# Recursive Sorts

Monday, January 27, 2025 12:21 PM

**Last Time:**
- Finish Hanoi
- Peaks
- HW1

**Today:**
- Sorts
- Recurrences
- **HW1 due Today!**
- **HW2 will be out, due Friday**
  - **(shorter)**

---

## Sort

1 12 9 8 2 8 3 ← n ← L

1 2 3 8 8 9 12 ← output → make a new list or modify L.

---

**Brute-force**: Generate all $n!$ orderings.

↳ very inefficient!!

**naive sorts**

- bubble sort
- selection sort
- insertion sort.

```
selection(L[1...n]):
  for i from 1 to n:
    x ← index of the smallest item in L[i...n]
    swap L[i] and L[x].
```

$O(n^2)$

**recursive solution**

**Alg**

① Finds and moves the 1 smallest item. → $\Theta(n)$

② Recursively runs itself on the remaining items (one less item).

(one less item). ↳ $T(n-1)$

$$T(n) = T(n-1) + \Theta(n) \longrightarrow \Theta(n^2)$$

## Better Sorting Algorithms

- Quicksort ✓
- Mergesort

↓ recursion based sorts

- Heapsort, ...

↓ reduction to a data structure problem.

## Merge sort

subroutine

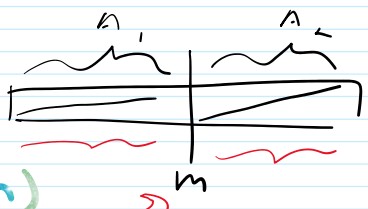
**Merge**

- Given two sorted lists $A_1$ and $A_2$, we produce a list $B$ which has all the items from $A_1$ and $A_2$, and $B$ is sorted.

**merge sort**

$T(n)$

```
m_sort (A[1...n])
    if n ≤ 1:
        the input is already sorted.
    otherwise:
        m ← ⌊n/2⌋
        m_sort (A[1...m])        T(n/2)
        m_sort (A[m+1, n])       T(n/2)
        merge (A[1...n], m)      Θ(n)
```

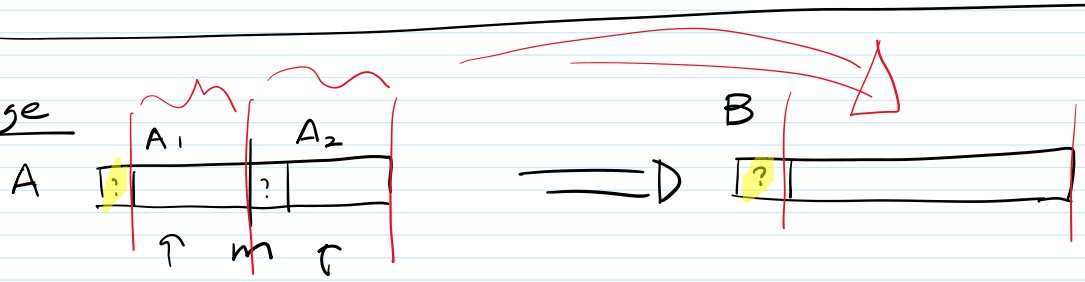


merge ( A[1...n], m) → $\Theta(n)$

Proof → Induction!!

---

**Merge**




**Alg**

**Obs** What item can be B[1] ?

Either $A_1[1]$ or $A_2[1]$

$A_1[1]$ ↓ $A[1]$ ; $A_2[1]$ ↓ $A[m+1]$

① Figure out the first item to put in B → $\Theta(1)$

② WLOG A[1] is put into B[1]

Run the algorithm recursively on

A[2...m] & A[m+1...n] → T(n-1)

to

put into B[2...n]

```
Merge (A[1...n], m):
  i ← 1, j ← m+1
  for k from 1 to n:
      if j > n:
          B[k] ← A[i]; i ← i+1
      else if i > m:
          B[k] ← A[j]; j ← j+1
      else if A[i] ≤ A[j]:
```

