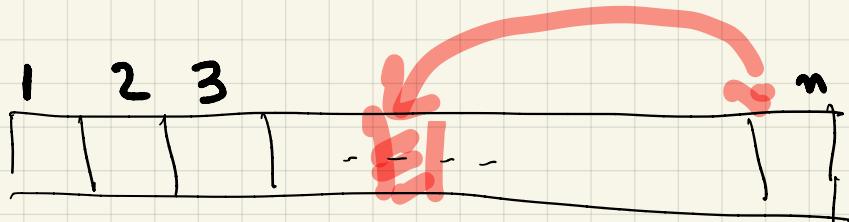


# DIVIDE - AND - CONQUER PARADIGM

---

1. DIVIDE PROBLEM INTO SMALLER SUBPROBLEMS
2. SOLVE THE SUBPROBLEMS RECURSIVELY
3. COMBINE THE SOLUTIONS OF SUBPROBLEMS INTO A SOL. OF THE ORIGINAL PROBLEMS

# SELECTION SORT



array of  $n$  integers  
TASK: Sort them

$$v = \text{find\_max}(a[1..n])$$

$$\text{swap } v \leftrightarrow a[n]$$

$$\text{select\_sort}(a[1..n-1])$$

SELECT-SORT (  $a[1..r]$  )

if ( $r < 1$ ) exit

else { max =  $a[1]$ ; ind-max = 1; ;

for ( $i = 1$  to  $r$ ) if ( $a[i] > \text{max}$ ) {

$\text{max} = a[i]$ ;  $\text{ind\_max} = i$ ; }

$a[\text{ind\_max}] \leftrightarrow a[r]$

SELECT-SORT(  $a[1..r-1]$  )

{

recursive implementation

## RECURRENCE:

$$T(n) = c \cdot n + T(n-1)$$

time for recursive call  
or input of size  $n-1$

time for left-to-right pass

# SOLVE RECURRENCE BY ITERATION

$$T(n) = C \cdot n + T(n-1)$$

$$T(n-1) = C \cdot (n-1) + T(n-2)$$

$$T(n-2) = C \cdot (n-2) + T(n-3)$$

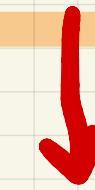
~  
o  
:

$$T(1) = C \cdot 1 + T(0)$$

$$\overbrace{T(n) + T(n-1) + \dots + T(1)}^{\text{[Orange Bar]}} = C \cdot (1+2+\dots+n) +$$
$$+ T(n-1) + \dots + T(1) + T(0)$$
$$\overbrace{\quad\quad\quad}^{\text{[Orange Bar]}}$$

$$T(n) = C \cdot \frac{n(n+1)}{2}$$

$$T(n) = \Theta(n^2)$$



SUBSTITUTION  
METHOD

# DIVIDE - AND - CONQUER PARADIGM

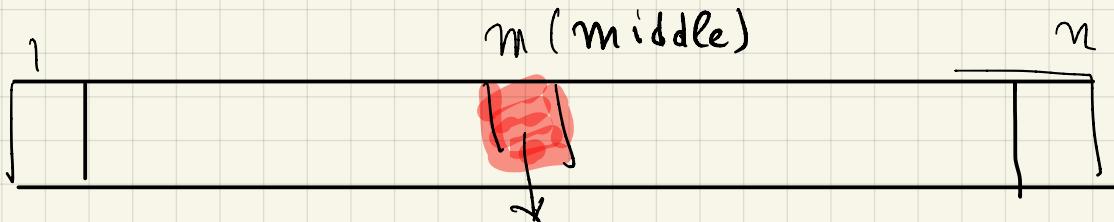
---

1. DIVIDE PROBLEM INTO SMALLER SUBPROBLEMS
2. SOLVE THE SUBPROBLEMS RECURSIVELY
3. COMBINE THE SOLUTIONS OF SUBPROBLEMS INTO A SOL. OF THE ORIGINAL PROBLEMS

# BINARY SEARCH

INPUT : Sorted array  $a[1..n]$  (sorted  $\nearrow$ ) and integer  $x$

GOAL : Determine if  $x$  is in the array and where it is.



