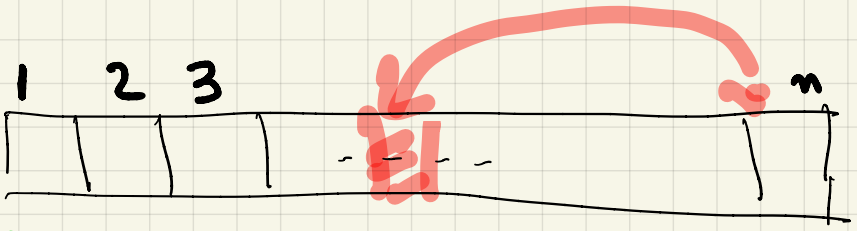
DIVIDE-AND-CONQUER PARADIGM

1. DIVIDE PROBLEM INTO SMALLER SUBPROBLEMS

2. SOLVE THE SUBPROBLEMS RECURSIVELY

3. COMBINE THE SOLUTIONS OF SUBPROLEMS INTO A SOL. OF THE ORIGINAL PROBLEMS

SELECTION SORT



array of n integers

TASK: Sort them

```
v = find_max(a[1..n])
swap v ⟷ a[n]
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] > max) {
                              max = a[i]; ind_max = i; }
        a[ind_max] ⟷ a[r]
        SELECT-SORT( a[1.. r-1])
 }
```

recursive implementation

RECURRENCE:

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

$T(n-1)$: time for recursive call on input of size $n-1$

$c \cdot n$: time for left-to-right pass

SOLVE RECURRENCE BY ITERATION

SUBSTITUTION METHOD

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

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

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

$$\vdots$$

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

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

$$+ T(n-1) + \cdots + T(1) + T(0)$$

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

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

DIVIDE-AND-CONQUER PARADIGM

1. DIVIDE PROBLEM INTO SMALLER SUBPROBLEMS

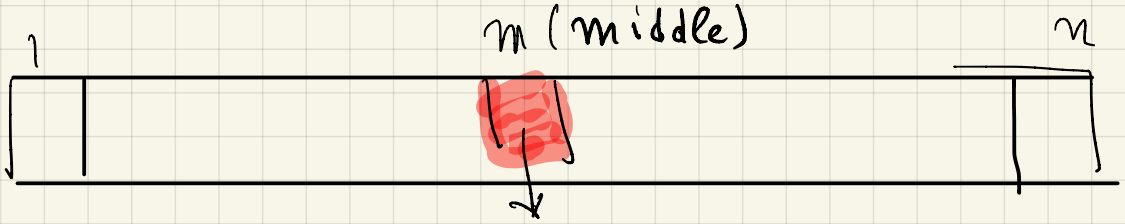
2. SOLVE THE SUBPROBLEMS RECURSIVELY

3. COMBINE THE SOLUTIONS OF SUBPROBLEMS INTO A SOL. OF THE ORIGINAL PROBLEMS

BINARY SEARCH

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

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



compare with x and continue recursively to the left, or to the right.

```
BinSearch ( 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 + (r-l)/2.
  if ( x ≤ a[m] ) { r = m; BinSearch (a[l..r],x) }
  else { l = m+1; BinSearch (a[l..r],x) }
}
```

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

SOLVE RECURRENCE

Assume $n = 2^k$,

$k = \log n$

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

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

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

$$\vdots$$

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

ADD + CANCEL

$$\Downarrow$$

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

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

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

What if n is not a power of 2 ?

