

selection sort:

1	4	12	3	6	5
i		minIndex	j		

arr[i, n) 未排序
arr[0, i) 已排序

循环不变量

从 arr[i, n) 中 找最小值的 index

遍历一遍，确定 minIndex 的值。

遍历后，swap(arr, i, minIndex)

双重循环：

for (int i=0, i<n

for j=i, j<n.

if arr[j] < arr[minIndex]

swap(, ,) ~~minIndex = j~~

swap

Tip: 泛型 (generics) <E>

constraint: <E extends Comparable<E>>

必须是可比较的泛型

XXXXXX

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

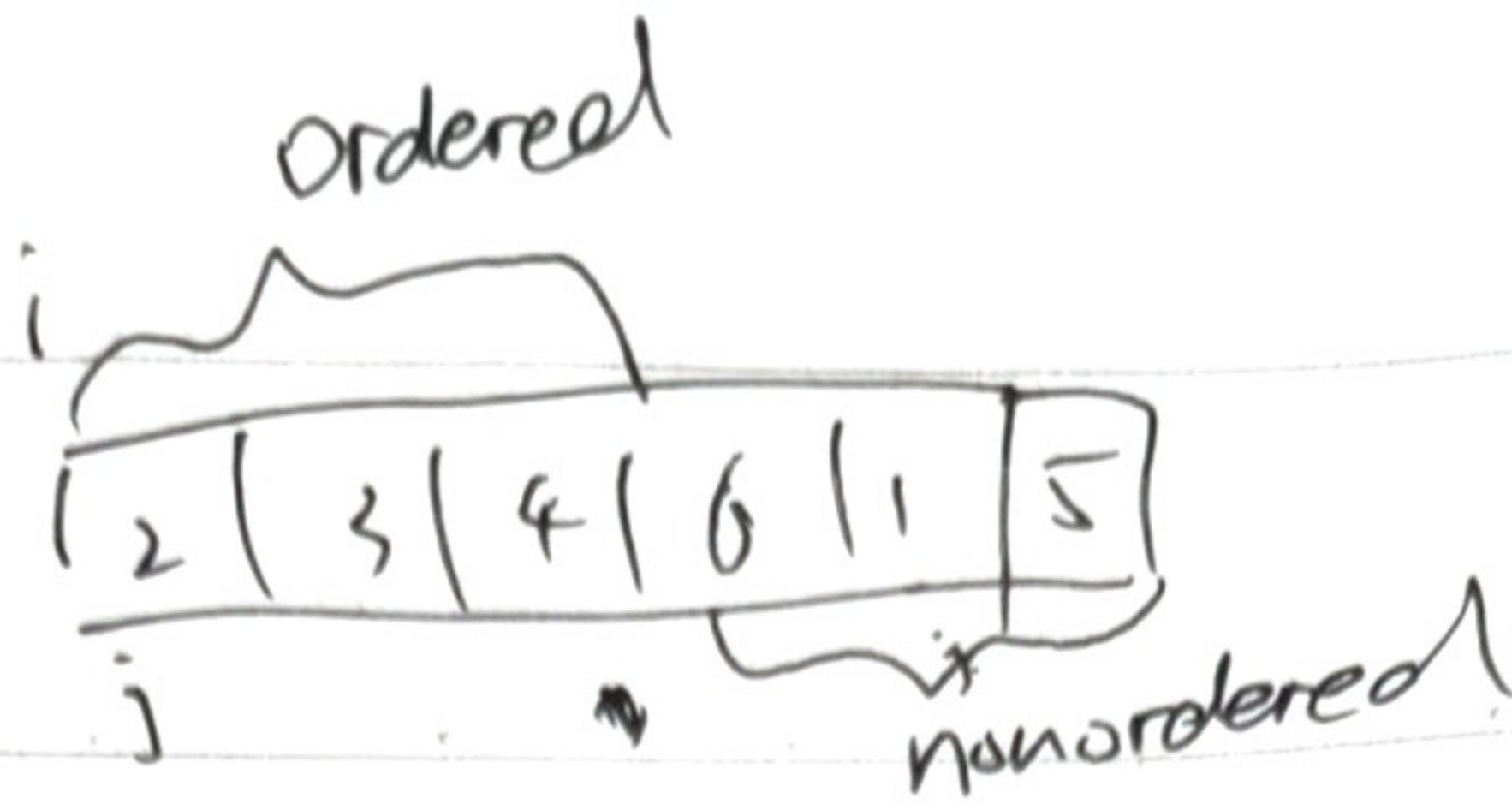
$$O(n^2)$$

	Best	Worst	Avg
Selection	n^2	n^2	n^2

Insertion $n(\text{顺序})$ $n^2(\text{倒序})$ n^2 Merge $n \log n$ $n \log n$ $n \log n$

place
in-

Insertion Sort



arr[i, n) 未排序

arr[0, i) 已排序.

把 arr[i] 放到合适的位置.

(循环不变量)

和 selection sort 不同的是：这里只代表 arr[0, i)

这一部分排好了，不代表这是最终整个数组排序的最终结果

将 arr[i] 插入到合适的位置

① $\text{for } (i=0, i < n, i++)$ $\text{for } (j=i, j-1 \geq 0, j--)$
 $\text{if } arr[j] < arr[j-1]$
 $\text{swap}(, ,)$
 else break

