Data Structure

- Search tukuo123 on Youtube

- Grading Policy

Exam x2	(2x25%)
Assignments x4	(4x10%)
QA	(10%)

Objectives:

1° Find the "desired" data in a collection of data "efficiently"

2° Update a collection of data "efficiently"

3° How to store a collection of data in memory

Example: library, Student Score

```
class Student {
    int id;
    int score;
}
```

Array

n: # data

① Find the top 10 students

② Find all students whose score is between 60 ~ 80

③ Given a student ID, find the corresponding score

④ Modify the score of a given student

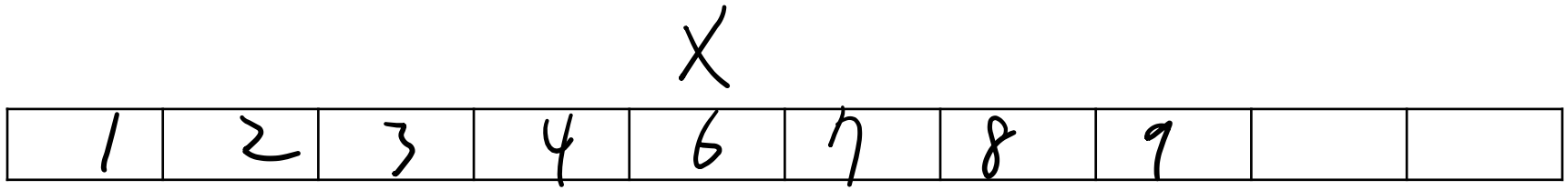
⑤ ADD / DELETE student

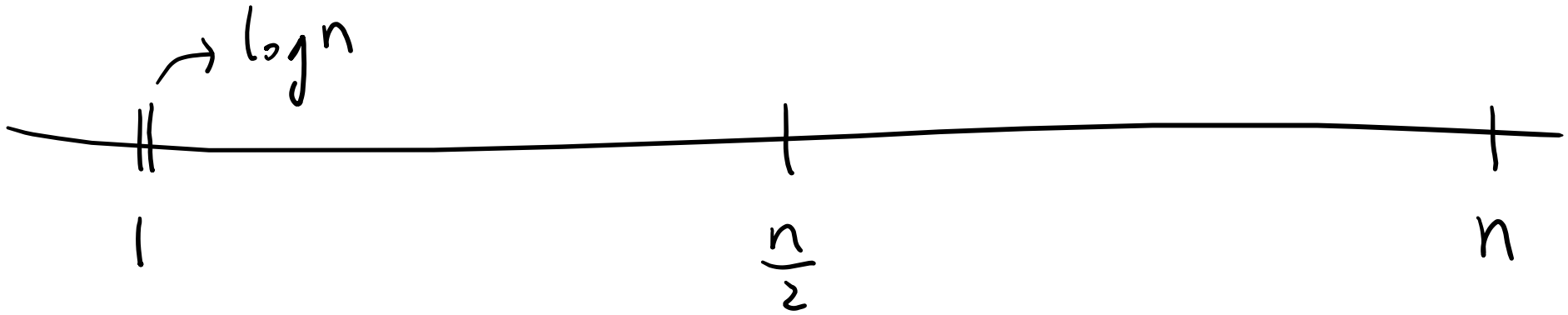
n^2

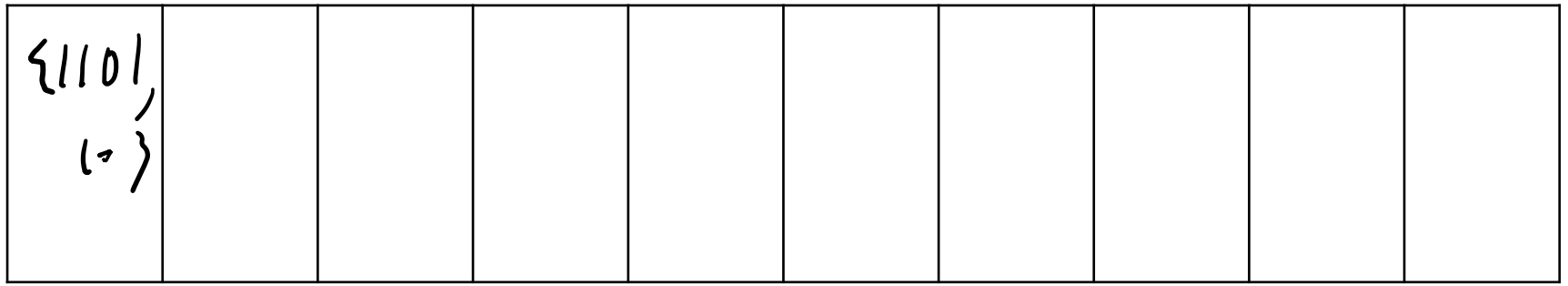
$n \log n$

max = 6

pos = 3






{1101,
1-}

Big-O Notation (Upper bound) $\leq$

$n^2 = O(n^3)$ $\quad$ $n^2 = O(n^4)$ $\quad$ $n^2 = O(n^2)$

$2n^2 = O(n^2)$ $\quad$ $2n^2 + n + 3 = O(n^2)$ $\quad$ $3 = O(1)$

$n^2 \neq O(n)$

$f(n) = O(g(n))$ if $\exists\ c > 0,\ n_0 > 0$

such that $f(n) \leq c \cdot g(n)$ $\quad \forall\ n \geq n_0$

$$\log n = O(\sqrt{n}) = O(n^{1/2})$$

$\log_{10} n \qquad \log_2 n \qquad \ln n$

differ by a constant

$$\log n = O\left(n^{0.000000001}\right)$$

$O(\log n)$
$||$

$$(1)+\left(\frac{1}{2}\right)+\left(\frac{1}{3}+\frac{1}{4}\right)+\left(\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}\right)+\left(\frac{1}{9}+\frac{1}{10}+\frac{1}{11}+\frac{1}{12}+\frac{1}{13}+\frac{1}{14}+\frac{1}{15}+\frac{1}{16}\right)+\cdots+\frac{1}{n}$$

$$\leq (1)+(1)+\left(\frac{1}{2}+\frac{1}{2}\right)+\left(\frac{1}{4}\ \frac{1}{4}\ \frac{1}{4}\ \frac{1}{4}\right)+\left(\frac{1}{8}\ \cdots\ \frac{1}{8}\right)+\cdots = O(\log n)$$

Big-Ω (lower bound) $\geq$

$n^2 = \Omega(n)$ $\quad$ $n^2 = \Omega(\sqrt{n})$ $\quad$ $0.5n^2 = \Omega(n^2)$

$n^2 - 3n = \Omega(n^2)$

$n^2 \neq \Omega(n^3)$

$f(n) = O(g(n))$ if $\exists\ c > 0,\ n_0 > 0$

such that $f(n) \geq c \cdot g(n)$ $\forall\ n \geq n_0$

Big-Θ (tight bound) =

$2n^2 = \Theta(n^2)$ $3n^2 + 4n + 3 = \Theta(n^2)$

$2n^2 \neq \Theta(n^3)$ $2n^2 \neq \Theta(n)$

$f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$

& $f(n) = \Omega(g(n))$

$k = O(1)$

$kn = O(n)$

$1 + 2 + 4 + 8 + 16 + \cdots + 2^k = 2^{k+1} - 1$

$\approx 2^{k+1}$

$n = 2^k = 2^{10}$

$1 + 2^x \cdots$

$\Lambda 9$

$\times 2$

2^{10}

$\times 2$

Time Complexity

- Basic operation takes $O(1)$ time

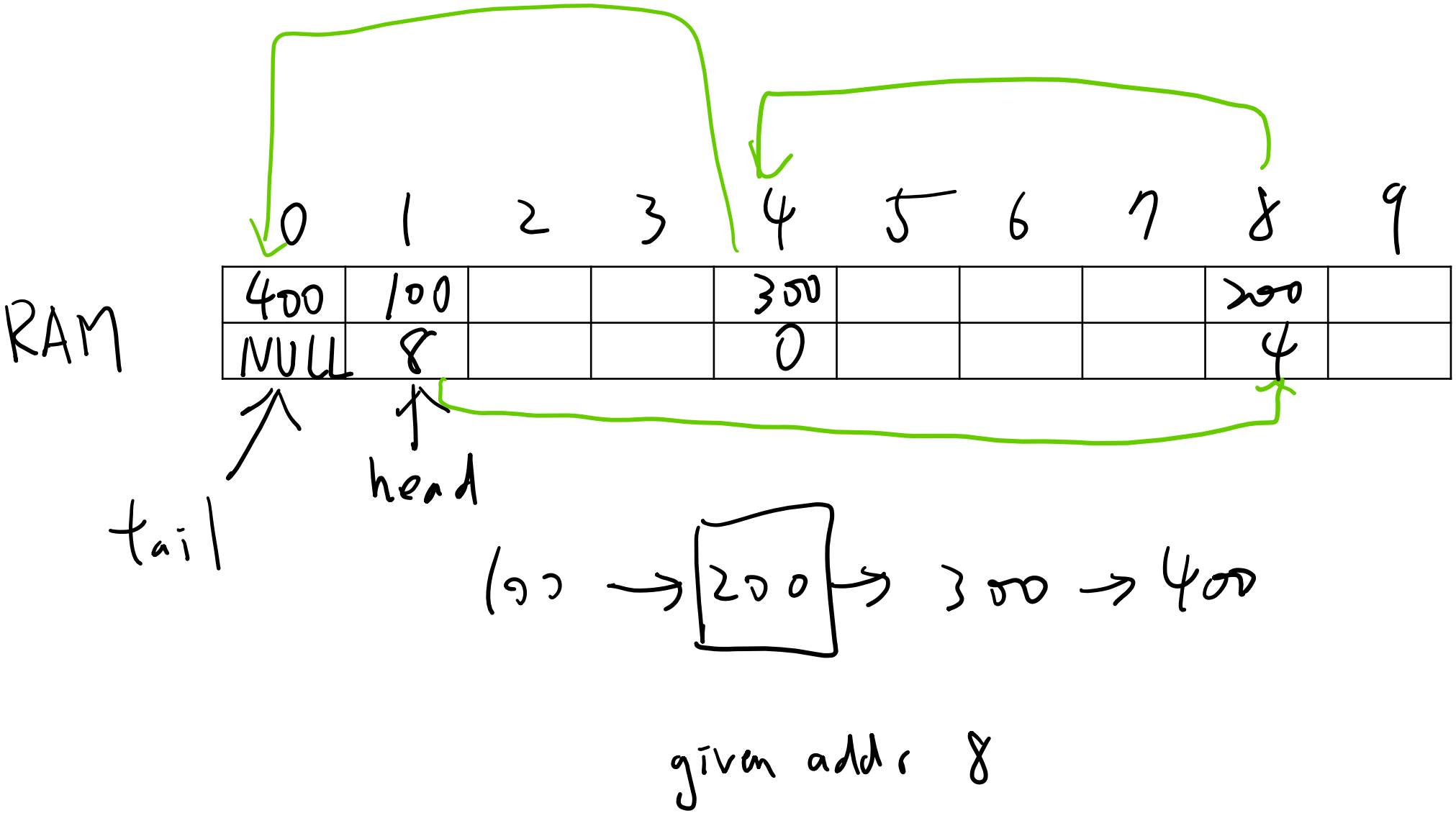
  e.g. $+ - * /$, $\geq$, $==$, $<=$, if, else, , arr[x], init. basic variable

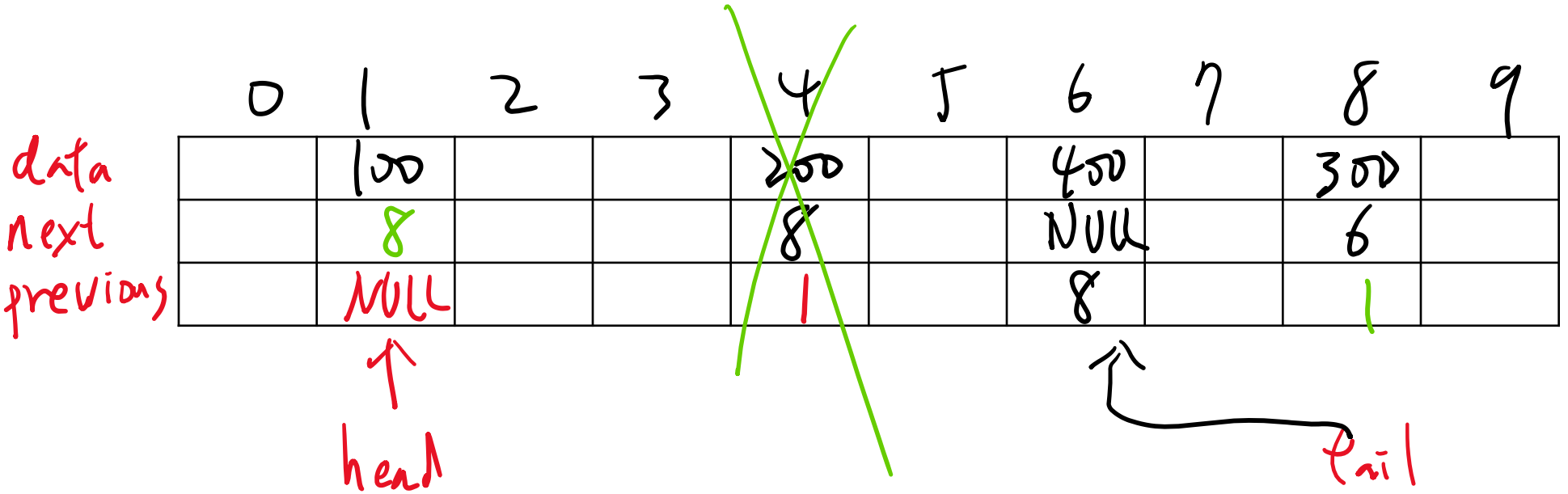
```
          O(1)     O(1)    O(1)
for (int i=0; i<n; i++) {

    sum += i; O(1)          Θ(n)

}
```

```
for (int i=0; i<n; i++){
    for(int j=0; j<m; j++) {
        sum += i*j;
    }
}
```

$\Theta(nm)$

	Array	Linked List
Delete	$O(n)$ $\Theta(k)$ k: # following data	$O(n)$: singly linked list $O(1)$: doubly linked list
Add-back	$O(1)$	$O(1)$
Traverse	$\Theta(n)$ faster due to cache	$\Theta(n)$
Access the k^{th} data	$O(1)$	$\Theta(k)$
Search data	$O(n)$ faster due to cache	$O(n)$