Compare  $x$  with  $m$  and continue recursively  
to the left, or to the right.

Bin Search ( $a[l..r]$ ,  $x$ ) {

if ( $l > r$  and  $a[l] == x$ )

{ print " $x$  at position  $l$ "; return; }

if ( $l > r$  and  $a[l] != x$ )

{ print " $x$  not in the array"; return; }

$$m = l + \frac{r-l}{2}.$$

if ( $x \leq a[m]$ ) {  $r = m$ ; Bin Search ( $a[l..r]$ ,  $x$ ) }

else {  $l = m+1$ ; Bin Search ( $a[l..r]$ ,  $x$ ) }

}

## RECURRENCE

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

## SOLVE RECURRENCE

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

$$T(n/2) = T(n/4) + c$$

$$T(n/4) = T(n/8) + c$$

⋮  
⋮  
⋮

$$T\left(\frac{n}{2^{k-1}}\right) = T\left(\frac{n}{2^k}\right) + c$$

$\underbrace{= 2}_{=1}$

ADD + CANCEL

↓↓

$$T(n) = \underbrace{c + c + \dots + c}_{k \text{ times}} + T(1)$$

$$= c \cdot k + T(1)$$

$$= c \cdot \log n + T(1) = \Theta(\log n)$$

Assume  $n = 2^k$ .

$k = \log n$

What if  $n$  is not a power of 2 ?

Then for some  $k$ :  $2^{k-1} < n < 2^k \Leftrightarrow k-1 < \log n < k$

So  $k = \lceil \log n \rceil \approx \log n$



$$\underline{T(2^{k-1})} \leq \underline{T(n)} \leq \underline{T(2^k)}$$

$$C \cdot (k-1) + T(1) \leq T(n) \leq C \cdot k + T(1)$$

So:  $T(n) = \Theta(\log n)$ .

---

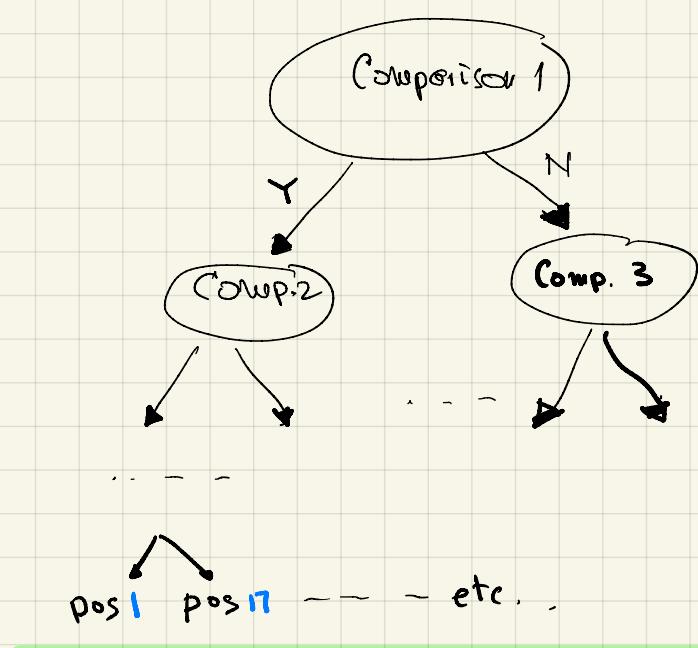
WE CAN ALWAYS ASSUME  $n$  has the "nice" form we need.

# LOWER BOUND FOR THE SEARCH PROBLEM

Q: Can we find  $x$  in array  $a[1..n]$  with less than  $\log n$  comparisons?

A: NO!

PROOF: Suppose there is an algorithm that finds  $x$  with  $(\log n - 1)$  comparisons ('Y' or 'N' answer) (for all  $x$ ).