* Simplify: (get rid of "break")

 $\text{for } (i=0, i < n, i++)$ ② $\text{for } (j=i, (j-1 \geq 0) \& (arr[j] < arr[j-1]), j--)$
 $\text{swap}(, ,)$

optimization: 在 swap(arr, j, j-1) 都是三次操作

→ 将 swap 改为赋值, which 只操作一次

③ (但不影响 $O(n^2)$)

```
for (i=0, i<n, i++) {
```

// 将 arr[i] 插入到合适的位置：通过每存 ~~arr[i]~~ arr[j] 的值

 t = arr[i]

 int j;

```
    for (j=i, j-1 >= 0 && t < arr[j-1], j--) {
```

 arr[j] = arr[j-1];

}

 arr[j] = t.

}

优点：① 与底层 JVM 优化 ^{TC} 有关

② n^2 , 寻址次数少

\rightarrow 无论 input 是怎样的，都是 $O(n^2)$

Best case: $O(n)$ 和 Selection Sort 不同。

$\star \rightarrow$ (内层 inner loop 在近乎有序的 input 下, 只执行一次)

但整体还是 $O(n^2)$

Conclusion: ① 循环不变量

② selection 和 insertion 都是基于可以比较的

具体 compare to 这个方法是 ~~根据类里设计~~ 根据类里设计. 取略

③ 复杂度 $O(n^2)$

④ Insertion 对完全有序的数组, $O(n)$

(shell sort)

Merge sort

recursion

mergesort(arr, l, r)

① 对 arr [l, mid] 排序

② 对 arr [mid+1, r] 排序

③ 合并 [l, mid] [mid+1, r].