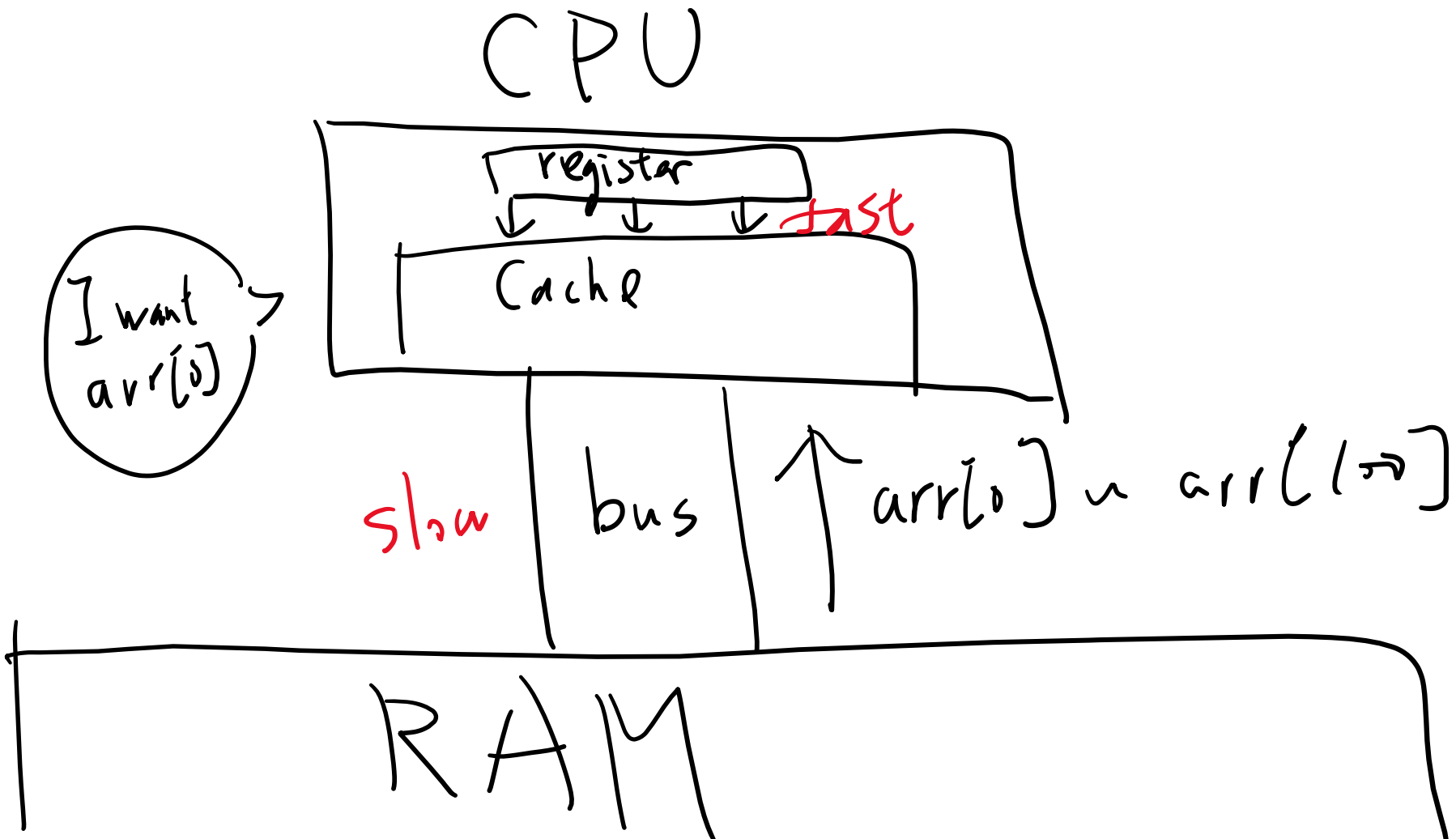

0 1 2 3 4 5 6 7 8 9
RAM
400 100 300 200
NULL 8 0 4
tail
head
100 → 200 → 300 → 400
given addr 8



100 ⇔ 200 ⇔ 300 ⇔ 400

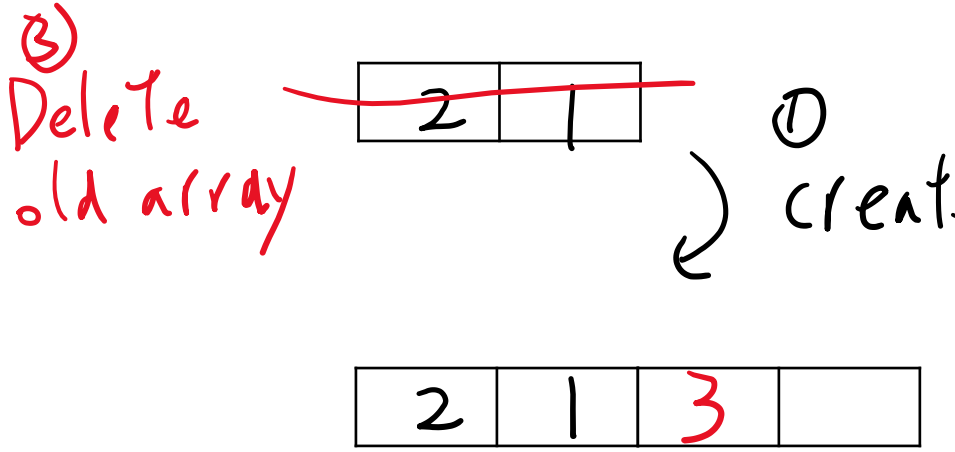
A: ARR[0], ARR[1], ARR[2]

B: ARR[0], ARR[10000], ARR[50000]



1. Dynamic Array & Amortized Time Complexity
2. HW1
3. Data structures covered in this course
4. Sorting Algorithms:
    a. Insertion Sort
    b. Merge Sort (2 versions)
    c. Randomized Quick Sort

Dynamic Array (Vector)



push_back : O(1) if array is not full (best case)

O(n) otherwise (worst case)

Q: How to decide the capacity of the new array

① new capacity = old capacity + c (solution ①)

② new capacity = 2 x old capacity (solution ②)

Amortized Time Complexity

$$\frac{\text{Time complexity to perform a series of } (x) \text{ OP}}{x}$$

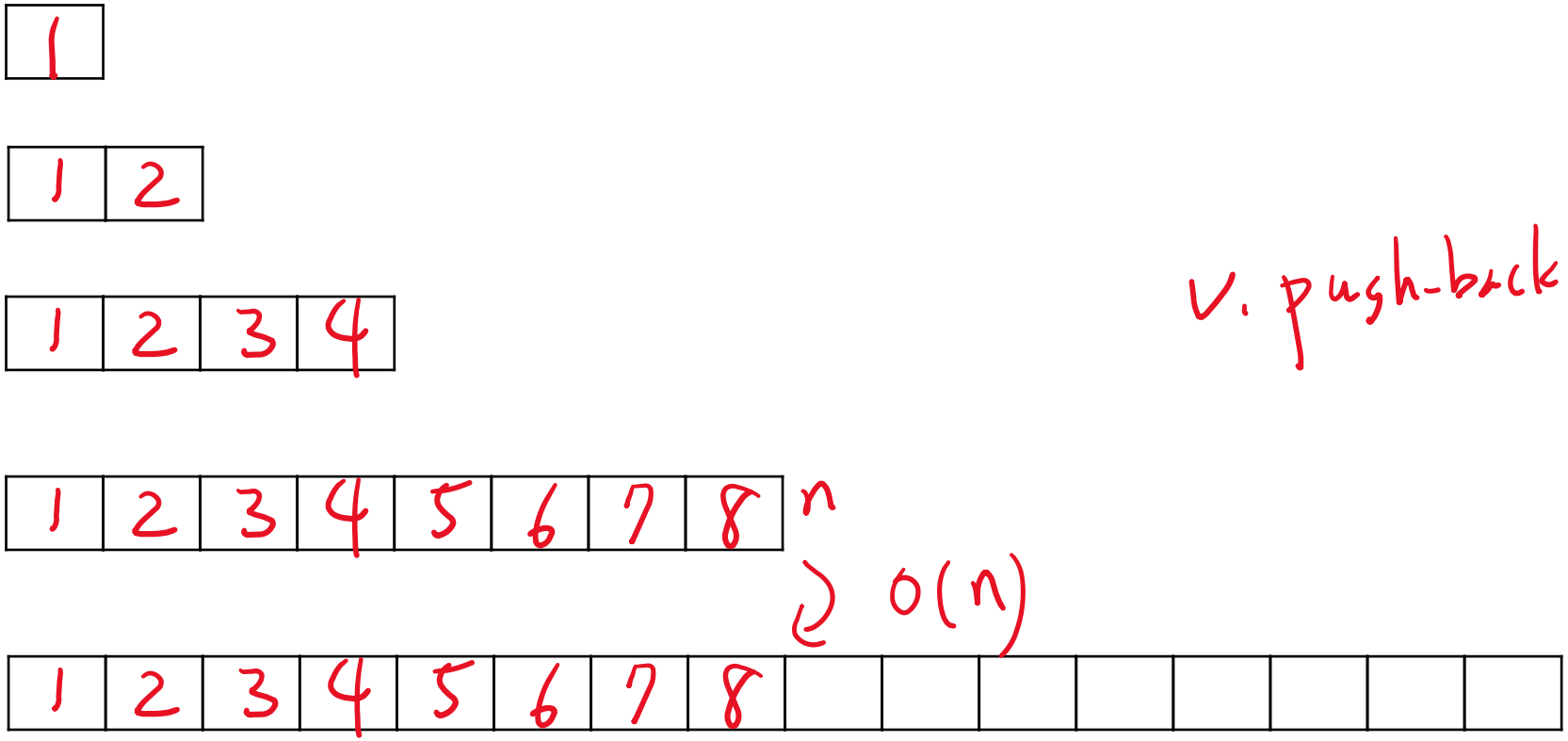
Amortized Time Complexity of solution ①

$$\frac{\Theta(1+2+3+\cdots+n)}{n} = \Theta(n)$$

we need copy-paste for EVERY push_back

Every push_back is a worst-case OP.

Amortized Time Complexity of solution ②



$n = 2^k$

$$1 + 2 + 4 + 8 + 16 + 32 + 64 + \cdots + 2^k = \Theta(2^k)$$

$$= \Theta(n)$$

= Time complexity to perform push_back n times

$\Rightarrow$ Amortized time complexity $= \frac{\Theta(n)}{n} = O(1)$

```cpp
  do {
    std::cin >> myint;
    myvector.push_back (myint);
  } while (myint);

  std::cout << "myvector stores " << int(myvector.size()) << " numbers.\n";

  return 0;
}
```

Edit & run on cpp.sh

The example uses push_back to add a new element to the vector each time a new integer is read.

Complexity

Constant (amortized time, reallocation may happen).

O(1)

If a reallocation happens, the reallocation is itself up to linear in the entire size.

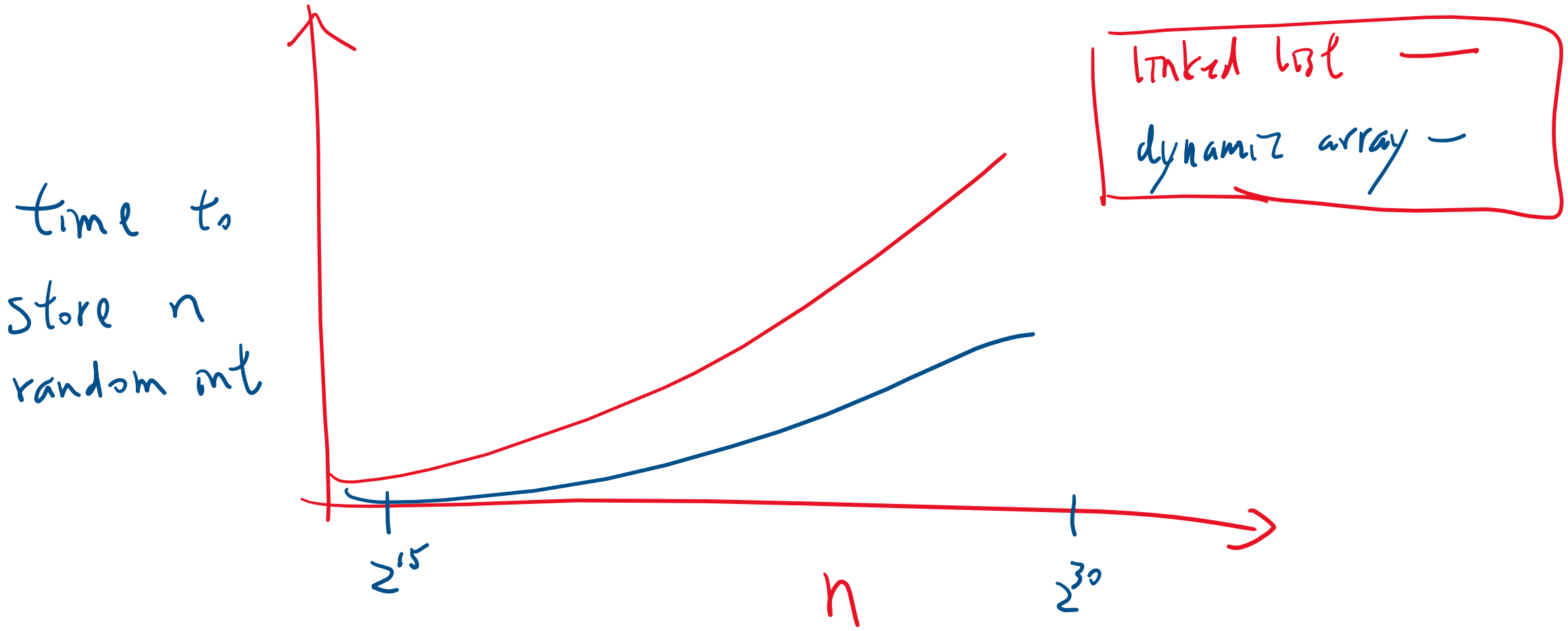
O(n)

Iterator validity

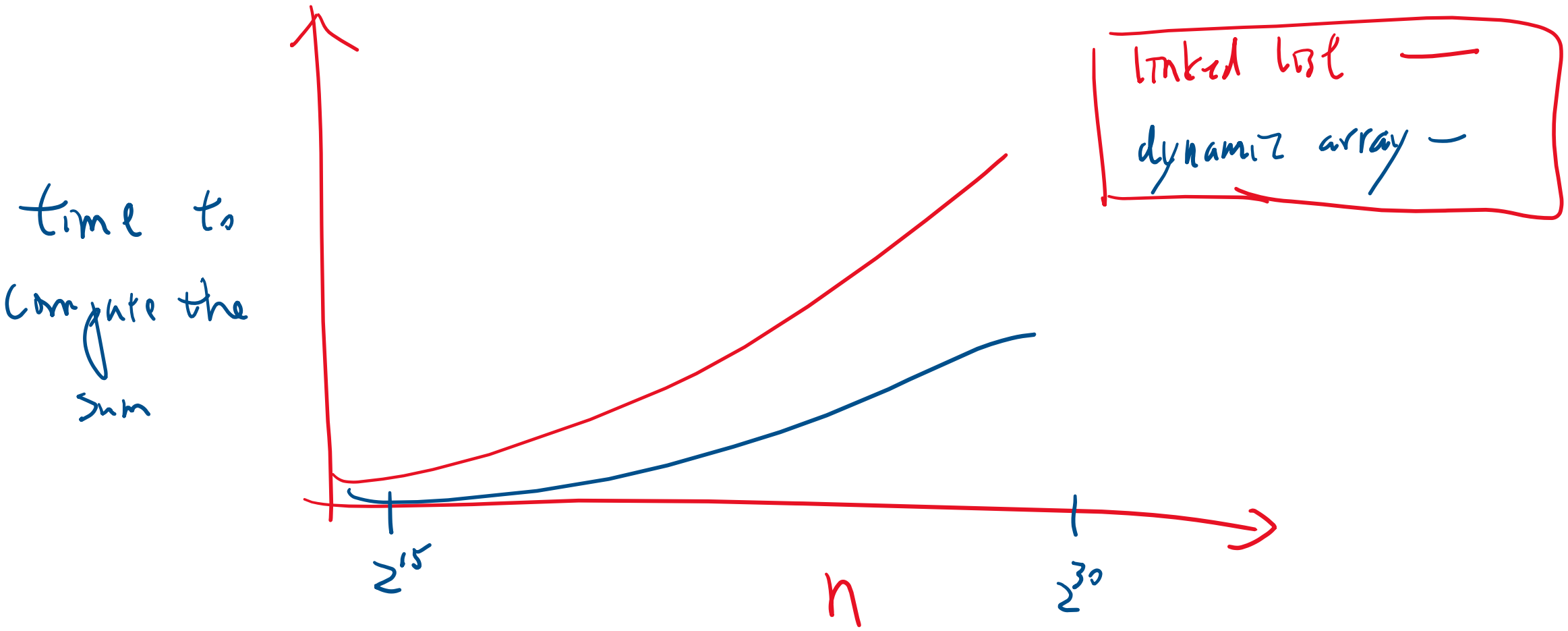
If a reallocation happens, all iterators, pointers and references related to the container are invalidated.