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

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

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

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

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

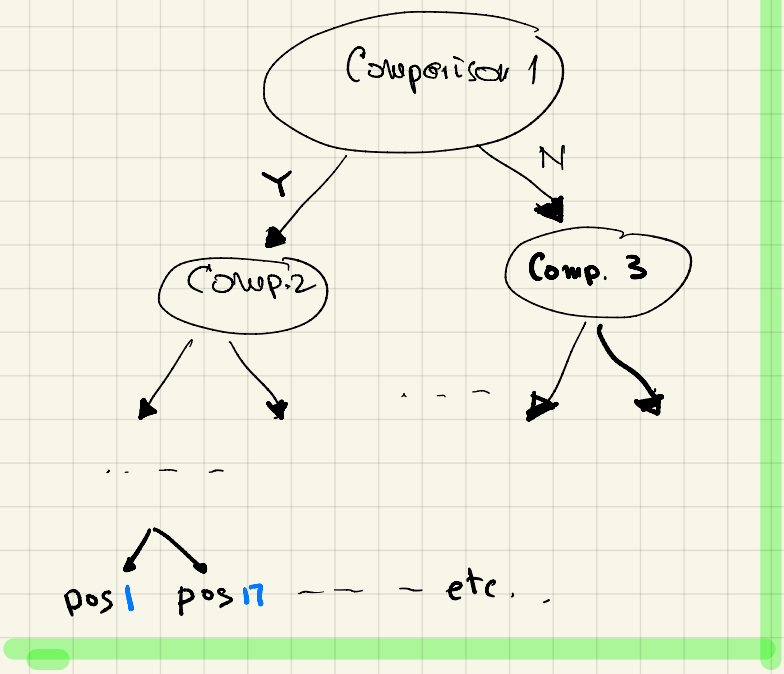
WE CAN ALWAYS ASSUME n has the "nice" form we need.

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 $x \in \{1, \ldots, n\}$ can be described by a sequence of at most $(\log n - 1)$ Y's and N's.

1 = 'YY ... Y'

17 = 'YY ... N'

etc.

1 ⟶ YY.. ...

⋮

n ⟶ NY.....

(the path that indicates where is x)

n elements $\leq 2^{\log n - 1} = \frac{n}{2}$ 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: $\underbrace{2 \cdot 2 \cdot \cdots \cdot 2}_{n \text{ times}}$

Suppose n is 200 bits long

$n \geq \underbrace{100\cdots0}_{200 \text{ bits}} = 2^{199}$

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

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

- TIME: $\frac{2^{199}}{2^{20} \cdot 2^{25}} = 2^{154}$ years >> Age of Universe

DIVIDE-AND-CONQUER

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

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

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

$T(n)$ = no. of multiplic. for input n

```
mod_exp(n) {
  if (n==0) return 1;
  if (n == 1) return 2 (mod m);
  else {
      t = mod_exp(n/2)
      if (n even ) return t*t (mod m )
      else return t*t*2 (mod m ).
```

$T(\frac{n}{2})$ ← 1 multiplication

2 multiplic. →

RECURRENCE

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

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

⇓

LINEAR TIME ! (not logarithmic)

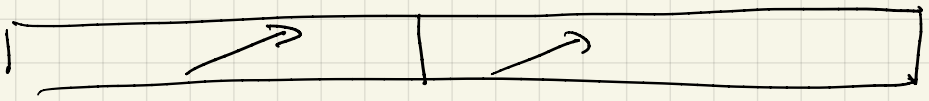
because input size $= \lceil \log(n+1) \rceil \simeq \log n$

