ↘ l2
[(and (empty? list-one (emty? listtwo)
      empty] ← 3 ops
[(empty? list-one) list-two] 2 ops
[(empty? list-two) list-one] 2 ops
[else (if (< (first list-one) → 4 ops
              (first list-two))
```



Time-complexity of merge

Let $T(n)$ denote the time complexity of merge.

What is n?

What is our input size.

input size is size of list 1 + size of list 2

$n = l1 + l2$ → $n-1$

$T(n) = T(l1 + l2 - 1)$

or $T(l2 + l1 - 1)$

$+ C$

$T(n) = T(n-1) + C.$

$T(n) = O(n)$

Time-complexity of merge sort.

Let $T(n)$ be the time-complexity of merge-sort.

$$T(n) = O(n) + C + 2T\left(\frac{n}{2}\right)$$

$O(n)$ ↑ merge

Recurrence

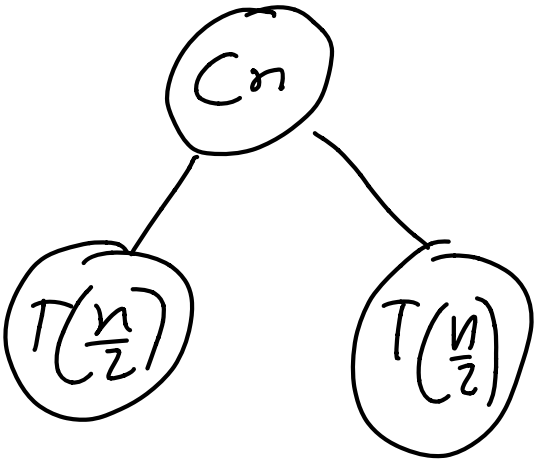
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