Otherwise, only the end iterator is invalidated, and all iterators, pointers and references to elements are guaranteed to keep referring to the same elements they were referring to before the call.

HW1 (Compare linked list and dynamic array)



HW1 (Compare linked list and dynamic array)



Data Structure Covered in this Course

- Sorted Data structure

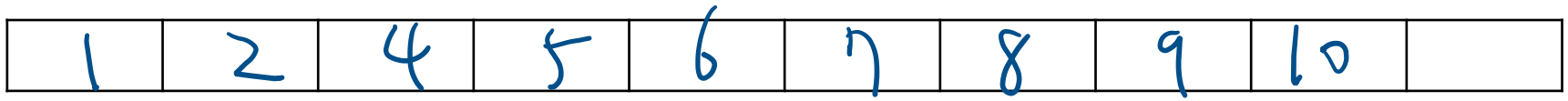
	Sorted Array
search	$O(\log n)$
insert	$O(n)$
find_next	$O(1)$

① B-tree

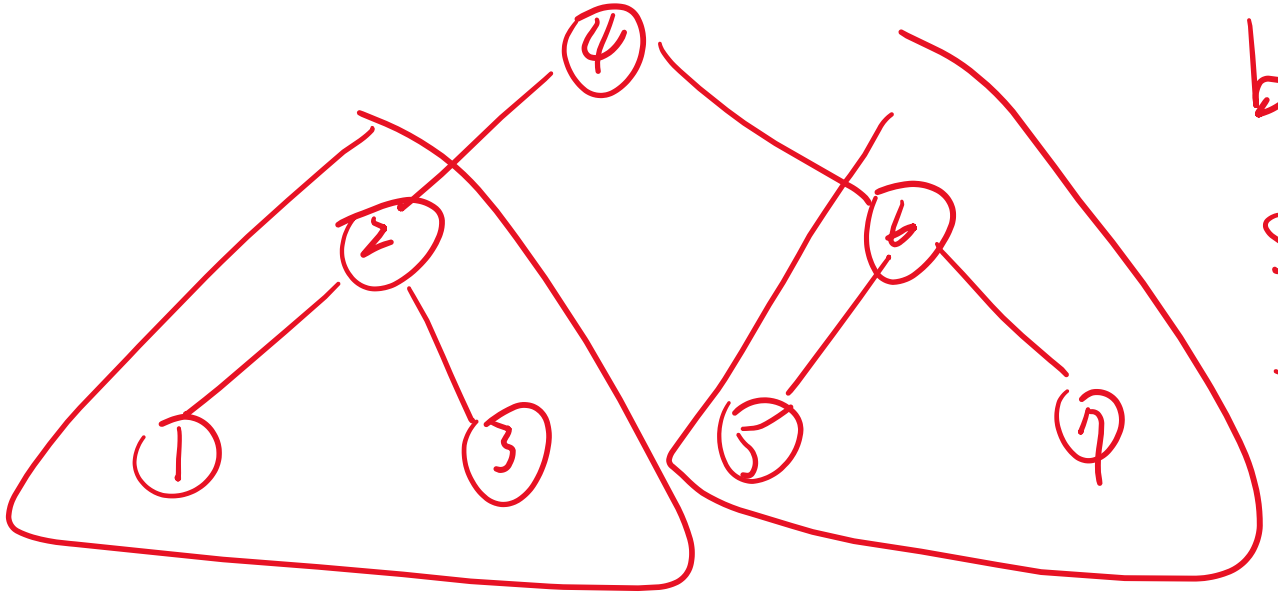
② skip list

③ treap

randomized data structure



1	2	3	4	5	6	7			



binary search tree

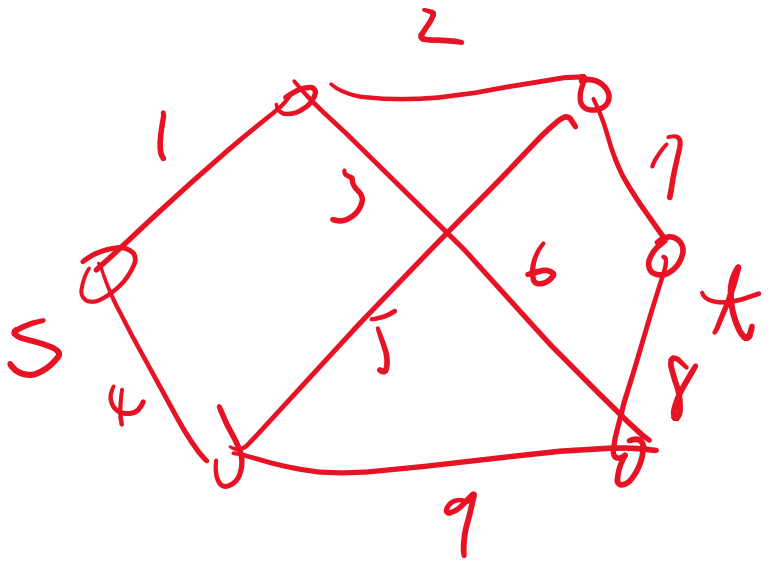
Hash Table (Randomized data structure)

- O(1) search
- O(1) insert

Heap

- Insert
- Decrease Key
- Extract Min

Shortest Path Algorithm

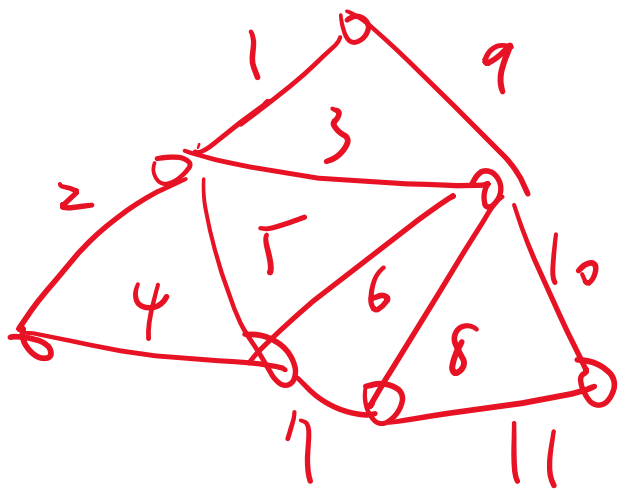


Disjoint Set

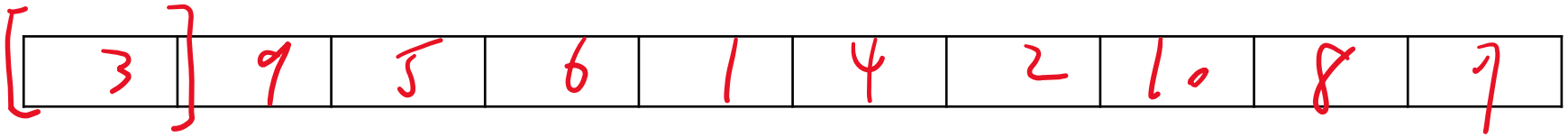
{a, b, c}, {d, e, f}, {g, h}

- Init Set
- Find
- Union

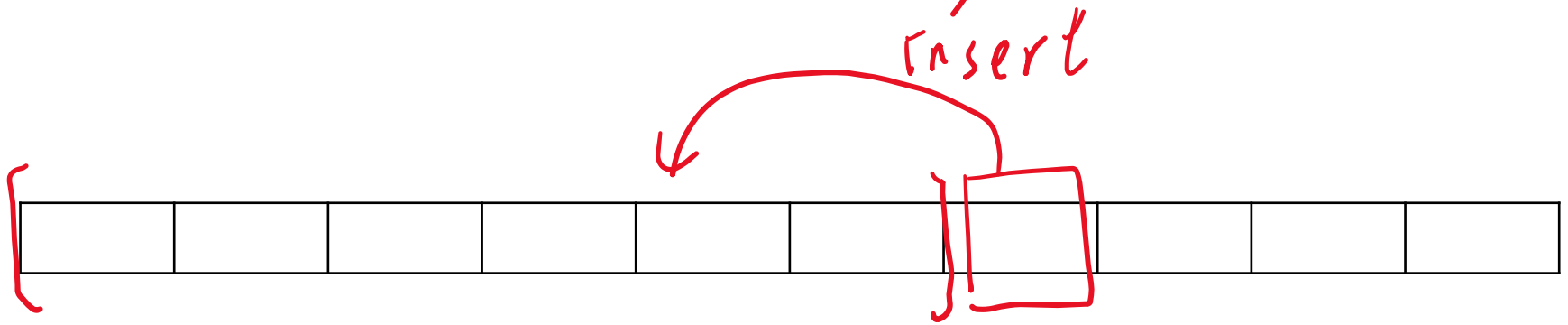
Minimum Spanning Tree

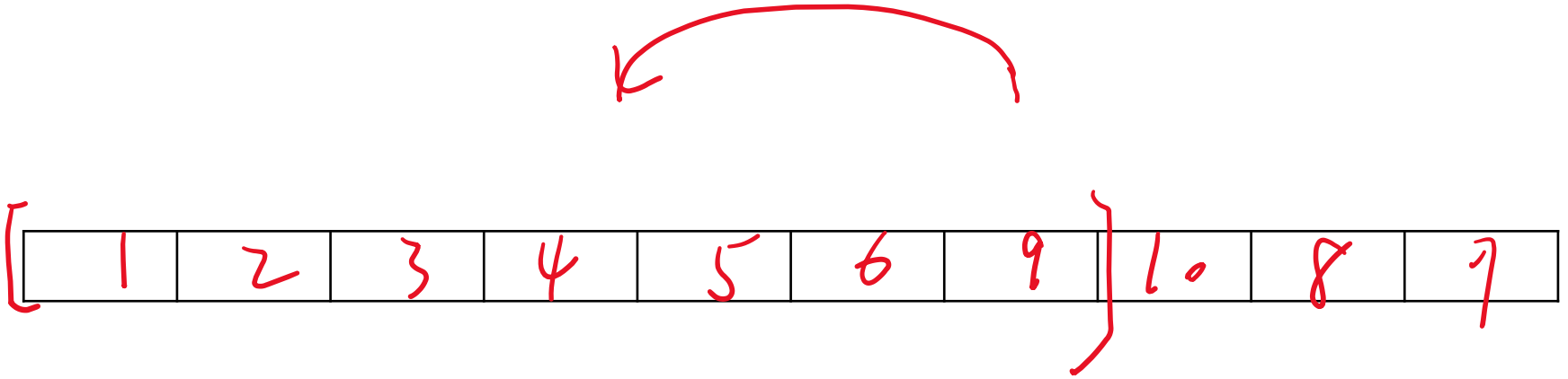


Insertion Sort



[ ] : sorted subarray





tmp = 4

In the i^{th} iteration: $O(i)$ time (worst-case)

* insert arr[i] to the correct position

method 1:
① find the correct position
② Shift right the remaining array

method 2:
swap arr[i] & arr[i-1] ↺ repeat
until i == 0 or arr[i-1] < arr[i]

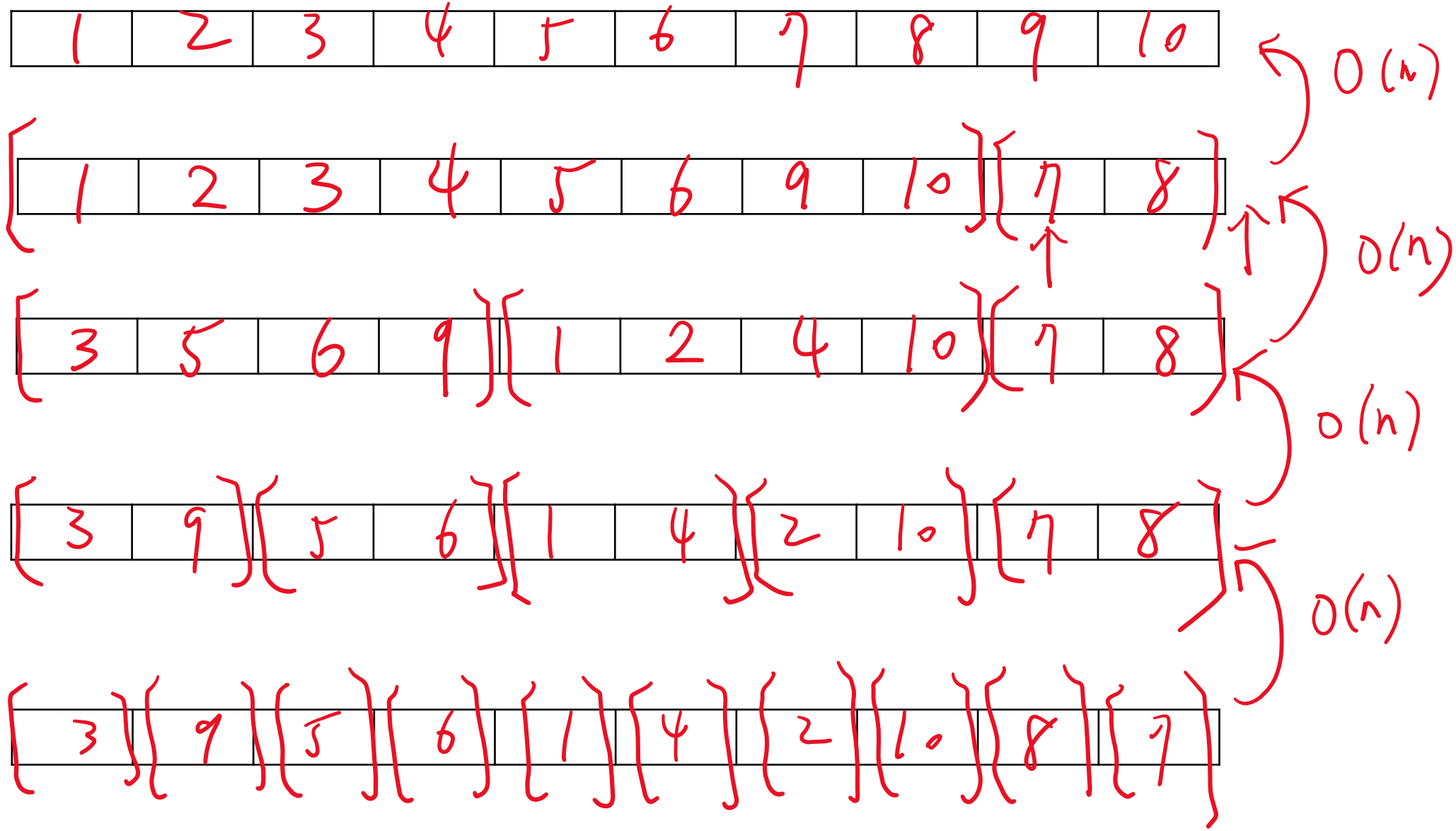
Total Time complexity

$= O(1+2+3+\cdots+n)$

$= O(n^2)$ (worst-case)