~~pseudo-code:~~

mergesort (arr, l, r) {

最基础问题. if ($l \geq r$) ~~return~~ return;int ~~mid~~ = $(l+r)/2$

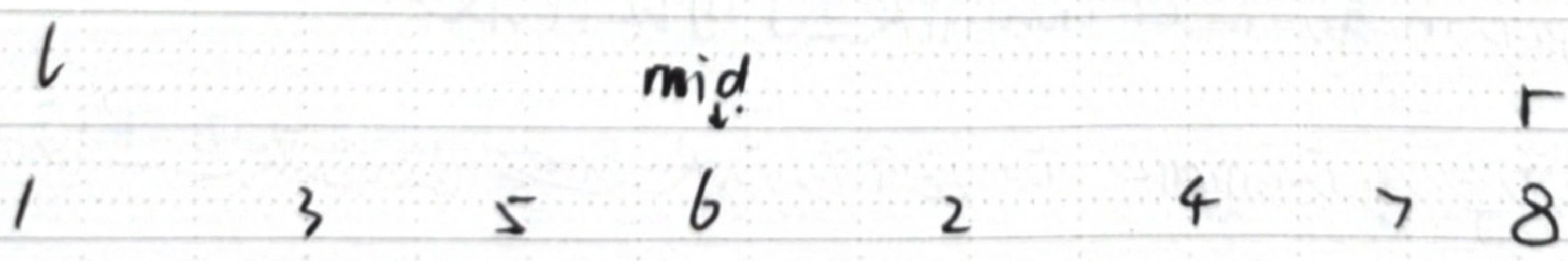
把原问题

① mergesort (arr, l, mid)

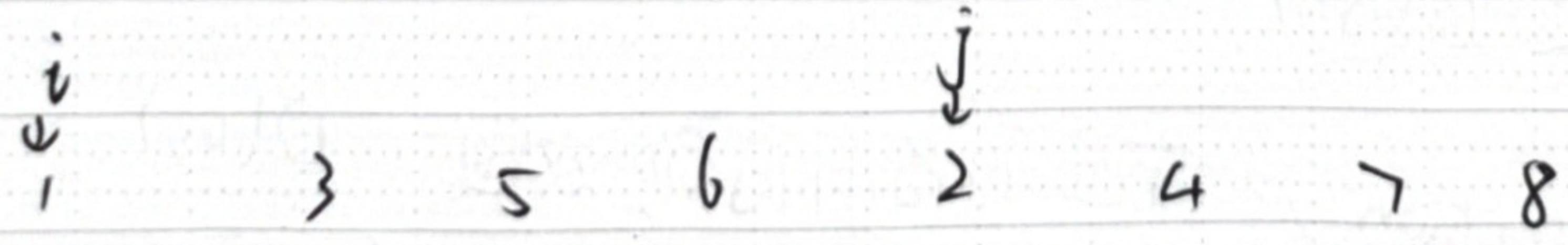
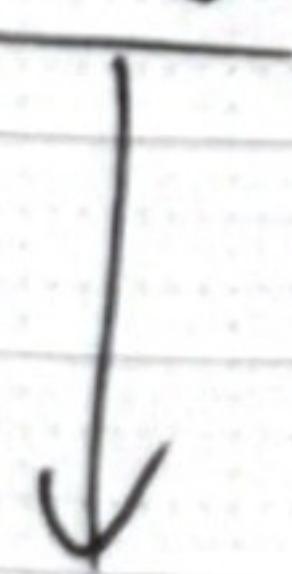
转化为更小

② mergesort (arr, mid+1, r)

阶段问题.

③ ~~mergesort~~ & merge (arr, l, mid, r)

copy:



is not in-place algorithm

TIPS

合并部分：合并 $\text{arr}[l, \text{mid}]$, $\text{arr}[\text{mid}+1, r]$.

merge {

 E[] temp = copy (arr, l, r+1)

 int i=1, j=mid+1

// 每轮循环为 arr[k] 赋值.

 for (int k=l, k<=r, k++) {

 if (i > mid) {

 arr[k] = temp[j-₁^l]
 $\underbrace{\hspace{1cm}}$
 \star
 \downarrow

(Tips): arr[l] = temp[0]?
 $+1 \rightarrow 1$
 $j \rightarrow j-l$.

 j++;

}

 else if (j > r) {

 arr[k] = temp[i-₁^l];
 \downarrow
 i++;

 else if (temp[i-1] ≤ temp[j-1]) {

 arr[k] = temp[i-1];
 i++;

 else {
 arr[k] = temp[j-1];

 j++;

}

}

分治：

```
public void sort (arr) {
    sort (arr, 0, arr.length - 1);
```

y

```
private void sort (arr, l, r) {
```

if ($l >= r$) return;

int mid = $\frac{l+r}{2}$ → can be $\frac{l+(r-l)}{2}$ 防止溢出
(32位整型)

sort (arr, l, mid);

sort (arr, mid+1, r);

merge (arr, l, mid, r);