MERGE SORT

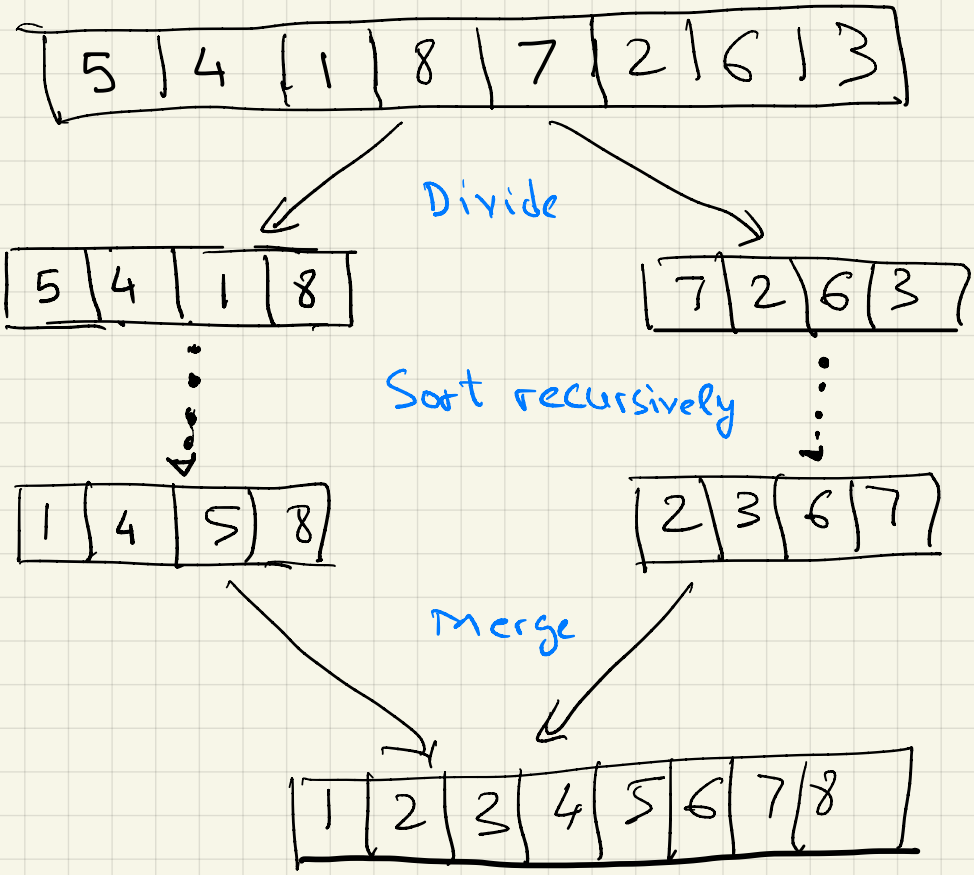
A:

(1). SORT LEFT HALF

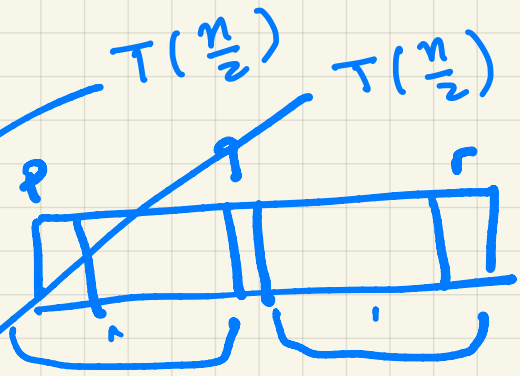
(2). SORT RIGHT HALF



(3) MERGE THE TWO HALVES



```
mergeSort( A [p..r])
    q = p + (r-p)/2;
    mergeSort( A [p..q]);
    mergeSort (A [q+1.. r]);
    merge  (A[], p, q, r);
```



```
MERGE(A, p, q, r)
1   n1 = q − p + 1
2   n2 = r − q
3   let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
4   for i = 1 to n1
5       L[i] = A[p + i − 1]
6   for j = 1 to n2
7       R[j] = A[q + j]
8   L[n1 + 1] = ∞
9   R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

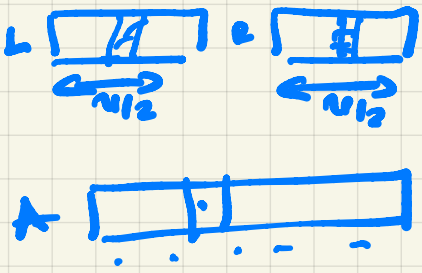
copy left Half in L

copy Right Half in R

traverse L and R simult
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 $k = \log n$.

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

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

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

$$\vdots$$

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

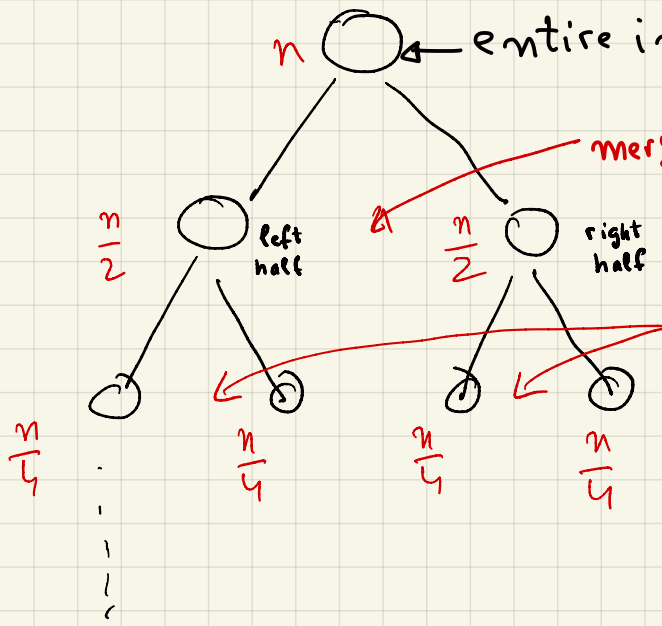
$$= 2^k \cdot T\left(\frac{n}{2^k}\right) + c \cdot k \cdot n$$

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

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

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

RECURSION TREE for MergeSort



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

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

a: no. of rec. calls

b: shrinkage factor

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

SOLUTION

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

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

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

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

Else MASTER TH DOES NOT APPLY

MASTER Th. → Examples

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

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

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

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

(1)

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

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

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

$$\underbrace{n^2}_{\text{WINNER}} \quad \text{vs.} \quad \underbrace{n}_{\text{LOSER}}$$

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

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

(2)

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

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

$a = 1$

$b = 3/2$

$f(n) = 1$

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

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

1 vs. 1 ⟶ TIE

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

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

$a = 3$

$b = 4$

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

$n^{\log_4 3}$ LOSER vs. $n \cdot \log n$ WINNER

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

$\log_4 3 \approx 0.7924\ldots$

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

(4)

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

$$a = 2$$
$$b = 2$$
$$f(n) = n \cdot \log n$$

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

$\underbrace{n^1}_{\text{WINNER}}$ vs. $\underbrace{n \cdot \log n}_{\text{LOSER}}$

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

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

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

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

$\vdots$

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

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

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

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

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

Estimation using integrals

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

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

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

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

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

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

(2)

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

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

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

$$\vdots$$

$$= T(0) + \underbrace{\log(1) + \log(2) + \ldots + \log n}_{S}$$

We estimate S.

CUT-AND-BOUND METHOD

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

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

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

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

USING INTEGRALS

$$S = \underbrace{\log 1}_{=0} + \log 2 + \cdots + \log n$$

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

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

$$S \geq \int_1^n \log_2 e \cdot \ln x \, dx$$

$$= \log_2 e \cdot \int_1^n \ln x \, dx$$

$u = \ln x \Rightarrow du = \frac{1}{x}$

$dv = dx \Rightarrow v = x$

$$\log_2 e \cdot \left(x \ln x - \int 1 \, dx \right)_1^n$$

$$= \log_2 e \left[n \ln n - (n-1) \right]$$

$$= \Omega(n \cdot \ln n)$$

$$S \leq \int_2^{n+1} \log_2 e \cdot \ln x \, dx$$

$$= \log_2 e \cdot \left(x \ln x \Big|_2^{n+1} - \int_2^{n+1} 1 \, dx \right)$$

$(n+1) \cdot \ln(n+1) - 2 \ln 2$

$(n+1) - 2$

$$= O(n \cdot \ln n)$$

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