Best Case : $O(n)$ (when there are only $O(1)$ "errors" in the input array)

Merge Sort (Bottom Up Approach)



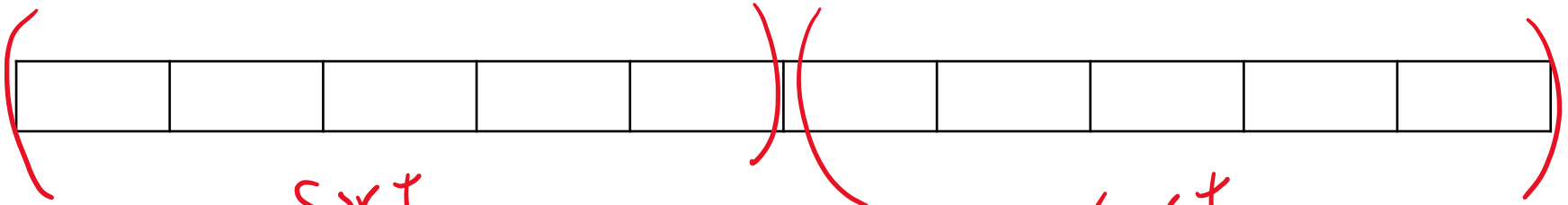
Merge Sort (Recursive)

① If $n \leq 2$, sort the array by comparing at most 2 numbers.

$T(\frac{n}{2})$ ② Merge Sort $arr[0 \to \frac{n}{2}]$

$T(\frac{n}{2})$ ④ Merge Sort $arr[\frac{n}{2}+1 \to n-1]$

$O(n)$ ③ Merge $(arr[0 \to \frac{n}{2}], arr[\frac{n}{2}+1 \to n-1])$



sort

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

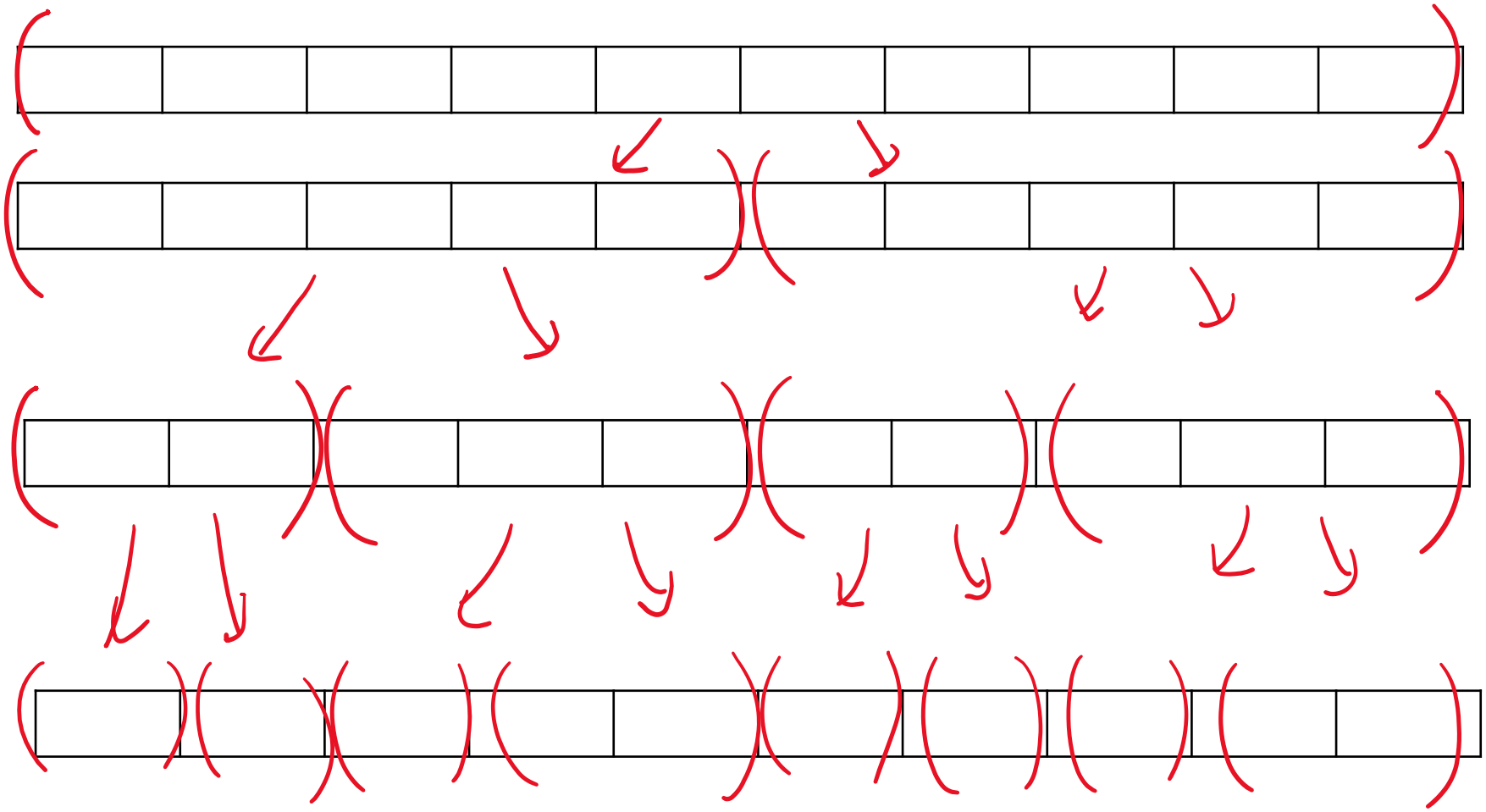
sort

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

merge $O(n)$

$T(n)$: Time complexity of merge sort on a size-n array

$T(1) = O(1)$



(Top-Down Approach)

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n) \\ T(1) = O(1) \end{cases}$$

$O(n)$ ← merge

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

For any constant c, $T(c) = O(1)$

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

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

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

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

$$= 4T\left(\frac{n}{4}\right) + O(n) + O(n)$$

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

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

$$T(n) = 4T\left(\frac{n}{4}\right) + O(2n) \qquad n \leftarrow n/4$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n/4}{2}\right) + O\left(\frac{n}{4}\right)$$

$$= 2T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)$$

$$4\left(2T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)\right) + O(2n)$$

$$= 8T\left(\frac{n}{8}\right) + O(n) + O(2n)$$

$$= 8T\left(\frac{n}{8}\right) + O(3n) = 16T\left(\frac{n}{16}\right) + O(4n)$$

Assume $n = 2^k$ ($\Leftrightarrow k = \log n$)

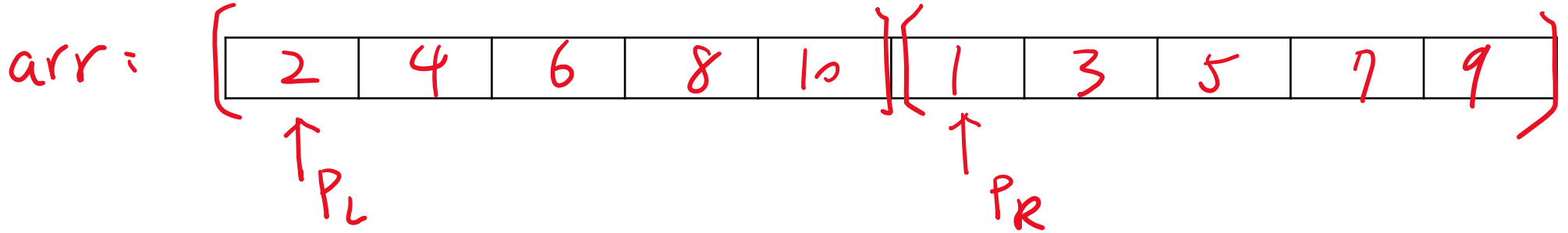
$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + O(in)$$

when $i = k$

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

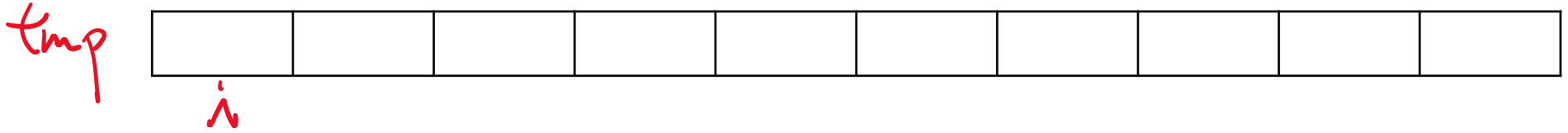
$$= \boxed{2^k O(1)} + O(kn)$$

$$= \boxed{O(n)} + O(n \log n) = O(n \log n)$$

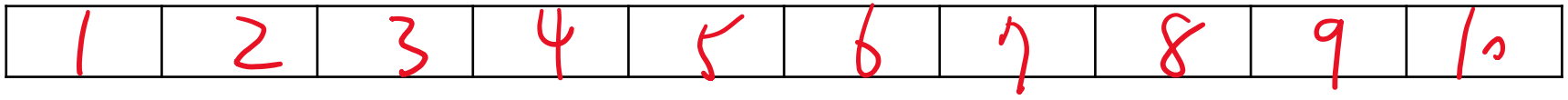


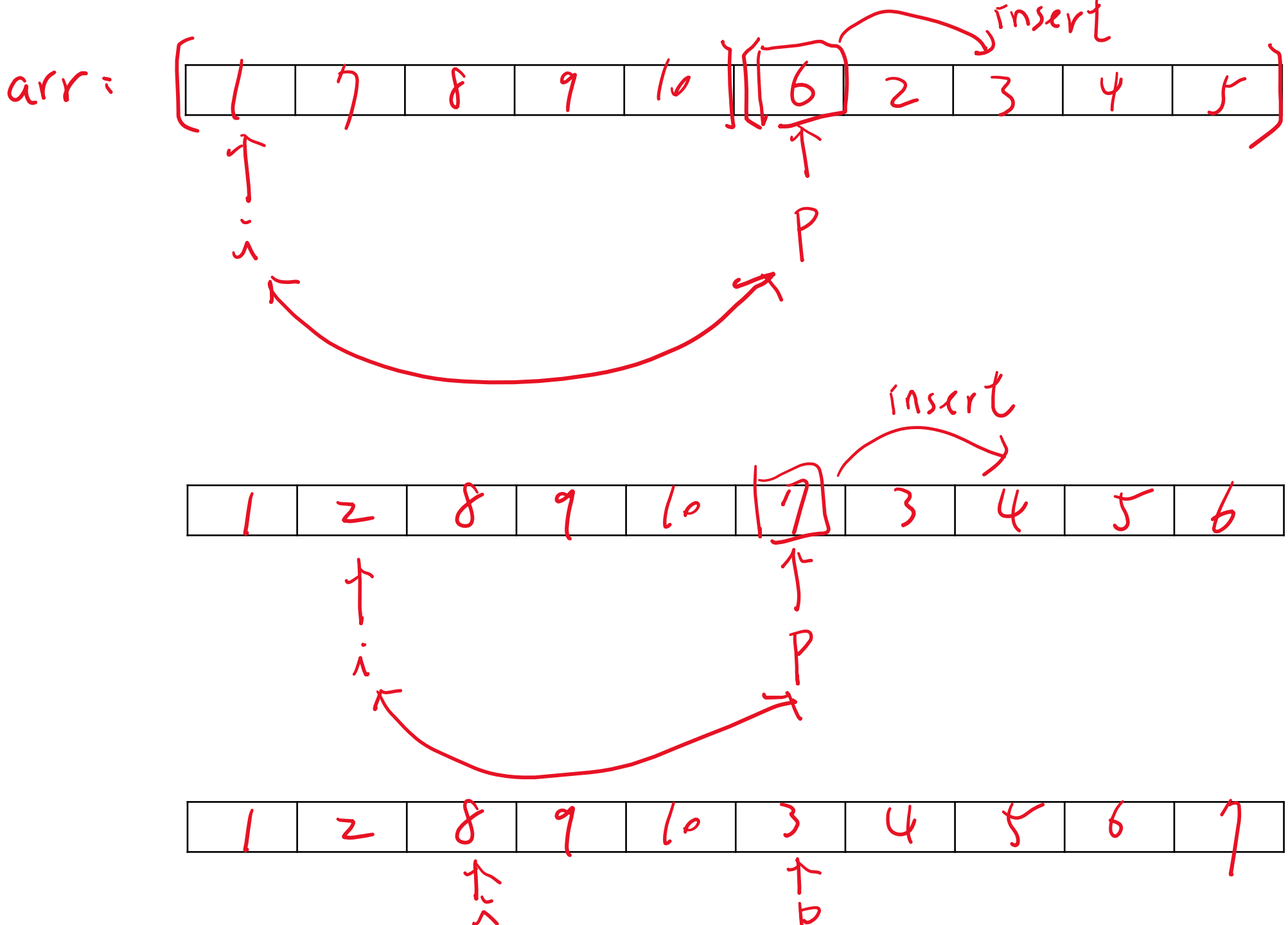

```
if ( arr[p_L] ≤ arr[p_R] ) {
    tmp[i++] = arr[p_L++]
}
```