$O(n \log n)$

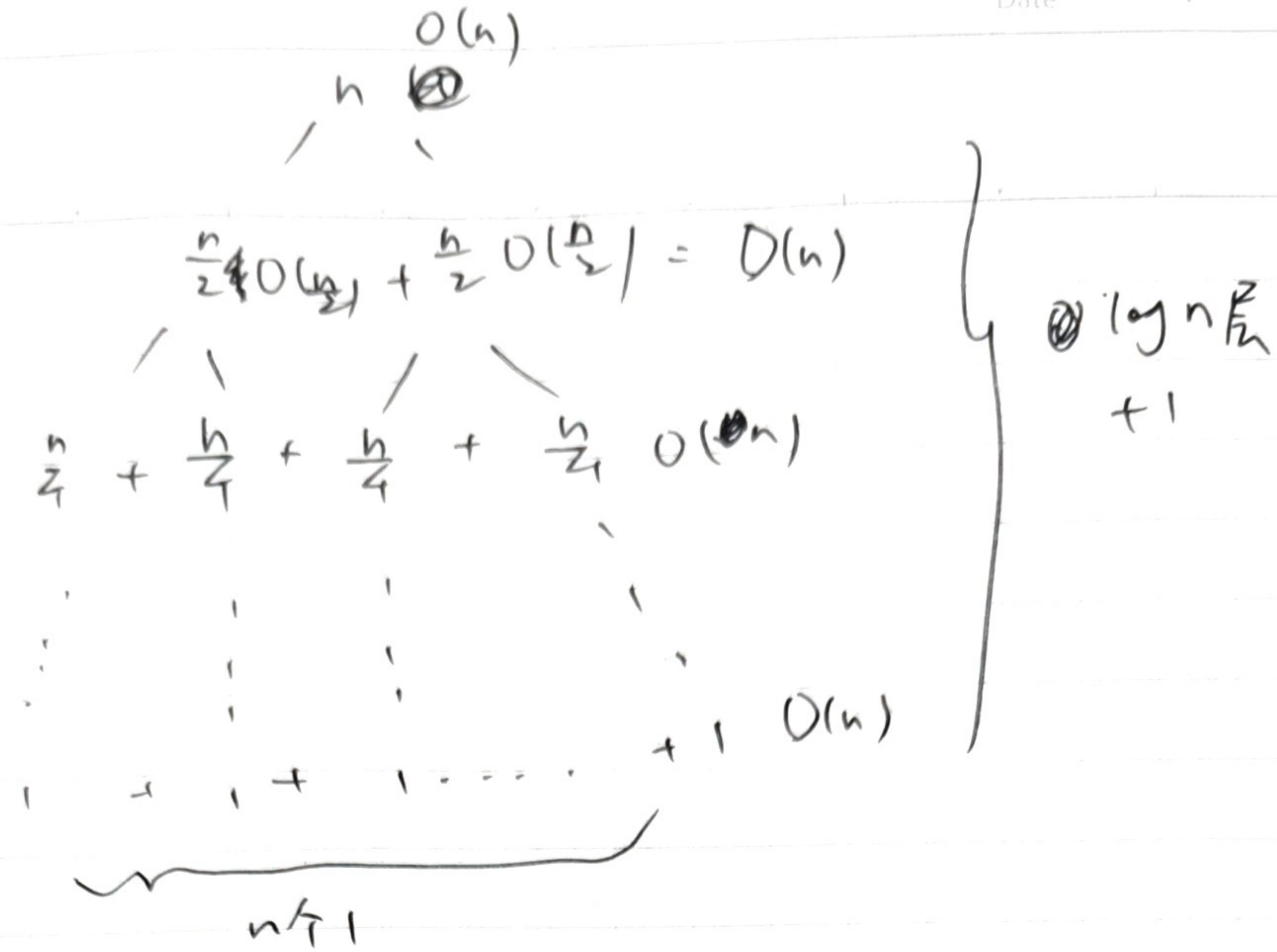
Optimization:

1. if $[arr[mid]] > arr[mid+1]$, 不执行 merge.

