CSE221 ALGORITHMS

Topic: Recursive Time Complexity, Merge Sort

Prepared by:
Saad Bin Sohan
BRAC University

Email: sohan.academics@gmail.com

GitHub: https://github.com/saad-bin-sohan

Date! 11/10/2023

Topic: Recursive Time Complexity

Recursive Time Complexity

- substitution method
- Recursive Tree Method
- Master's Theorem

T(n)

```
def recursive_function(n):
    if n == 1:
        return
    elif n > 1:
        for i in range(n):
            print(i)
        recursive_function(n-1)
```

① ..... if n == 1:

n ..... for i in range(n):

T(n-1)

we know the time complexity for the "for loop" in the above code is n.

and the Time complexity for the

recursive function call, let that function call be named = $T(n-1)$
and $T(n)$ = first time's function call

Recursive equation,

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

$T(1) = 1$

$T(n) = O(n^2)$

in substitution method,

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

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

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

⋮

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

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

$T(1) = 1$

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

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

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

$= \frac{n^2}{2} + \frac{n}{2}$

$\Rightarrow \frac{n^2}{2}$

$\Rightarrow n^2$

So its $\bigcirc(n^2)$

Recursive Tree Method:

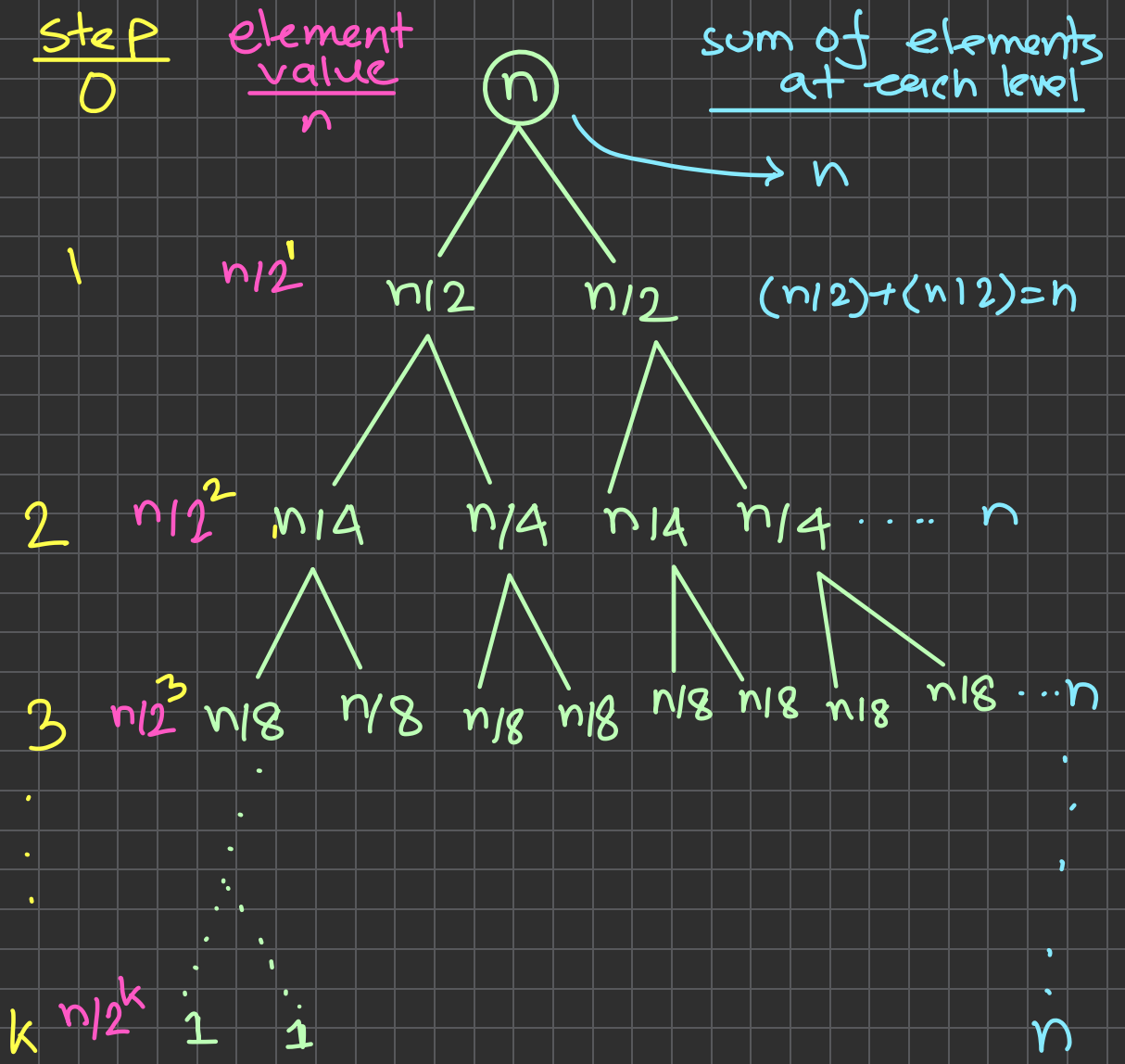
$$T(n) = 2T(n/2) + n, \quad T(1) = 1$$

soln:

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

নিচের tree থেকে we learn three things

1. root (initially function টা কার জন্য call হচ্ছে)
2. number of branches
3. who will be the branch element in each level


step
element value
sum of elements at each level
0
n
n
1
n/2^1
n/2
n/2
(n/2)+(n/2)=n
2
n/2^2
n/4
n/4
n/4
n/4
n
3
n/2^3
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n
k
n/2^k
1
1
n

so finally, $n/2^k = 1$

$$n = 2^k$$

$$\log_2 n = \log_2 2^k$$

$$\Rightarrow \log_2 n = k$$

maximum number of steps, $\boxed{k = \log_2 n}$

$T(n)$ = per step element * (step+1)

$$= n(\log_2 n + 1)$$

$$= n \log_2 n + n$$

$$\Rightarrow O(n \log_2 n)$$

Master's Theorem

$$T(n) = a\,T(n/b) + c * n^k$$

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } b^k < a \\ O(n^k \log n) & \text{if } b^k = a \\ O(n^k) & \text{if } b^k > a \end{cases}$$

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

soln: $a = 2,\ b = 2,\ c = 1,\ k = 1$

$b^k = 2^1 = 2,\ a = 2$

here $\boxed{b^k = a}$ → so,

$T(n) = O(n^k \log n)$

$= O(n \log n)$

Merge Sort

☆☆☆ Merge sort is the only Sorting method which is

Out of place algorithm.

which means this method sorts the elements somewhere outside of the given array.

All the other sorting methods

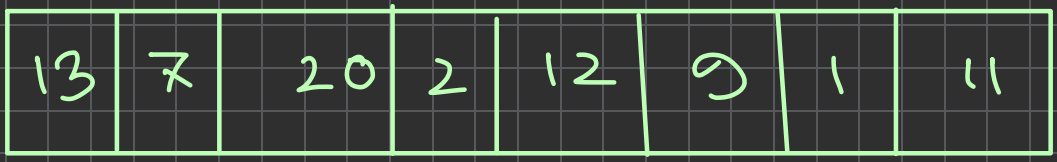
sort the elements in the given array.