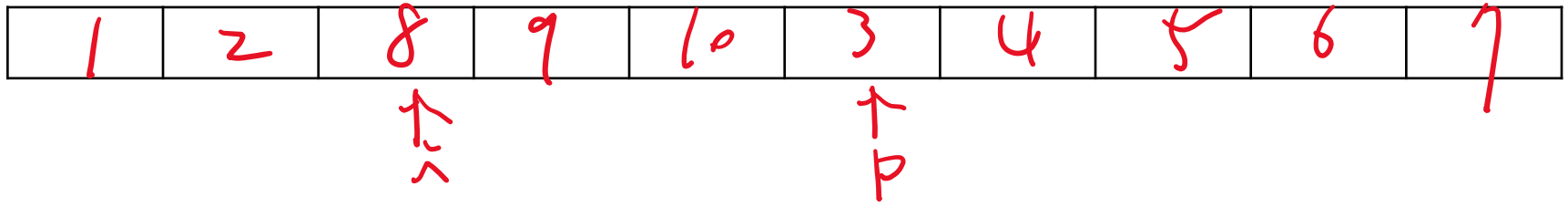
```
if ( arr[p_R] ≤ arr[p_L] ) {
    tmp[i++] = arr[p_R++]
}
```



goal: arr:




arr:
1
7
8
9
10
6
2
3
4
5
insert
i
P
1
2
8
9
10
7
3
4
5
6
insert
i
P
1
2
8
9
10
3
4
5
6
7



$O(n)$

```
for ( i = 0 ~ n/2 ) {
    if (arr[i] > arr[p]) {
        swap (arr, i+1, p); insert(arr, p);
    } else {
        i++;
    }
}
```

$O(n)$

$O(n^2)$

$$\begin{cases} T'(n) = 2T'\left(\frac{n}{2}\right) + O(n^2) \\ T'(1) = O(1) \end{cases}$$

$$T'\left(\frac{n}{2}\right) = 2T'\left(\frac{n/2}{2}\right) + O\left(\left(\frac{n}{2}\right)^2\right)$$

$$= 2T'\left(\frac{n}{4}\right) + O\left(\frac{n^2}{4}\right)$$

① $O(n^2 \log n)$

② $O(n^3)$

③ $O(n^2)$

$$T'(n) = 2\left(2T'\left(\frac{n}{4}\right) + O\left(\frac{n^2}{4}\right)\right) + O(n^2)$$

$$= 4T'\left(\frac{n}{4}\right) + O\left(\frac{n^2}{2}\right) + O(n^2)$$

$$T'(n) = 2T'\left(\frac{n}{2}\right) + O(n^2)$$

$$T\left(\frac{n}{4}\right) = 2T'\left(\frac{n/4}{2}\right) + O\left(\frac{n^2}{16}\right)$$

$$= 2T'\left(\frac{n}{8}\right) + O\left(\frac{n^2}{16}\right)$$

$$T'(n) = 4T'\left(\frac{n}{4}\right) + O\left(\frac{n^2}{2}\right) + O(n^2)$$

$$= 4\left(2T'\left(\frac{n}{8}\right) + O\left(\frac{n^2}{16}\right)\right) + O\left(\frac{n^2}{2}\right) + O(n^2)$$

$$= 8T'\left(\frac{n}{8}\right) + O\left(\frac{n^2}{4}\right) + O\left(\frac{n^2}{2}\right) + O(n^2)$$

$$T'(n) = 8T'\left(\frac{n}{8}\right) + O\left(\frac{n^2}{4}\right) + O\left(\frac{n^2}{2}\right) + O(n^2)$$

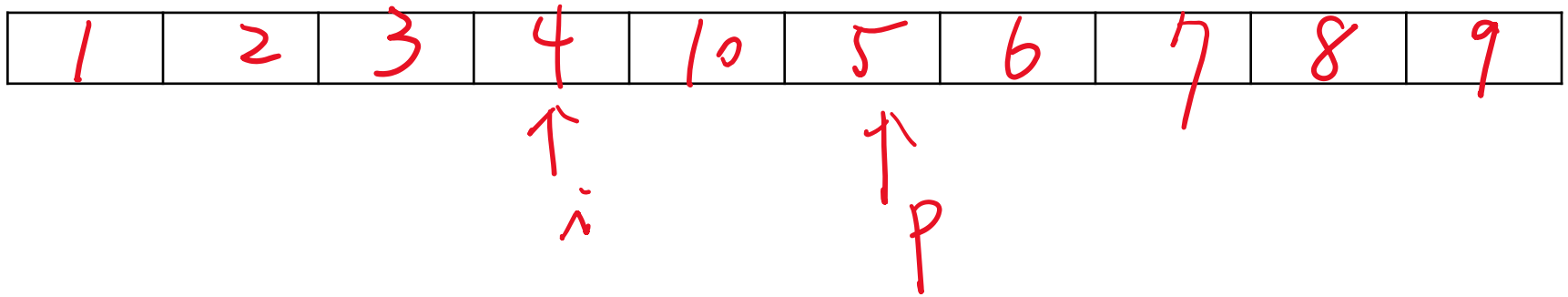
$$= 16T'\left(\frac{n}{16}\right) + O\left(\frac{n^2}{8}\right) + O\left(\frac{n^2}{4}\right) + O\left(\frac{n^2}{2}\right) + O(n^2)$$

$$= 2^i T'\left(\frac{n}{2^i}\right) + O\left(\frac{n^2}{2^{i-1}}\right) + O\left(\frac{n^2}{2^{i-2}}\right) + \cdots + O\left(\frac{n^2}{2^0}\right)$$

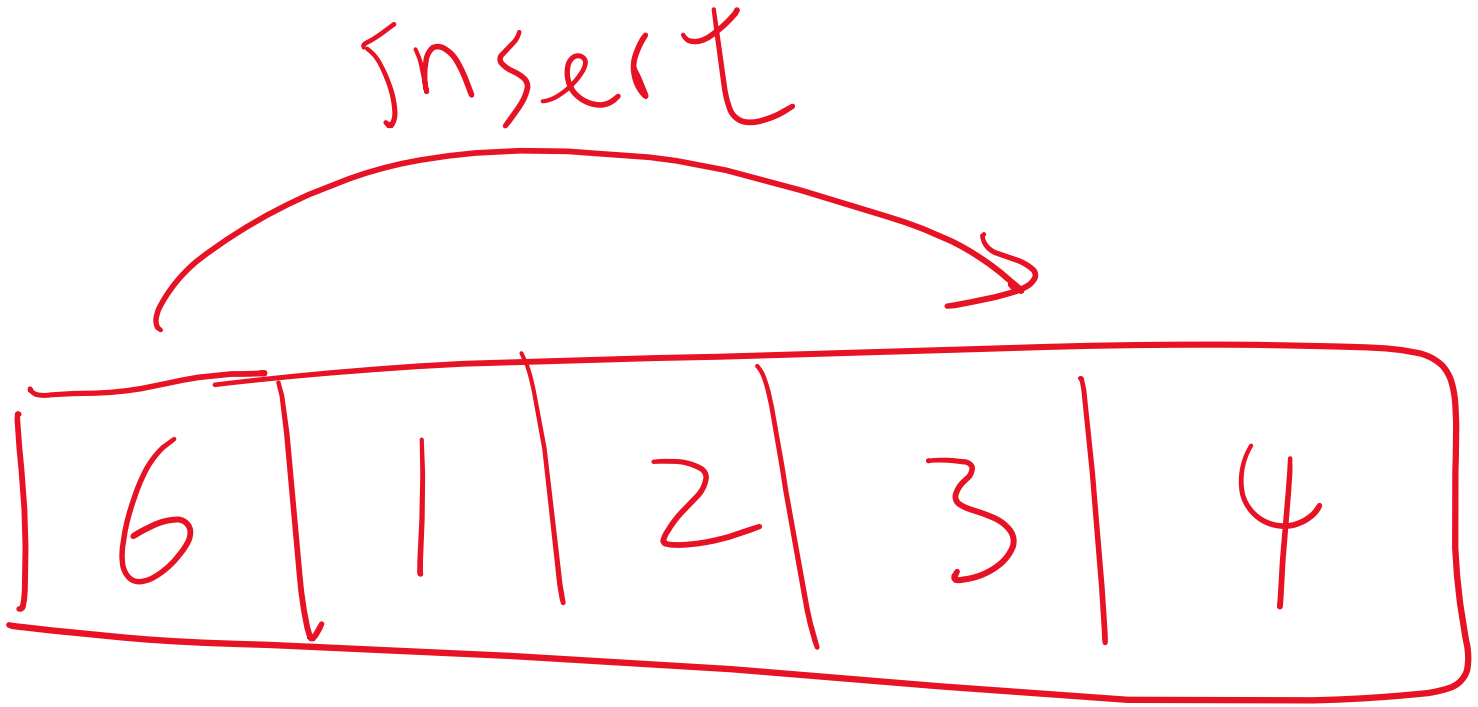
$n = 2^k$

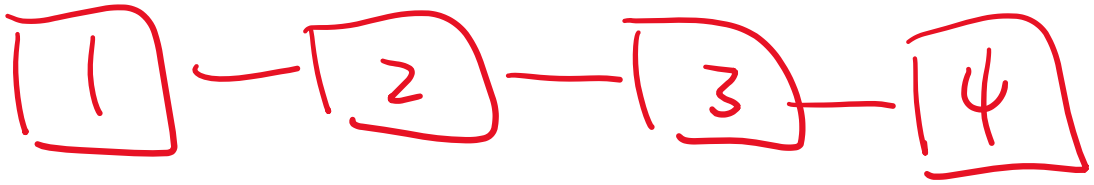
$$= 2^k T'\left(\frac{n}{2^k}\right) + O\left(\frac{n^2}{2^{k-1}}\right) + O\left(\frac{n^2}{2^{k-2}}\right) + \cdots + O\left(\frac{n^2}{2^0}\right)$$