However, 当 input 是 Bestcase 时. ~~(最好)~~ $O(n)$ (分析见下页)

2.

Date

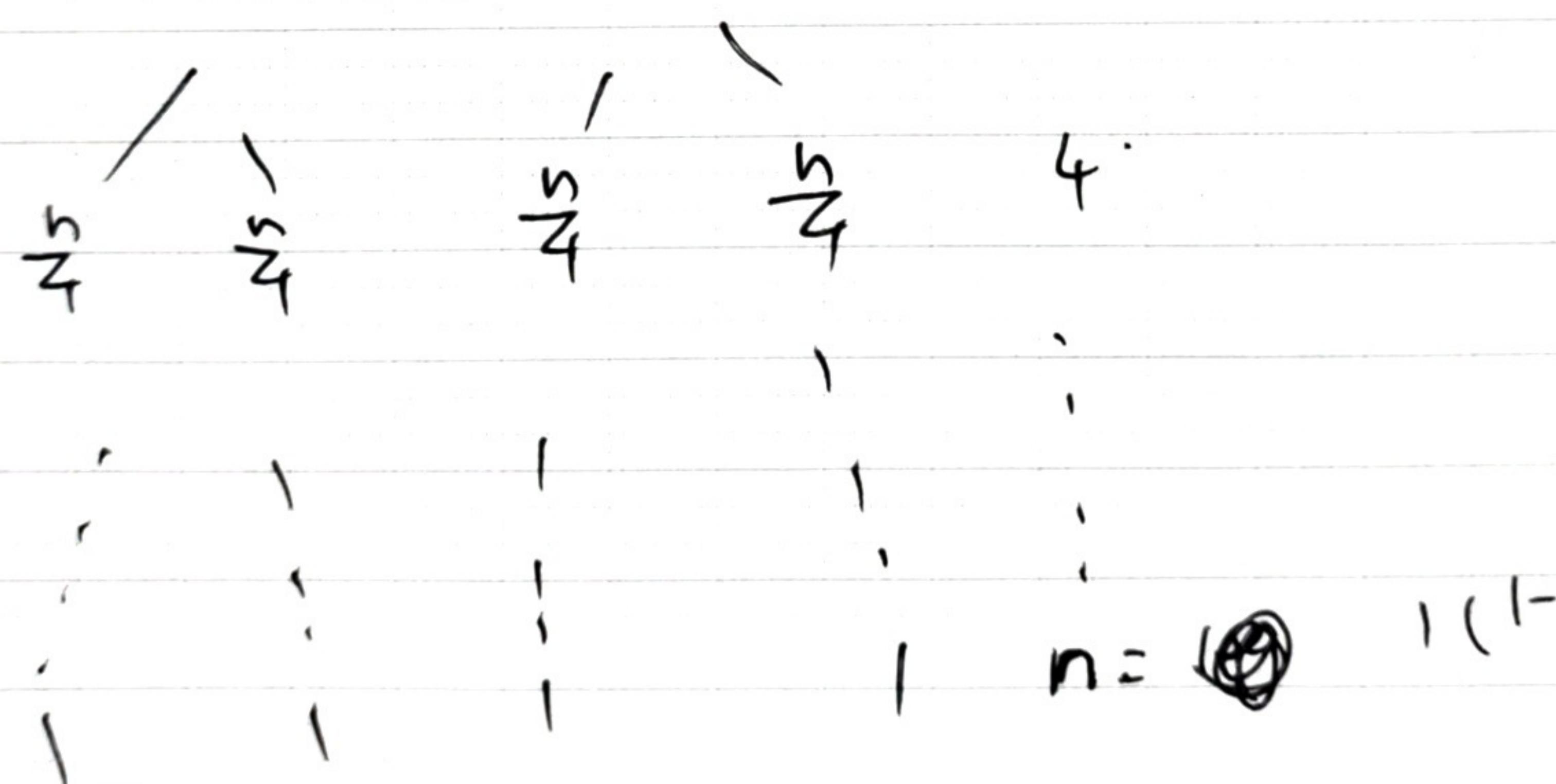


Best case :