Then each  $x \in \{1, \dots, n\}$  can be described by

a sequence of at most  $(\log n - 1)$  'Y's and 'N's.

$$I = 'Y Y \dots Y'$$

$$I' = 'Y N \dots N'$$

etc.

$I \rightarrow Y Y \dots \text{---} \quad (\text{the path that indicates where is } x)$

.

;

:

$$\underbrace{n}_{n \text{ elements}} \rightarrow \underbrace{N Y \dots \dots}_{2^{\log n - 1}}$$

$$\leq 2^{\log n - 1} = \frac{n}{2} \text{ labels. IMPOSSIBLE!}$$

# MODULAR EXPONENTIATION

$2^n \pmod{m}$ . Suppose  $m$  is fixed (const.)  
we want an efficient alg.  
as a function of  $n$ .

NAIVE ALG:  $\underbrace{2 \cdot 2 \cdot \dots \cdot 2}_{n \text{ times}}$

Suppose  $n$  is 200 bits long

$$n \geq 100 \dots 0 = 2^{199}$$

200 bits

$$\text{one year} \approx 32 \cdot 10^6 \text{ sec} \approx 2^5 \cdot 2^{20} \text{ sec.}$$

- Suppose we can do  $10^6 \approx 2^{20}$  multiplications per second

- TIME:  $\frac{2^{199}}{2^{20} \cdot 2^{20}} = 2^{154} \text{ years} \gg \text{Age of Universe}$

DIVIDE-AND-CONQUER

$$2^{20} = 2^{10} \times 2^{10}$$

$$2^{21} = 2 \times 2^{10} \times 2^{10}$$

$$2^n = \begin{cases} 2^{n/2} \cdot 2^{n/2}, & \text{if } n \text{ even} \\ 2^{n/2} \cdot 2^{n/2} \cdot 2, & \text{if } n \text{ odd} \end{cases}$$

$T(n) = \text{no. of multiplications}$

for input  $n$

mod-exp( $n$ ) {

if ( $n == 0$ ) return 1;

if ( $n == 1$ ) return 2(mod m);

else {

$t = \text{mod-exp}(n/2)$

$T(\frac{n}{2})$

if ( $n$  even) return  $t \times t \pmod{m}$

1 multiplication

else return  $t \times t \times 2 \pmod{m}$ .

—————

—————

2 multiplications

## RECURRENCE

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

SOLUTION :  $T(n) = \Theta(\log n)$

↓  
LINEAR TIME !

not logarithmic

because input size =  $\lceil \log(n+1) \rceil \approx \log n$

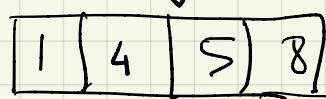
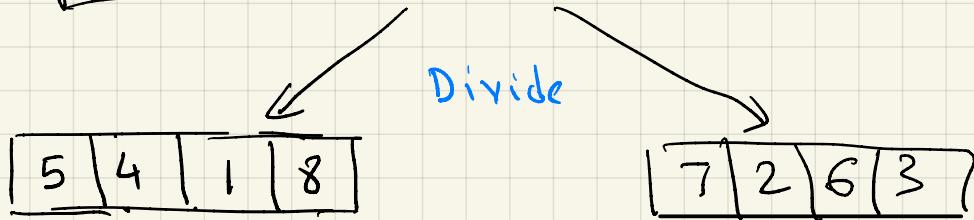
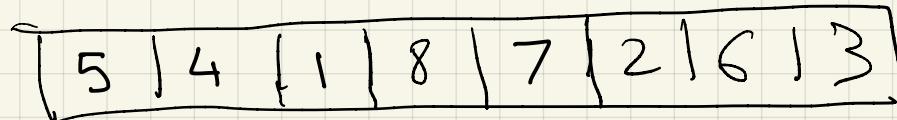
# MERGE SORT



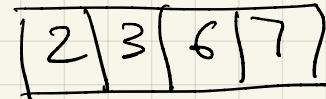
- (1) SORT LEFT HALF
- (2) SORT RIGHT HALF