$$= nT'(1) + O(n^2) = O(n^2)$$

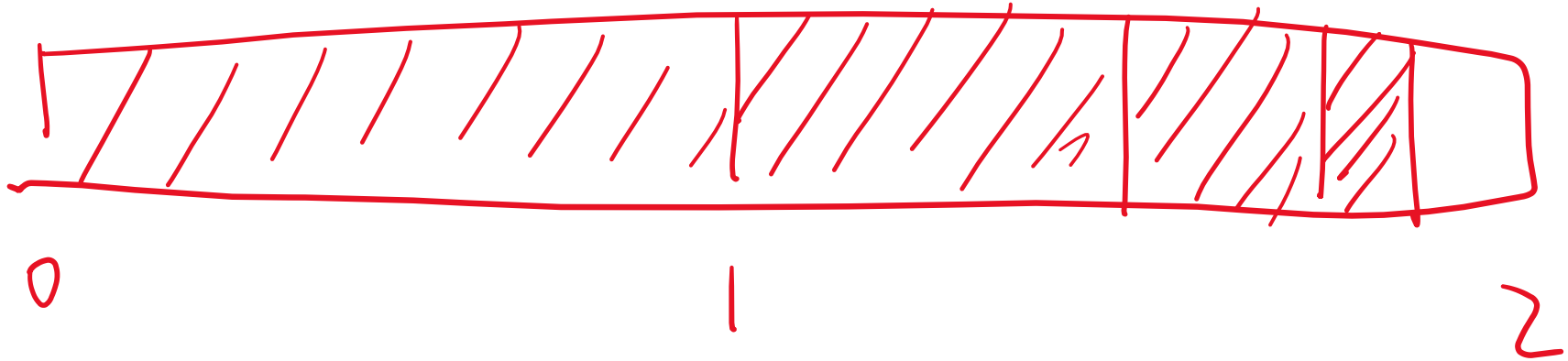

1
2
3
4
10
5
6
7
8
9
i
p

skip list

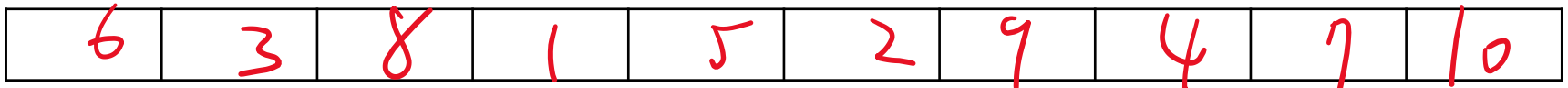




$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = 2$$



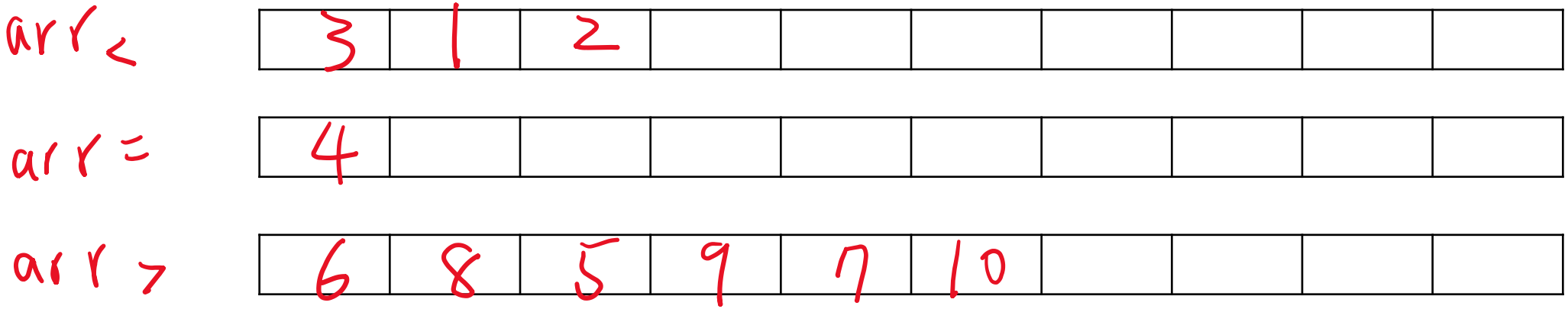
Quick Sort



pivot = 4

1° Randomly choose a pivot $O(1)$

2° Compare pivot and every element in arr



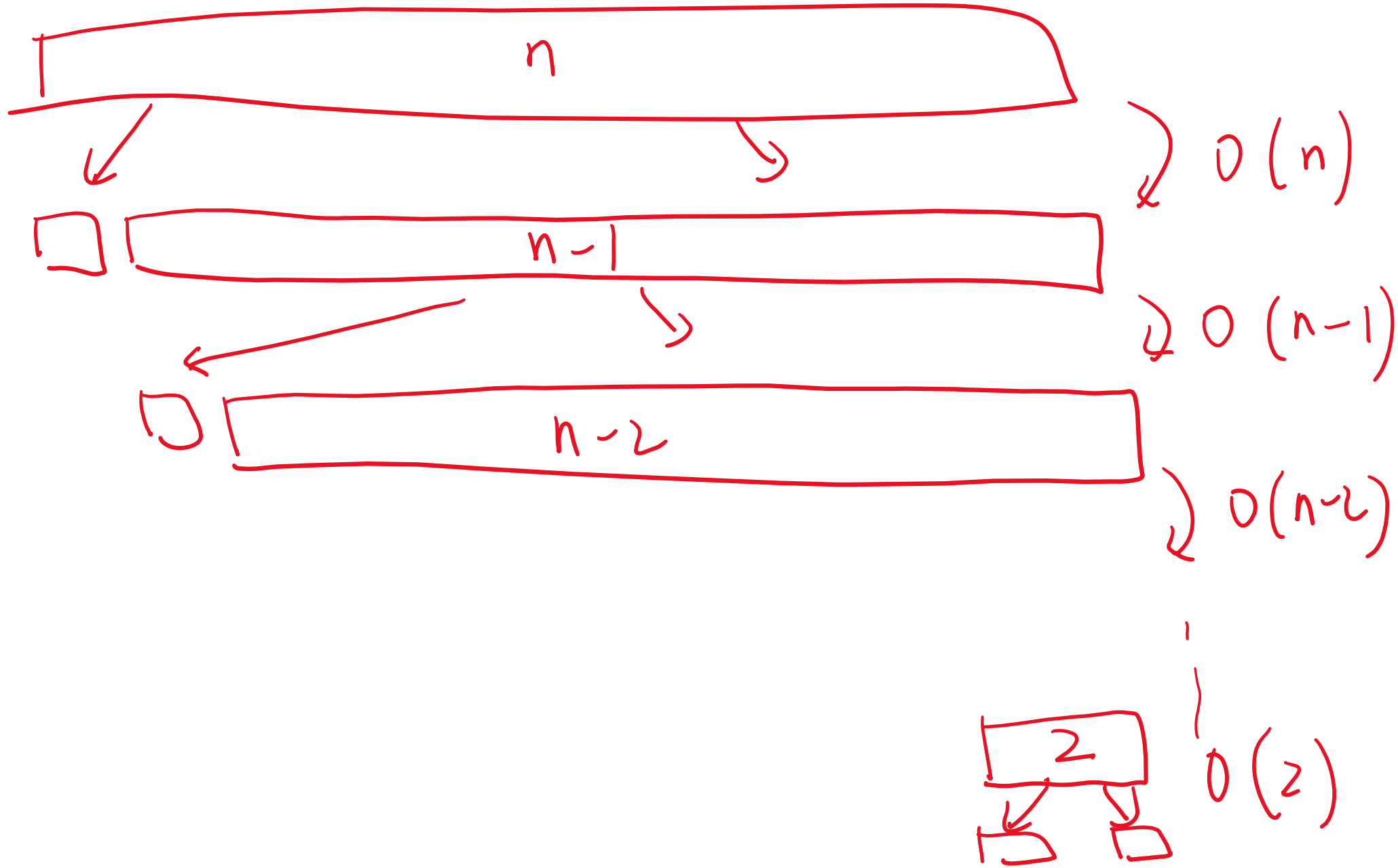
$O(n)$

3° Sort Arr< & Arr> by Quick Sort

4° Merge Arr<, Arr=, Arr>

	Quick Sort	Merge Sort
partition	difficult	easy
merge	easy	difficult

Worst Case of Quick Sort $O(n^2)$



Time Complexity of Quick Sort

=

comparisons

X: # of cards that stay at the same position.

$$E[X] = E[X_1 + X_2 + X_3 + \cdots + X_{52}] = E[X_1] + E[X_2] + \cdots + E[X_{52}]$$

$$= 52 \times \frac{1}{52} = 1$$

$X_i = 0$ if the i^{th} card changes position

$X_i = 1$ otherwise

$$X = X_1 + X_2 + \cdots + X_{52}$$

$$E[X_i] = 1 \cdot Pr[X_i = 1] + 0 \cdot Pr[X_i = 0]$$

$$= Pr[X_i = 1]$$

$$= \frac{1}{52}$$

X: # comparisons of Quick Sort

① For any 2 numbers in arr, they are compared at most once.

② $Pr[$ max and min are compared $]$ is small

$= \frac{2}{n} = Pr[$ 1st pivot = min or max $]$

(assuming all numbers are distinct)

e_i : the i^{th} smallest element in arr

$X_{ij} = 1$ if e_i & e_j are compared

$X_{ij} = 0$ otherwise

$$X = X_{1,1} + X_{1,2} + \cdots + X_{1,n}$$
$$+ \quad X_{2,2} + X_{2,3} + \cdots + X_{2,n}$$
$$+ \quad \cdots$$
$$+ X_{n,n}$$

$$Pr[X_{1,n} = 1] = \frac{2}{n}$$

$$E[X_{1,n}] = \frac{2}{n}$$

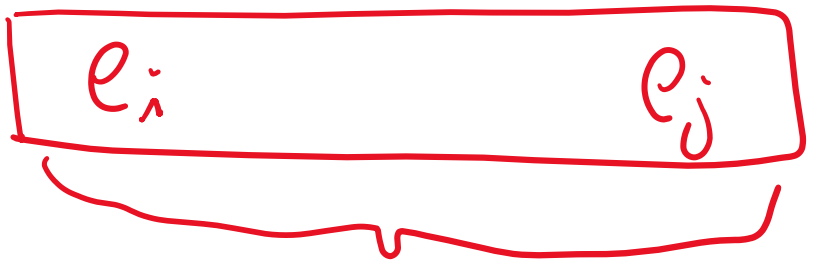
$$Pr[X_{i,i+1} = 1] = 1$$

$$E[X_{i,i+1}] = 1$$

Assume $i < j$

$$\Pr[X_{i,j} = 1] = \Pr[\text{pivot} = e_i \text{ or } e_j \text{ in arr}'] = \frac{2}{m} \leq \frac{2}{j-i+1}$$

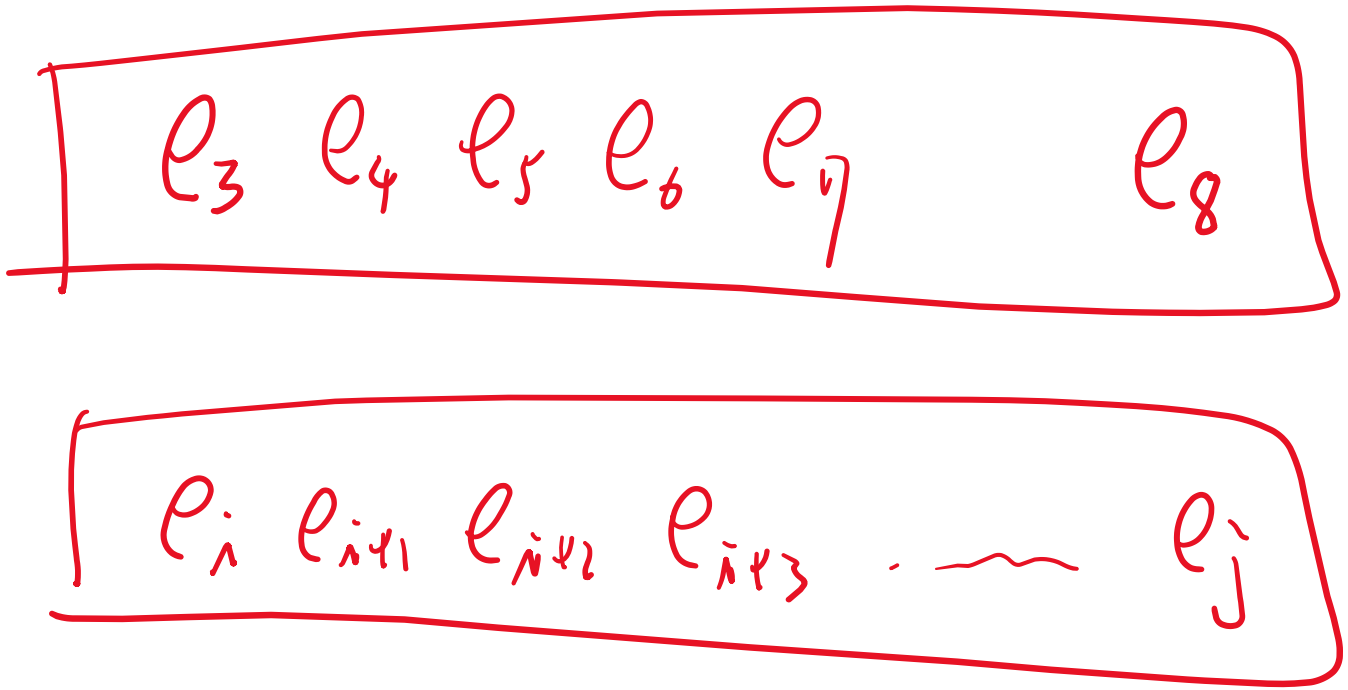
Consider the last array arr′ that contains both e_i & e_j

arr′: 

$m \geq j-i+1$

$$E[X_{i,j}] \leq \frac{1}{j-i+1}$$

distance between e_i & e_j


e_3 e_4 e_5 e_6 e_7 e_8
e_i e_{i+1} e_{i+2} e_{i+3} e_j

$$E[X] = E[X_{1,1}] + E[X_{1,2}] + E[X_{1,3}] + E[X_{1,4}] + \cdots + E[X_{1,n}]$$

$$+ E[X_{2,2}] + E[X_{2,3}] + \cdots + E[X_{2,n}]$$

$$+ \ddots$$

$$\leq \frac{2}{1} \quad \leq \frac{2}{2} \quad \leq \frac{2}{3} \quad \leq \frac{2}{4} \quad + \cdots \quad + \frac{2}{n}$$

$$\leq \frac{2}{1} \quad \leq \frac{2}{2} \quad + \cdots \quad + \frac{2}{n-1}$$

$$E[X] \le \frac{2}{1} + \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n} = O(\log n)$$