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

$$\frac{n}{2^{k_1}}$$

log vt

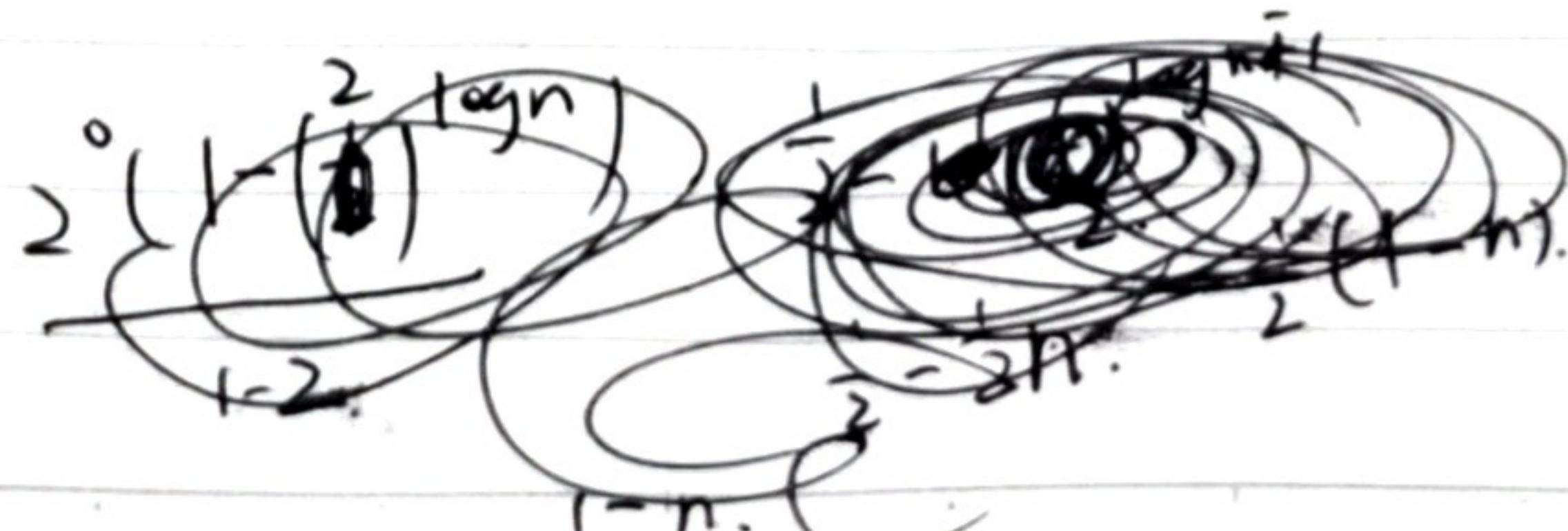


树的节枝、板。

$$1+2+4t \cdot \ln$$

$$1 + \log n \uparrow \\ \text{faktoriell: } 2^0 + 2^1 + 2^2 + \dots 2^{\log n} \quad \cancel{< n} \xrightarrow{\delta} O(n).$$

$$S = \frac{a_1(1-q^n)}{1-q} - \frac{1(1-2^{\log n+1})}{1-2}$$



JS
Python, PHP 等)

optimization: (不适用于所有高级语言)

2. Insertion: code 简单 $O(n^2)$ → 前面常数小.

Merge: code 复杂 $O(n \log n)$
→ 前面常数大

∴ 只有在 n 特别大时, merge 才明显优于 insertion,
otherwise, 有可能 insertion 快于 merge.

∴ optimization: if ($r - l \leq 15$) {

insertionsort.

} $\frac{\text{return}}{\delta}$.

3. 避免一次又一次的开空间 temp.