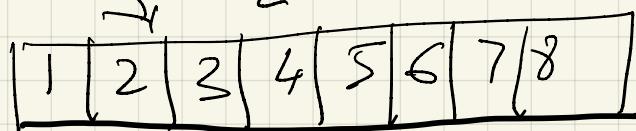
- (3) MERGE THE TWO HALVES



Sort recursively



Merge



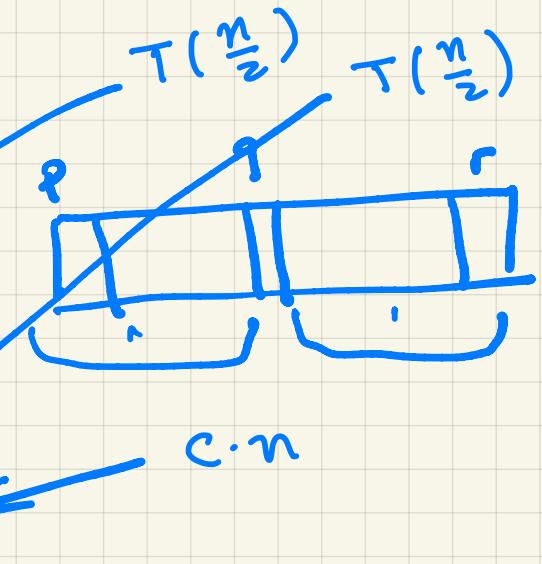
$\text{mergeSort}(A[p..r])$

$$q = p + \frac{r-p}{2};$$

$\text{mergeSort}(A[p..q])$ ;

$\text{mergeSort}(A[q+1..r])$ ;

$\text{merge}(A[3, p, q, r])$ ;



MERGE( $A, p, q, r$ )

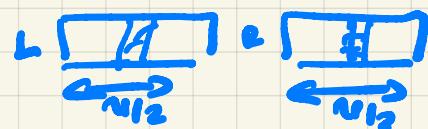
```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$            ← copy left half in L
5       $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$            ← copy right half in R
7       $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
```

Traverse L and R simultaneously  
write the smaller elem back in A

RUNTIME for merge :  $C \cdot n$  (or  $O(n)$ )

Why: Copy A into L, R:  $O(n)$       ↳ TOTAL:  $O(n)$ .

Write back in A :  $O(n)$



# RECURRENCE

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

## SOLVING THE RECURRENCE

Assume  $n = 2^k$ ; so  $k = \log n$ .

$$T(n) = 2 \cdot T(n/2) + C \cdot n$$

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

$$2^2 T(n/2^2) = 2^2 (2T(n/2^3) + C \cdot \frac{n}{2^2}) + 2Cn = 2^3 \cdot T\left(\frac{n}{2^3}\right) + 3Cn$$

⋮  
⋮  
⋮

$$2^{k-1} \cdot T\left(\frac{n}{2^{k-1}}\right) = 2^{k-1} \left( 2 \cdot T\left(\frac{n}{2^k}\right) + C \cdot \frac{n}{2^{k-1}} \right) + (k-1) \cdot C \cdot n =$$

$$= \underbrace{2^k}_{=n} \cdot T\left(\frac{n}{2^k}\right) + C \cdot \underbrace{k \cdot n}_{\log n}$$

$$= n \cdot T(1) + C \cdot n \cdot \log n$$

$$= \Theta(n \cdot \log n)$$

# RECURSION TREE for MergeSort

level 0

$n$  entire input:  $n$  elements

level 1

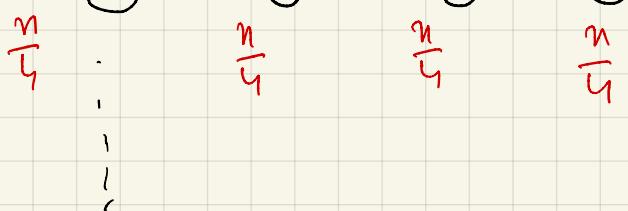
merge:  $Cn^2$  operations



each node:  $\frac{n}{2}$  elements

level 2

merge:  $C \cdot \frac{n}{2} + C \cdot \frac{n}{2} = C \cdot n$



each node:  $\frac{n}{4}$  elements

level  $K$

○ ← single elements



At each level: amount of work is  $C \cdot n$

(to do merge+rec. calls)