$$+ \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n-1} = O(\log n)$$

$$+ \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n-1} = O(\log n)$$

$$+ \ddots$$

$$+ \frac{2}{2} \quad O(\log n)$$

$$\text{Total} = O(n \log n)$$

O(1) - Additional Space Partition

Lomuto's Partition

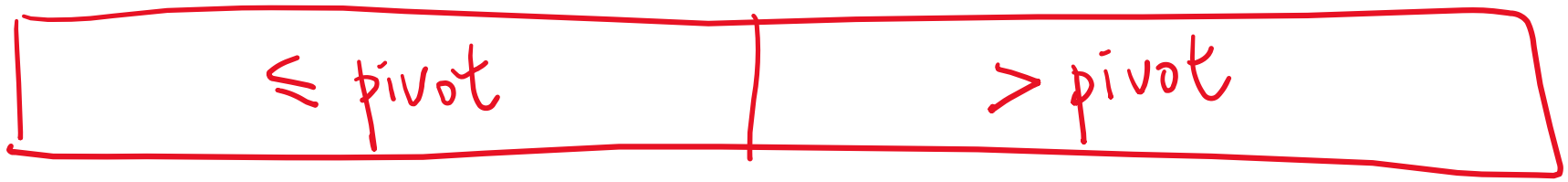
pivot = 4

3	1	5	7	2	8	10	4	6	9

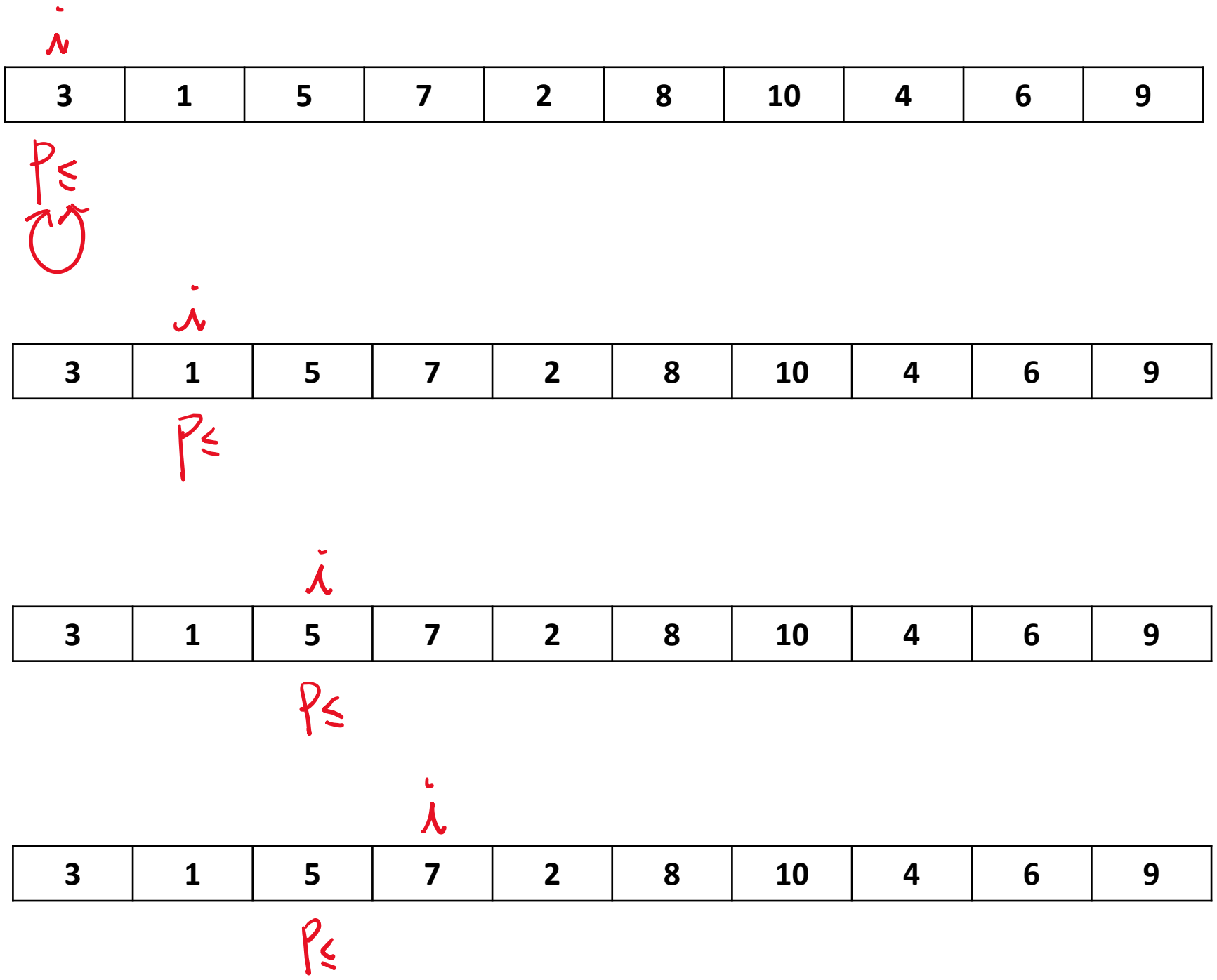
$P_{\leq}$

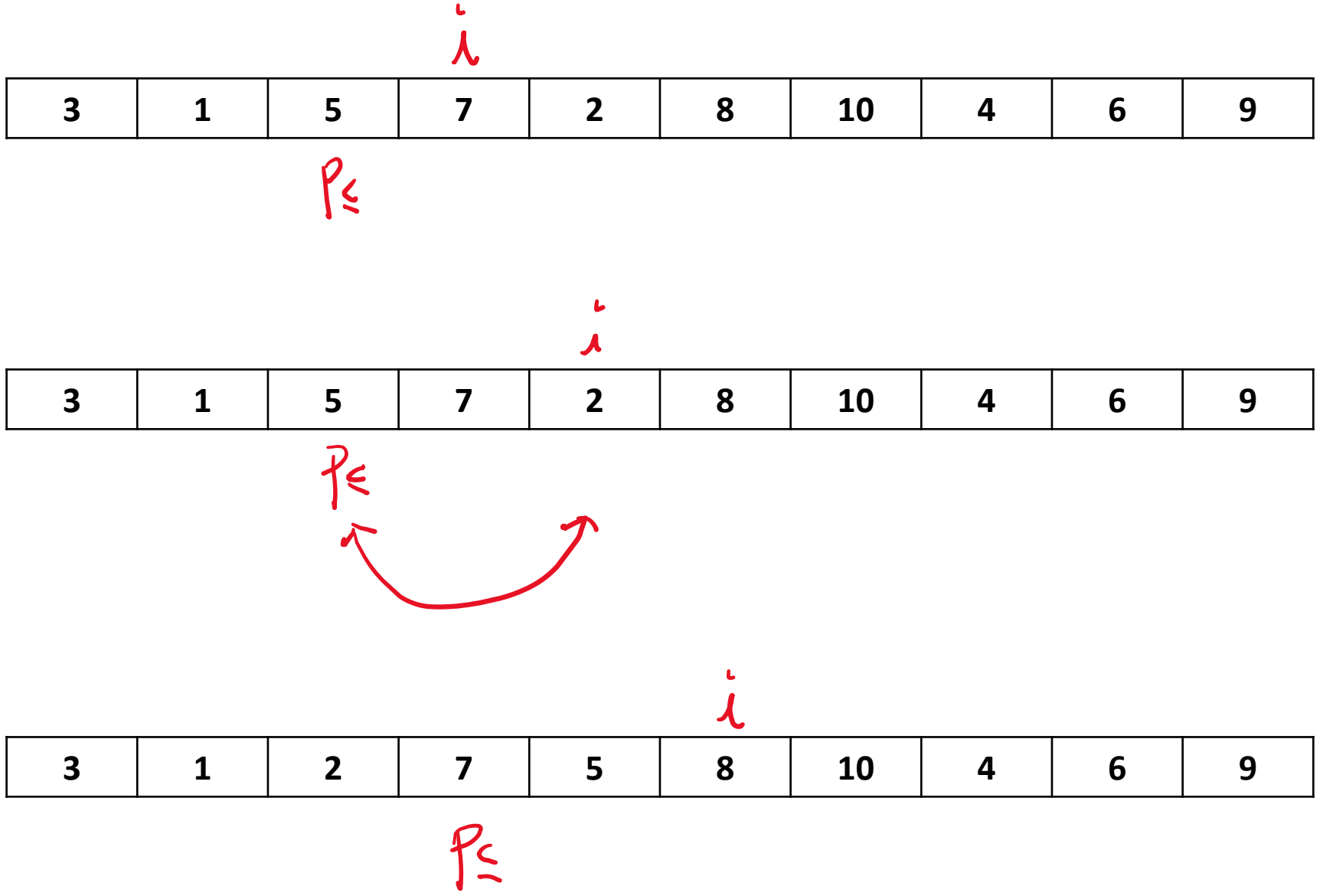
Hint:
① $P_{\leq}$: position to store the next ≤ pivot data
② $i : 0 \rightarrow n-1$

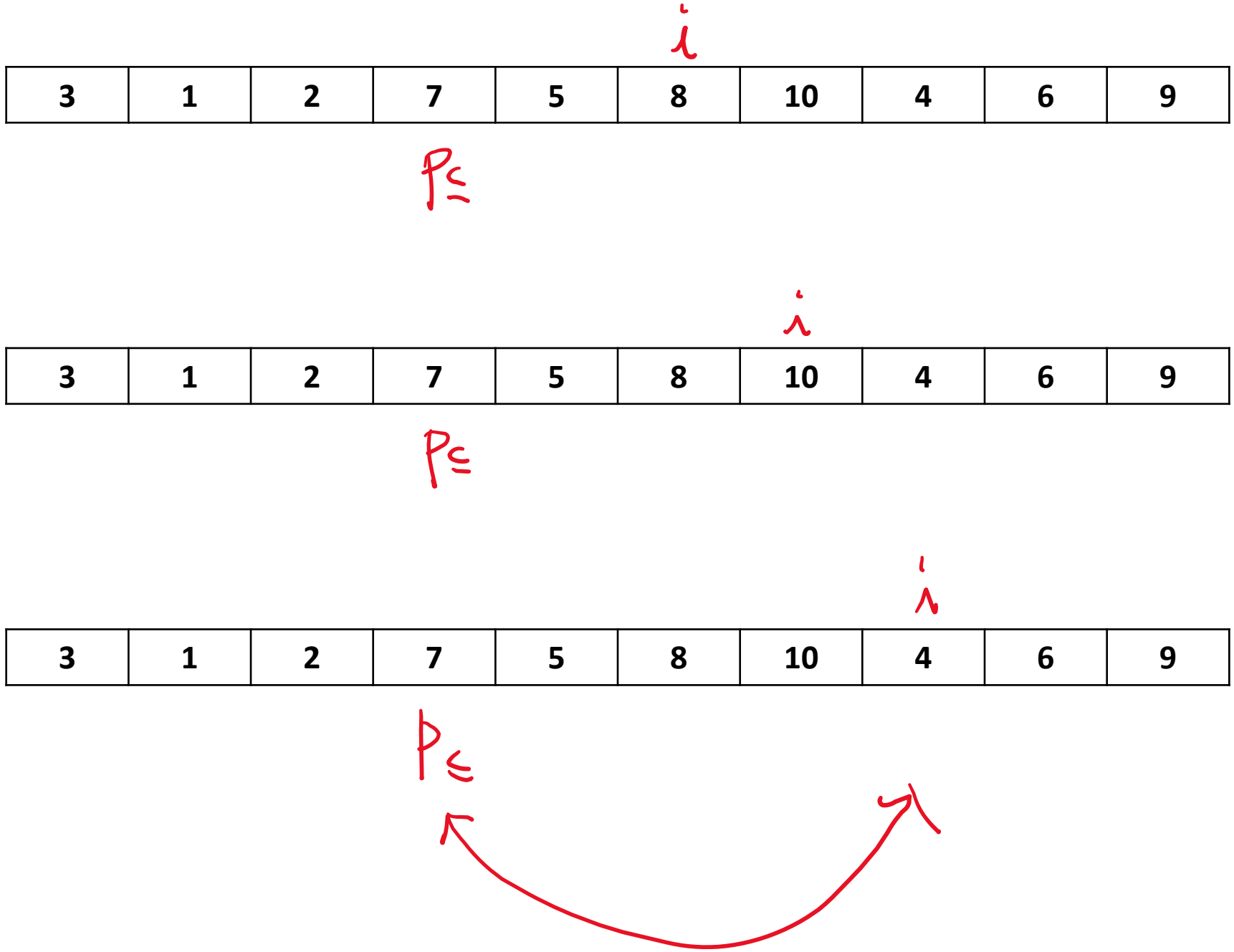
Goal:

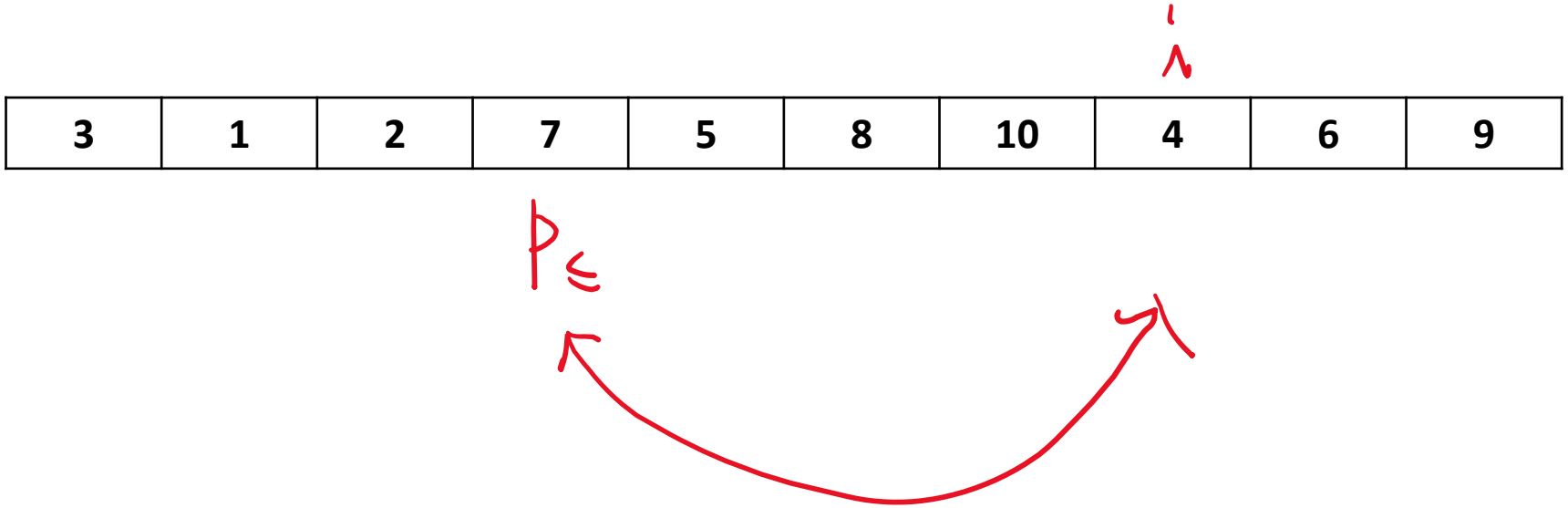


pivot = 4


i
3 1 5 7 2 8 10 4 6 9
P≤
i
3 1 5 7 2 8 10 4 6 9
P≤
i
3 1 5 7 2 8 10 4 6 9
P≤
i
3 1 5 7 2 8 10 4 6 9
P≤

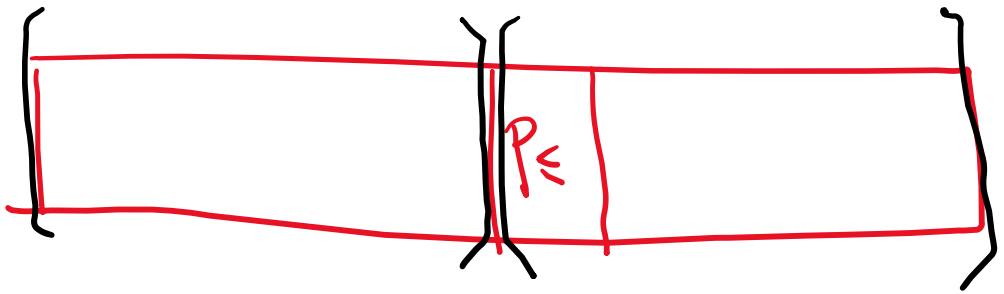

i
3 1 5 7 2 8 10 4 6 9
p≤
i
3 1 5 7 2 8 10 4 6 9
p≤
i
3 1 2 7 5 8 10 4 6 9
p≤


i
3 1 2 7 5 8 10 4 6 9
P≤
i
3 1 2 7 5 8 10 4 6 9
P≤
i
3 1 2 7 5 8 10 4 6 9
P≤



Arr[0 ~ Pe-1] : ≤ pivot

Arr[Pe ~ n-1] : > pivot



1	1	1	1	1	1	1	1	1	1

i pivot = 1

P<

||

original

array

Time Complexity of Quick Sort = ∞

when all inputs are identical.

Arr

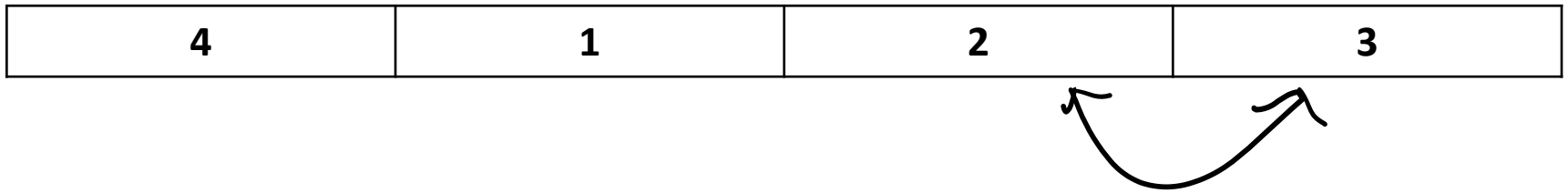
1	1	1	1	1	0	0	0	0	0

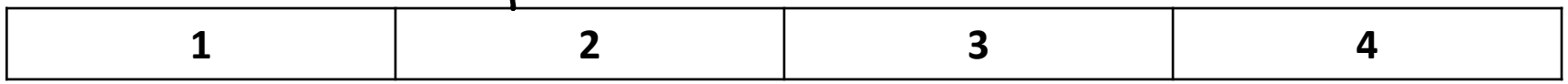
if pivot = 1 ⇒ partitioned into Arr and empty array

if pivot = 0 ⇒ partitioned into All-0 array and All-1 array

Hint : ③ swap (pivot , arr [n-1])

pivot = 2



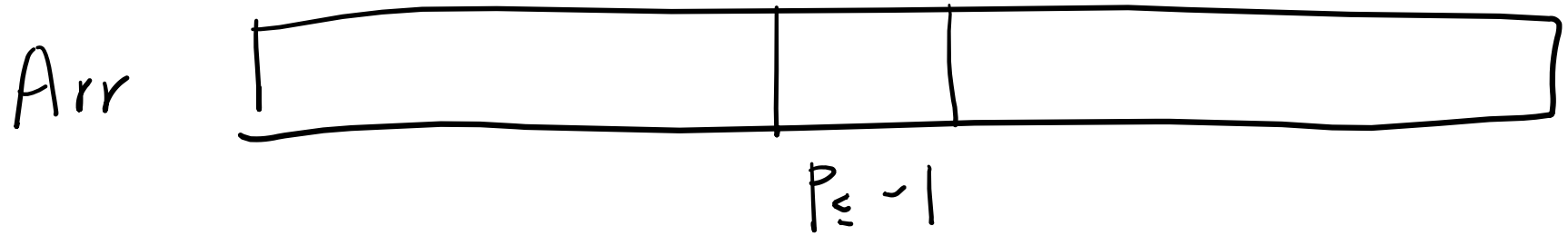


pe - 1

i

pe

Arr [pe - 1] = pivot



Arr is partitioned into

Arr[0 ~ pe-2] : ≤ pivot

Arr[pe-1] : = pivot

Arr[pe+1 ~ n-1] : > pivot

1° Swap (pivot, Arr[n-1]) $\Theta(n)$

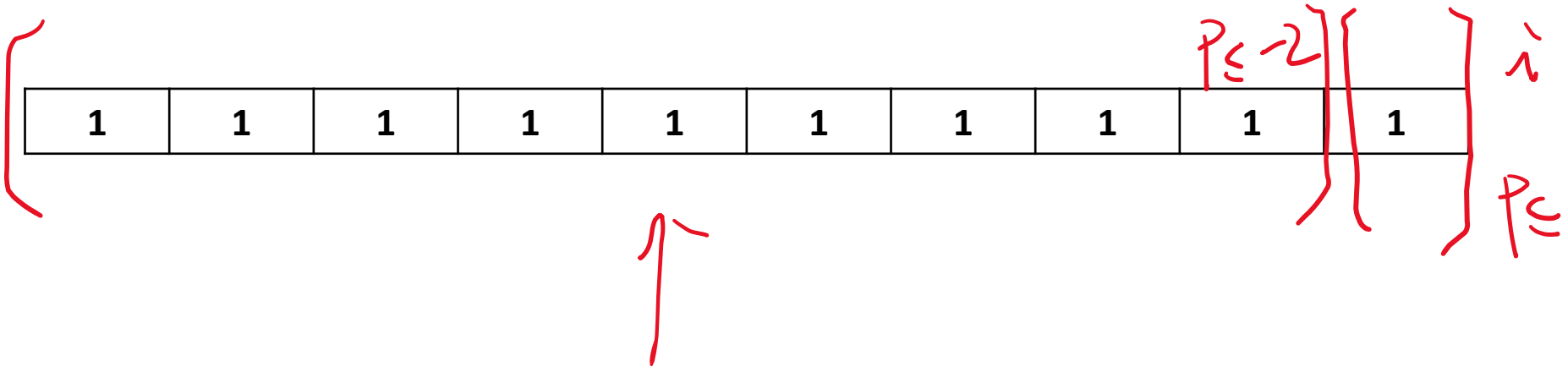
2° for (i=0; i<n; i++){

if (Arr[i] ≤ pivot) {

swap (Arr[i], Arr[p≤++]);