Merge sort uses Divide and Conquer Method

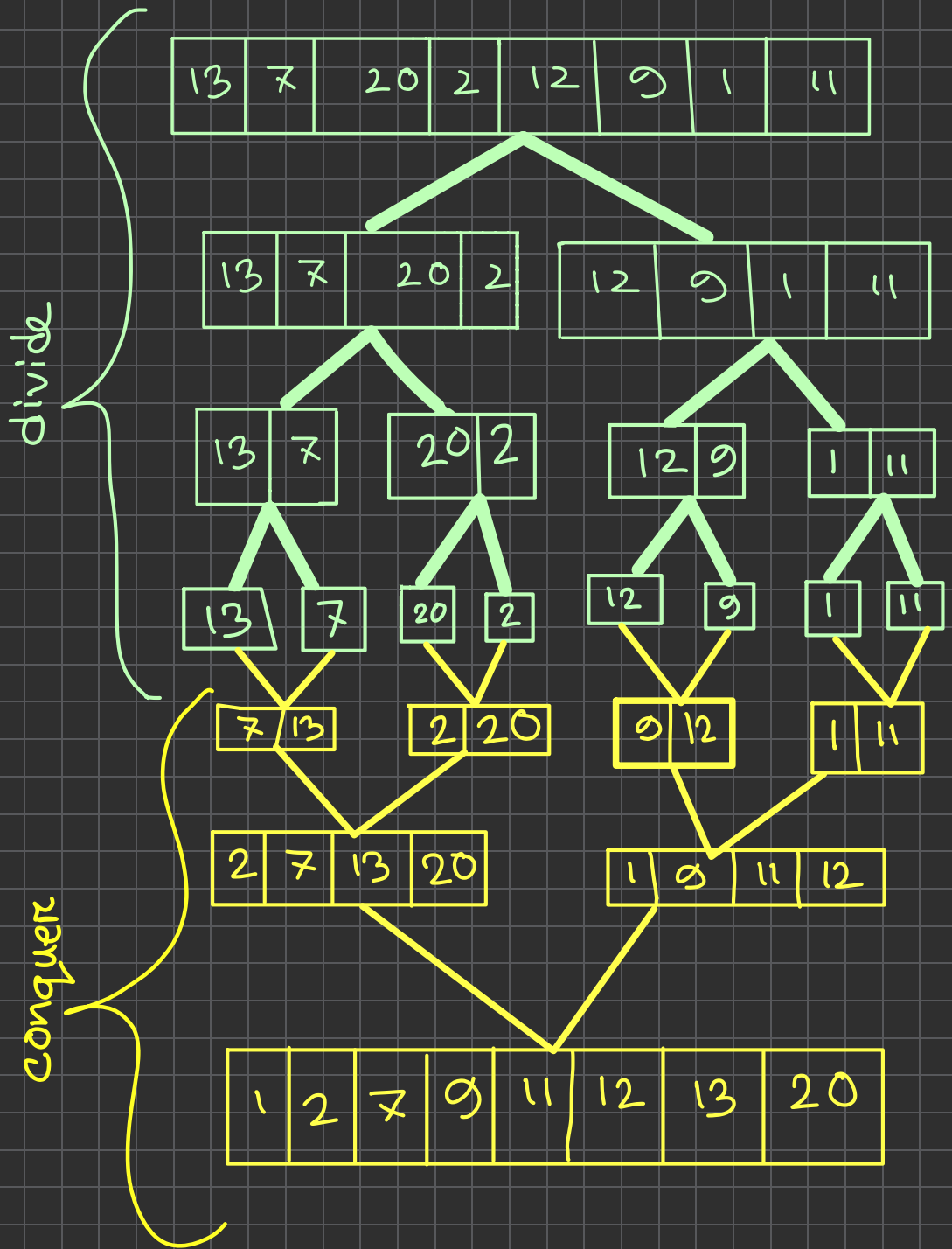
Divide = creating sub-arrays (here no sorting is done)

Conquer = where the elements are merged (while being sorted)

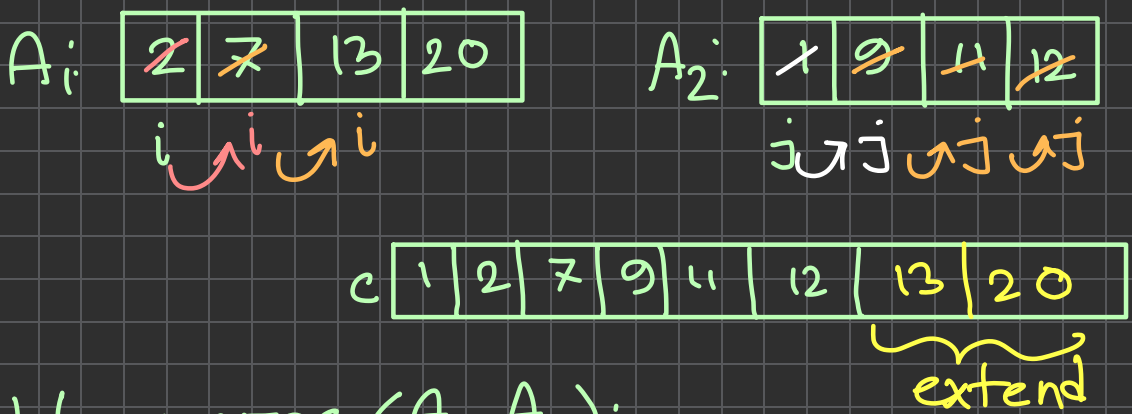
Que sort the given array using merge sort



Soln:


13 7 20 2 12 9 1 11
13 7 20 2
12 9 1 11
13 7
20 2
12 9
1 11
13
7
20
2
12
9
1
11
7 13
2 20
9 12
1 11
2 7 13 20
1 9 11 12
1 2 7 9 11 12 13 20
divide
conquer

in the sorting part:




```
def merge(A1, A2):
    C.len = A1.len + A2.len
    loop → (? condition)
        if (A1.i < A2.j):
            C.k = A1.i
            i += 1
            k += 1

        else:
            C.k = A2.j
            j += 1
            k += 1
```