$\Theta(n)$

$T(n) = T(n-1) + \Theta(1)$

```
else if A[i] ≤ A[j]:
    B[k] ← A[i]; i ← i+1
else
    B[k] ← A[j]; j ← j+1
Copy B back into A.
```

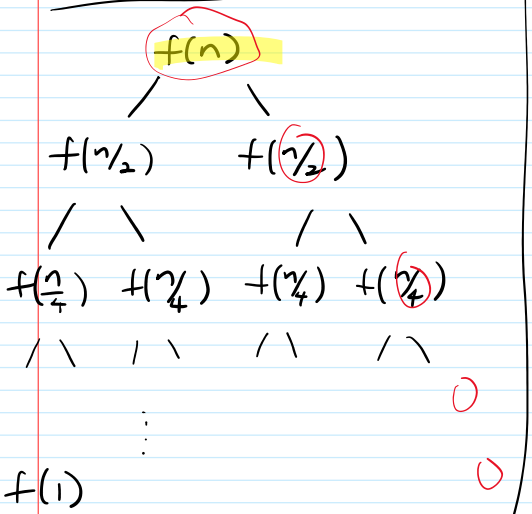
## Efficiency of Sort

$$T(n) = 2T(n/2) + \Theta(n) \implies \Theta(n \log n)$$

## Solving Recurrences (tree method)

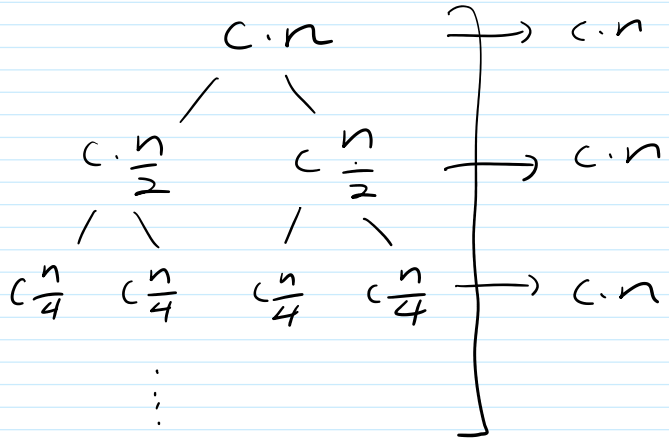
$T(n) = 2\,T(n/2) + \Theta(n)$ → $c \cdot n$

Call tree




putting in work → put in all non-recursive work.

recursion tree




Running time → sum of all work in tree.

- Every level is $c \cdot n$
- How many levels? $\Rightarrow \log n$

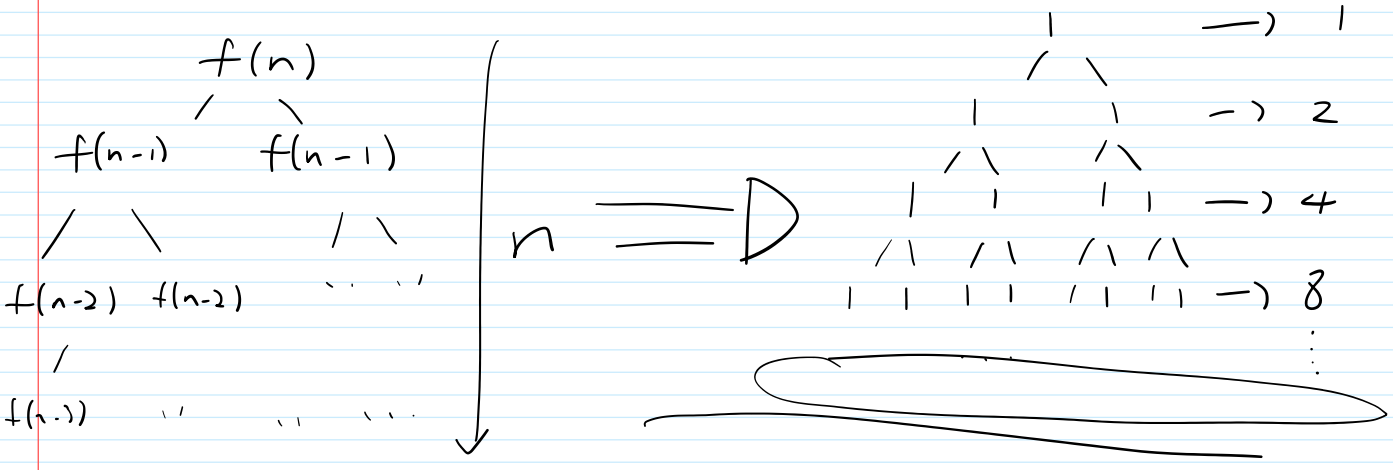
$c \cdot n \cdot \log n \rightarrow c \cdot n \log n$