no. of levels:  $\log n$

TOTAL WORK:  $K \cdot Cn = \log n \cdot Cn$

$$= C n \log n$$

# MASTER THEOREM

## RECURRENCE

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + f(n) ; \quad a \geq 1, b > 1;$$

a: no. of rec. calls

b: shrinkage factor

f(n): time for "combine step".

## SOLUTION

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & \text{if } \frac{n^{\log_b a}}{f(n)} = \Omega(n^\epsilon) \text{ (CASE 1)} \\ \Theta(f(n) \cdot \log n), & \text{if } \frac{n^{\log_b a}}{f(n)} = \Theta(1) \text{ (CASE 2)} \\ \Theta(f(n)), & \text{if } \frac{f(n)}{n^{\log_b a}} = \Omega(n^\epsilon) \text{ (CASE 3)} \end{cases}$$

So:  $n^{\log_b a}$  vs  $f(n)$

If TIE,  $T(n) = \Theta(f(n) \cdot \log n)$

If  $\frac{\text{WINNER}}{\text{LOSER}} = \text{poly}(n) \Rightarrow T(n) = \Theta(\text{WINNER})$

Else MASTER TH DOES NOT APPLY

## MASTER Th. → Examples

$$(1) \quad T(n) = 9T\left(\frac{n}{3}\right) + n$$

$$(2) \quad T(n) = T\left(\frac{2n}{3}\right) + 1$$

$$(3) \quad T(n) = 3T\left(\frac{n}{4}\right) + n \cdot \log n$$

$$(4) \quad T(n) = 2T\left(\frac{n}{2}\right) + n \cdot \log n$$

(1)

$$T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n$$

$$a=9, b=3, f(n)=n$$

$$n^{\log_3 9} \text{ vs. } n$$

$$\begin{matrix} n^2 \\ \curvearrowleft \\ \text{WINNER} \end{matrix} \text{ vs. } \begin{matrix} n \\ \curvearrowright \\ \text{LOSER} \end{matrix}$$

$$\frac{W}{L} = \frac{n^2}{n} = n \rightarrow \text{OK for Master Th.}$$

$$T(n) = \Theta(\text{WINNER}) = \Theta(n^2)$$

(2)

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

$\downarrow \frac{2n}{3} = \frac{n}{(3/2)}$

$$a = 1$$

$$b = 3/2$$

$$f(n) = 1$$

$$n^{\log_b a} \quad \text{vs.} \quad f(n)$$

$$n^{\log_{3/2} 1} = 0 \quad \text{vs.} \quad 1$$

$$1 \quad \text{vs.} \quad 1 \longrightarrow \text{TIE}$$

$$T(n) = \Theta(1 \cdot \log n) = \Theta(\log n)$$

(3)

$$T(n) = 3 T\left(\frac{n}{4}\right) + n \cdot \log n$$

$$a = 3$$

$$b = 4$$

$$f(n) = n \cdot \log n$$

$$n^{\log_4 3}$$

vs.

$$n \cdot \log n$$

LOSER

WINNER

$$\frac{\text{WINNER}}{\text{LOSER}} = \frac{n \cdot \log n}{n^{\log_4 3}} > n^{0.1} \cdot \log n$$

0.7924...

$$T(n) = \Theta(\text{WINNER}) = \Theta(n \cdot \log n)$$

(4)

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

$$\alpha = 2$$

$$b = 2$$

$$f(n) = n \cdot \log n$$

$$n^{\log_2 2} \text{ vs } n \cdot \log n$$

$n^1$   
WINNER

vs.