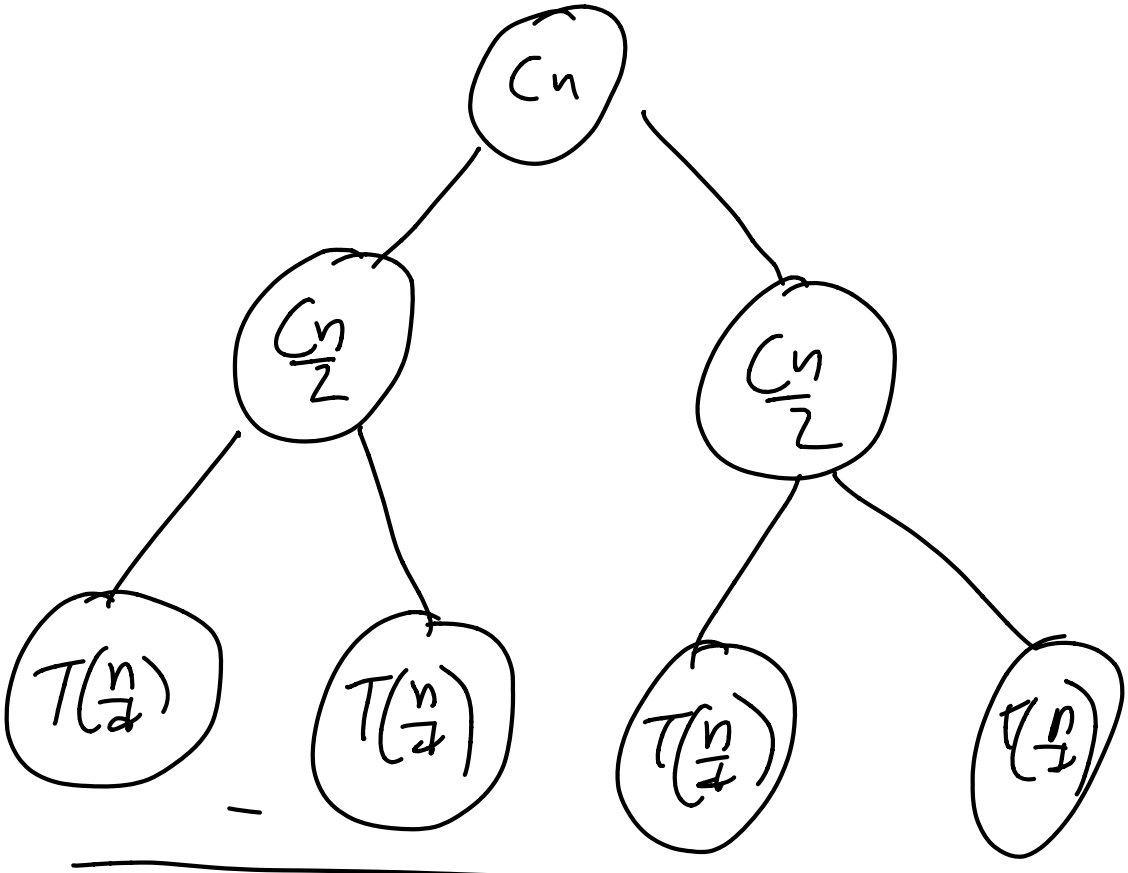
$+C\frac{n}{2}$ ↑ First half $+C\frac{n}{2}$ ↑ Second half $O(n)$ ↑ merge

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

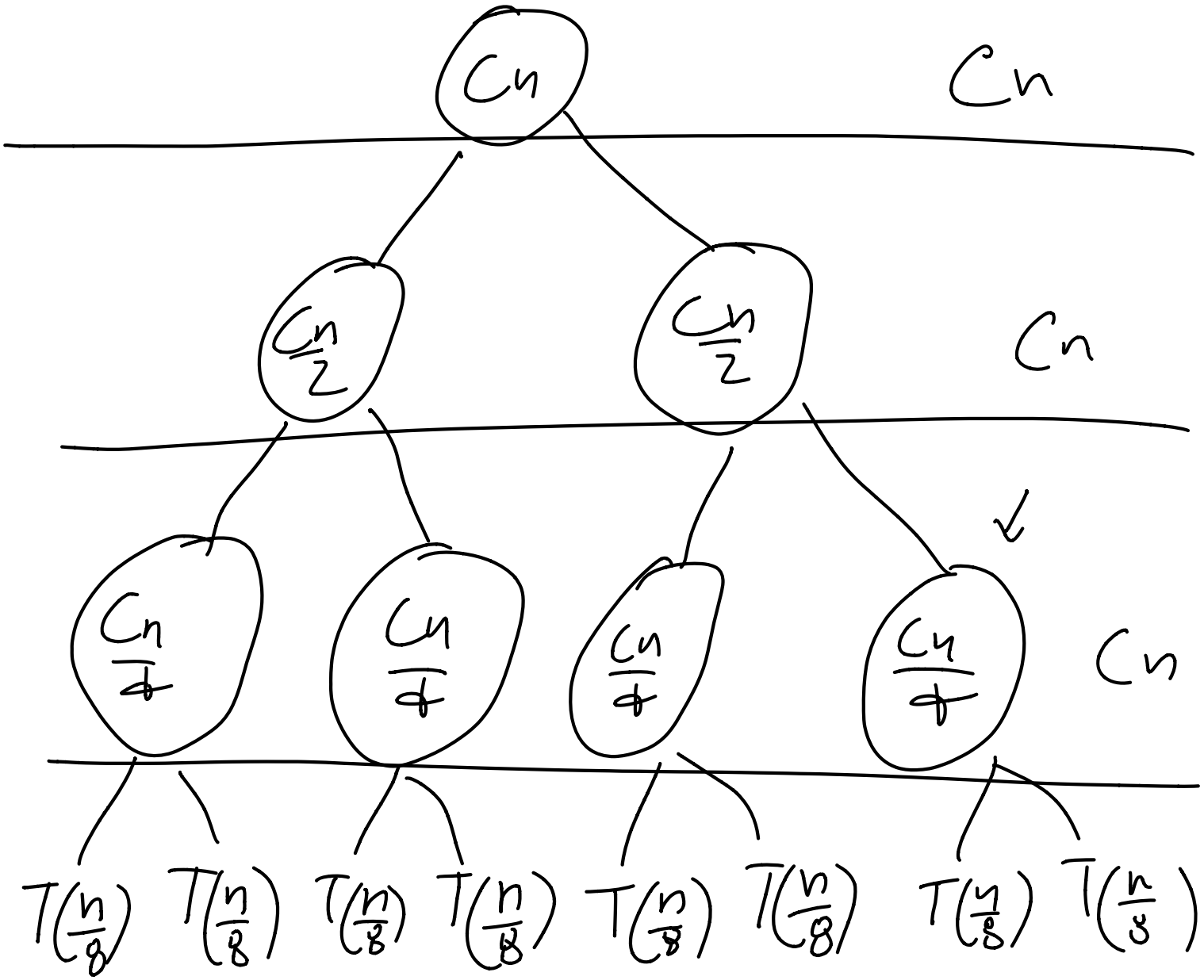
$$T(1) = \text{Constant}$$

Recursion Tree to solve the recurrence





$$T\left(\frac{n}{2}\right) = \frac{Cn}{2} + 2T\left(\frac{n}{4}\right)$$

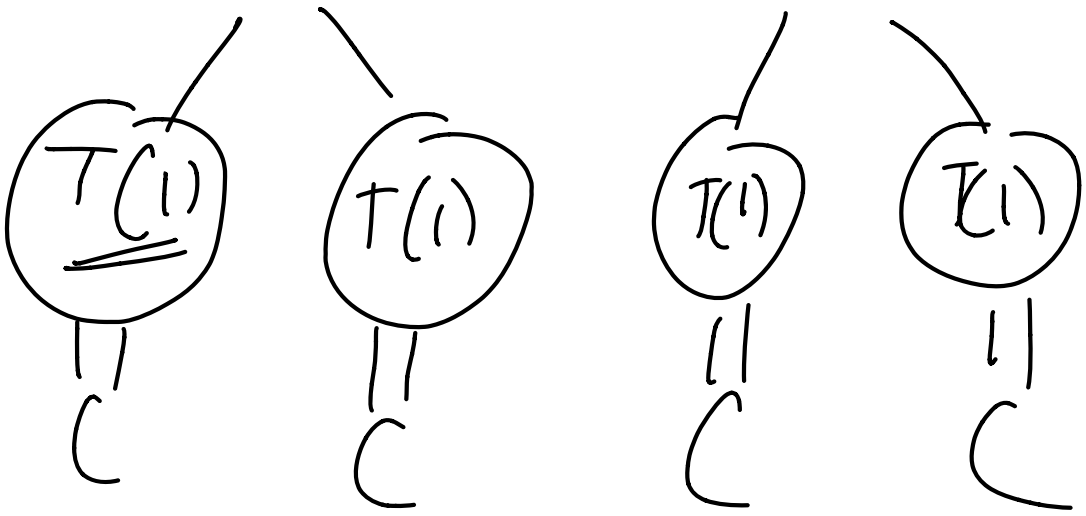


If we add everything in each level we Cn

$\left(\frac{Cn}{2^K}\right) \times 2^K = Cn$., K is the level

$T(n) = Cn \times$ (Height of this tree)

Every time we are dividing $\frac{n}{2}$.



For what value of k.

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

$$T(n) = n \log_2 n.$$

$$= O(n \log_2 n)$$

Remark

Recurrence Relation.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$+ \frac{cn}{2} + \frac{cn}{2}$$

$$= 2T\left(\frac{n}{2}\right) + O(n)$$

$$+ Cn$$

$$= 2T\left(\frac{n}{2}\right) + O(n) .$$

$$O(n) + Cn = \underline{O(n)}$$

$$O(n) + Cn = Kn + Cn$$

$$= (K+C)n$$

$$= \underline{O(n)}$$

$$O(n) + \frac{n}{2} = O(n). \checkmark$$

$Cn + \frac{n}{2}$

Treat $O(n) = Cn$

$O(n) + \frac{n}{2} = Cn + \frac{n}{2}$

$= \left(C + \frac{1}{2}\right) n = \underline{O(n)}$

↑ constant

Treat $O(n) = \underline{Cn}$.

Remark: Racket has a sort
Racket is using quicksort

It works very well in practice

Worst-case time-complexity
- $O(n^2)$ - quicksort

Average-time-complexity
- $O(n \log n)$