$c \cdot n \cdot \log n \rightarrow c \cdot n \log n$

$T(n)$ is $\Theta(n \log n)$

## Hanoi

$T(n) = 2T(n-1) + 1$

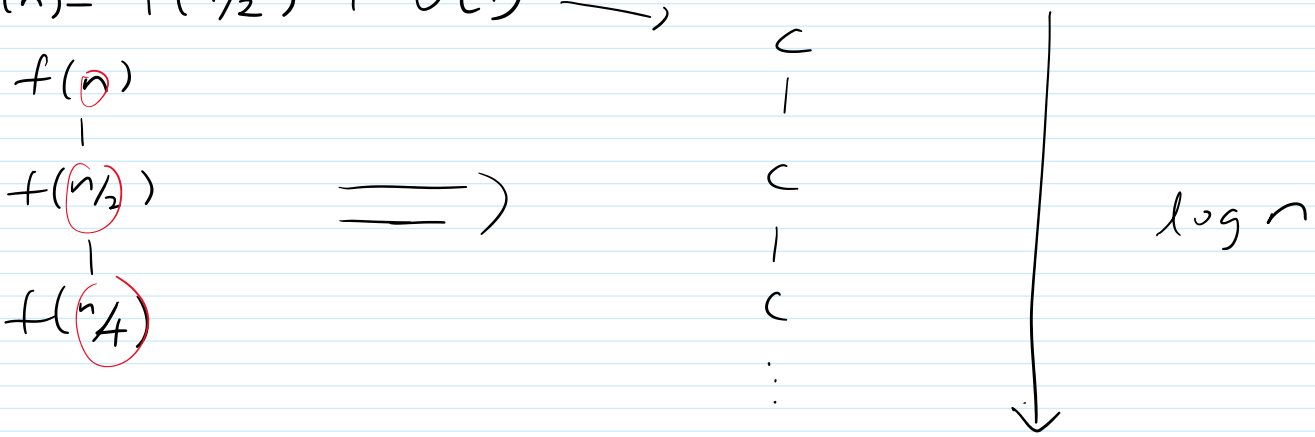



Bottom level is $2^l \Rightarrow 2^n$

$\Theta(2^n)$

## Peak

$T(n) = T(n/2) + \Theta(1) \rightarrow$




$T(n)$ is $c \log n \rightarrow \Theta(\log n)$

$T(n) = T(n-1) + \Theta(n)$

$f(n)$
|
$f(n-1)$
|
$f(n-2)$
⋮

⇒

$n$
↓
$n-1$
↓
$n-2$
↓
$\Theta(n^2)$

$T(n) = T(n-1) + T(n-2)$

Recursion trees

3 easy cases

① Every level has equal work.
↳ (# of levels) · (work per level)

② Every level increase geometrically.
↳ $T(n)$ is just the bottom level.

③ Every level decrease geometrically.

$c \cdot n$
$c \cdot n/2$
$c \cdot n/4$
⋮
$1$

⇒ only the first level matters.