$n \cdot \log n$   
LOSER

$$\frac{\text{WINNER}}{\text{LOSER}} = \frac{n \cdot \log n}{n} = \log n \begin{cases} \text{not a} \\ \text{polynomial} \end{cases}$$

4

WE CANNOT USE MASTER TH.

## EXAMPLES of other recurrences

---

(1)  $T(n) = T(n-1) + \sqrt{n}$ ,  $T(0) = 1$

(2)  $T(n) = T(n-1) + \log n$ ,  $T(0) = 1$

---

(1) 
$$\begin{aligned} T(n) &= T(n-1) + \sqrt{n} \\ &= T(n-2) + \sqrt{n-1} + \sqrt{n} \end{aligned}$$

$$= T(n-3) + \sqrt{n-2} + \sqrt{n-1} + \sqrt{n}$$

⋮

$$= T(0) + \underbrace{\sqrt{1} + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n}}_S$$

estimate using integrals or  
the "cut-and-bound" technique

$$1 + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n} \leq n \cdot \sqrt{n}$$



$$1 + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n} \geq \frac{n}{2} \cdot \sqrt{\frac{n}{2}}$$

$$\text{So } T(n) = \Theta(n \cdot \sqrt{n})$$

# Estimation using integrals

$$S = \sqrt{1} + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n}$$

$$f(x) = \sqrt{x} ; \quad S = f(1) + f(2) + \dots + f(n)$$

$$\int_0^n f(x) dx \leq S \leq \int_1^{n+1} f(x) dx$$

$$S \geq \int_0^n x^{1/2} dx = \frac{x^{3/2}}{3/2} \Big|_0^n = \frac{n^{3/2}}{3/2} - 0$$

$$S \leq \int_1^{n+1} x^{1/2} dx = \frac{x^{3/2}}{3/2} \Big|_1^{n+1} = \frac{(n+1)^{3/2}}{3/2} - \frac{1}{3/2}$$

$$\Rightarrow S = \Theta(n^{3/2}).$$

(2)

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

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

$$= T(n-3) + \log(n-2) + \log(n-1) + \log n$$

'

'

'

'

$$= T(0) + \log(1) + \log(2) + \dots + \log n$$

S

We estimate S.

CUT-AND-BOUND METHOD

$$S \leq n \cdot \log n$$

$$S = O(n \cdot \log n)$$

$$S \geq \frac{n}{2} \cdot \log \frac{n}{2} = \frac{1}{2} \cdot n (\log n - 1)$$

$$= \frac{1}{2} n \cdot \log n - \frac{1}{2} n$$

$$\text{So } S = \Theta(n \cdot \log n)$$

## USING INTEGRALS

$$S = \log_2 1 + \log_2 2 + \dots + \log_2 n$$

$\approx 0$

$$f(x) = \log_2 x = \log_2 e \cdot \ln x$$

$$S = f(2) + \dots + f(n)$$

$$\begin{aligned} S &\geq \int_1^n \log_2 e \cdot \ln x \, dx \\ &= \log_2 e \cdot \int_1^n \ln x \, dx \end{aligned}$$

$$\begin{aligned} u = \ln x &\Rightarrow du = \frac{1}{x} dx \\ du = dx &\Rightarrow u = x \end{aligned}$$

$$\begin{aligned} &\log_2 e \cdot \left( x \ln x - \int 1 \, dx \right) \Big|_1^n \\ &= \log_2 e \left[ n \ln n - (n-1) \right] \\ &= \Omega(n \cdot \ln n) \end{aligned}$$

$$\begin{aligned} S &\leq \int_2^{n+1} \log_2 e \cdot \ln x \, dx \\ &= \log_2 e \cdot \left( x \ln x \Big|_2^{n+1} - \int_2^{n+1} 1 \, dx \right) \\ &\quad \text{((n+1) · ln(n+1) - 2 ln 2)} \end{aligned}$$

$$\begin{aligned} &\quad \text{((n+1) - 2)} \\ &= O(n \cdot \ln n) \end{aligned}$$

So:  $S = \Theta(n \cdot \log n)$