}

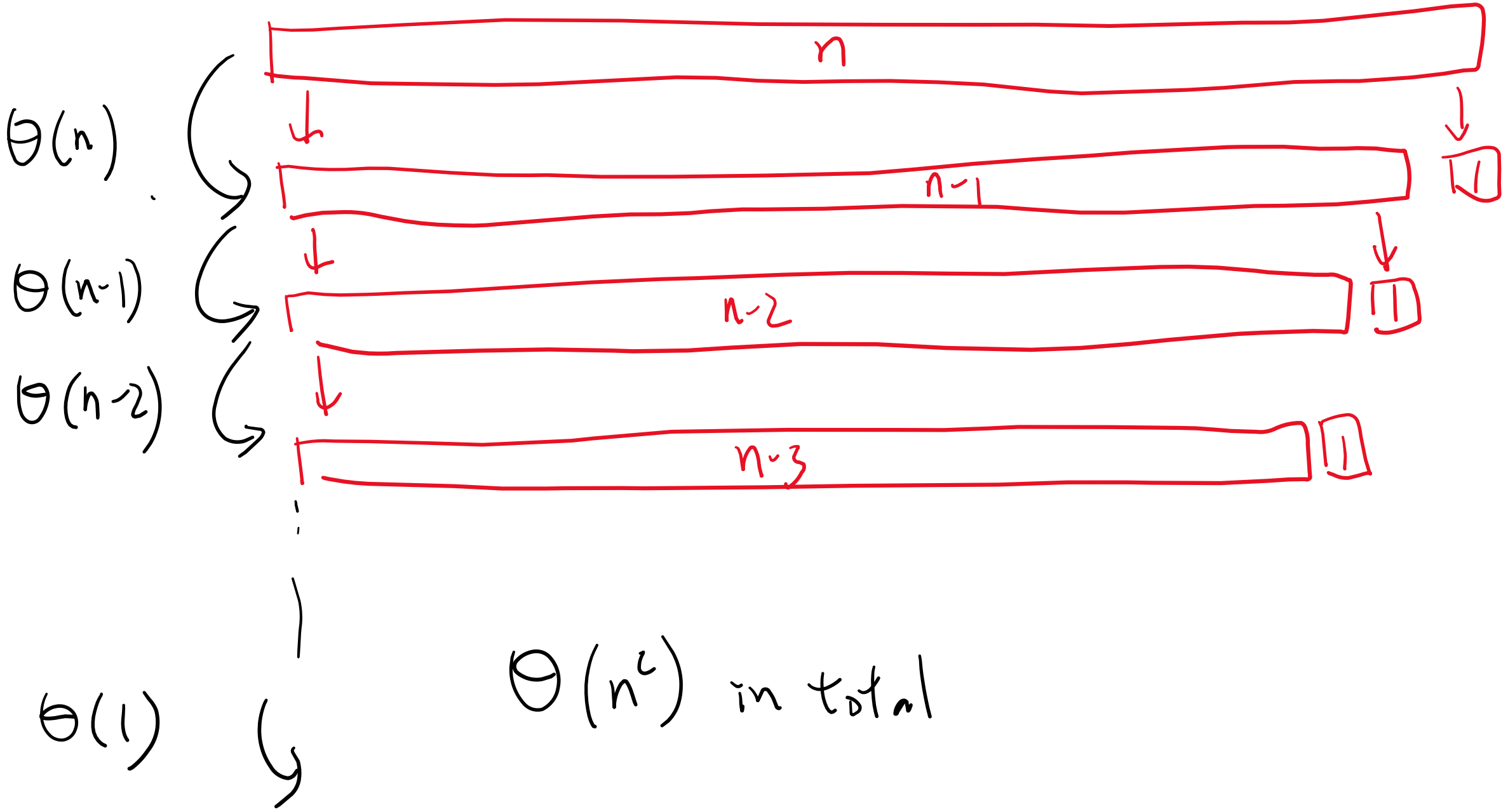
}



n-1 ← 1° Arr[0 ~ p<=-2] = Arr[0 ~ n-2]

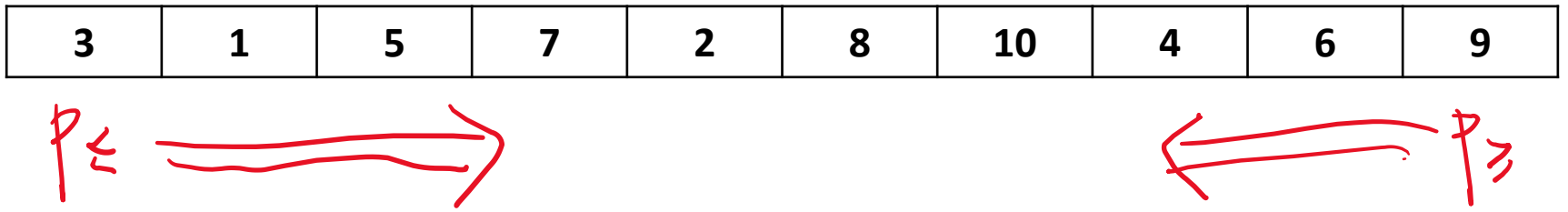
1 ← 2° Arr[p<=-1] = Arr[n-1]

0 ← 3° empty array


n
Θ(n)
n-1
1
Θ(n-1)
n-2
1
Θ(n-2)
n-3
1
Θ(1)
Θ(n²) in total

Hoare Partition

pivot=4



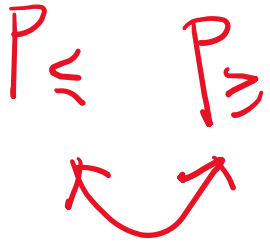
Hint: ① P≤ : position to store the next ≤pivot data

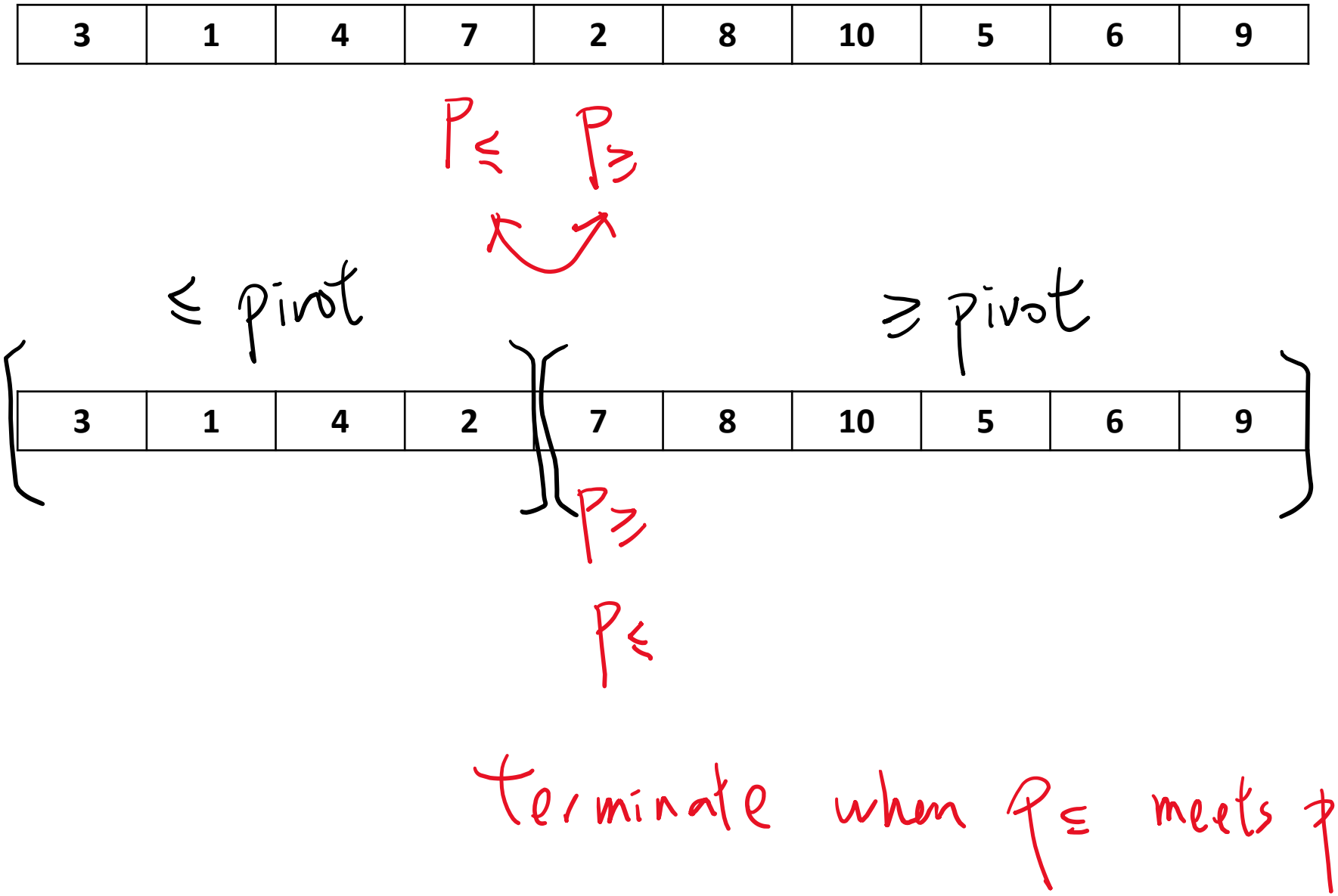
② P≥ : position to store the next ≥pivot data

pivot = 4

3	1	5	7	2	8	10	4	6	9

3	1	4	7	2	8	10	5	6	9




3 1 4 7 2 8 10 5 6 9
P≤ P≥
≤ pivot
≥ pivot
3 1 4 2 7 8 10 5 6 9
P≥
P≤
terminate when P≤ meets P≥

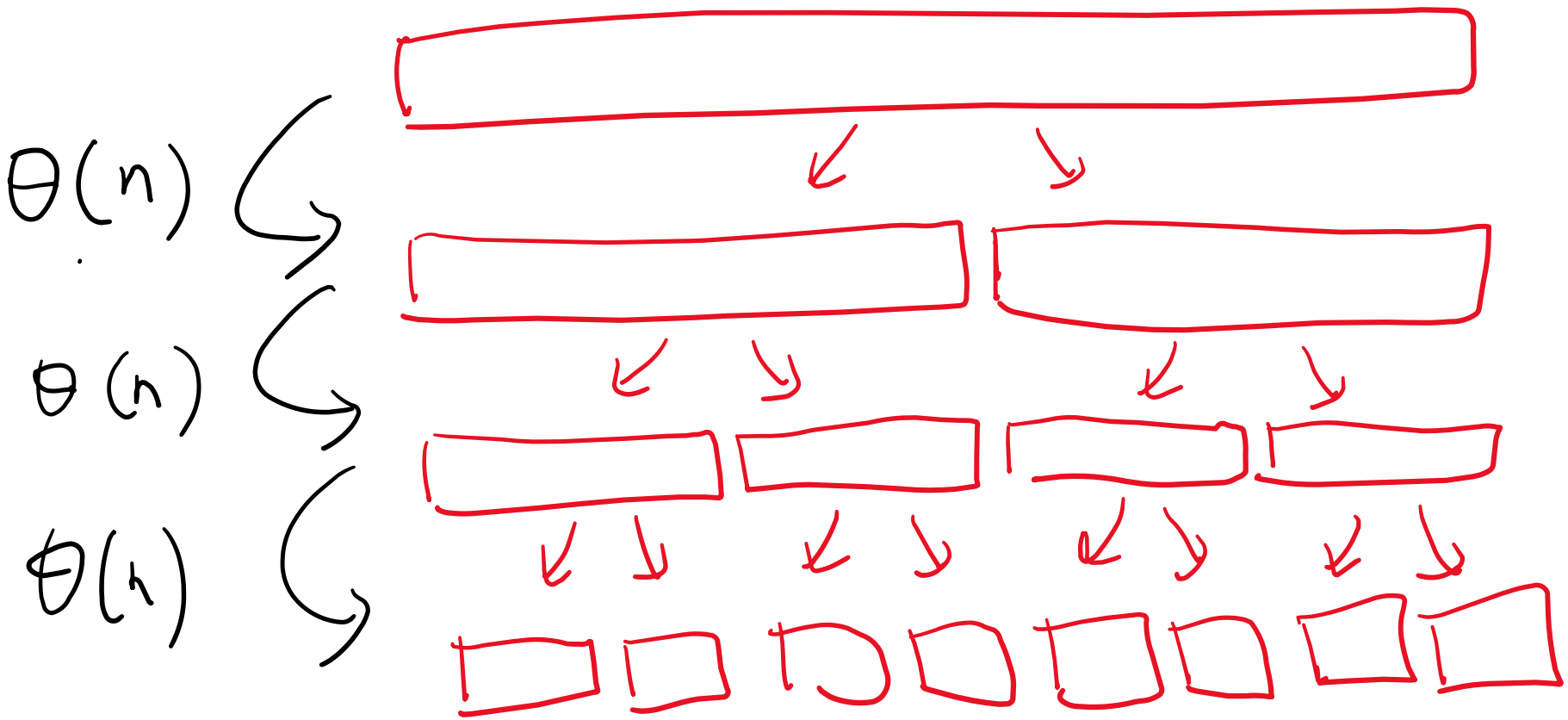
pivot = 1

1	1	1	1	1	1	1	1	1	1

$P_{\leq}$ $P_{\geq}$

$P_{\leq}$ and $P_{\geq}$ meet at $arr[\frac{n}{2}]$

Arr is partitioned into 2 size $-\frac{n}{2}$ arrays



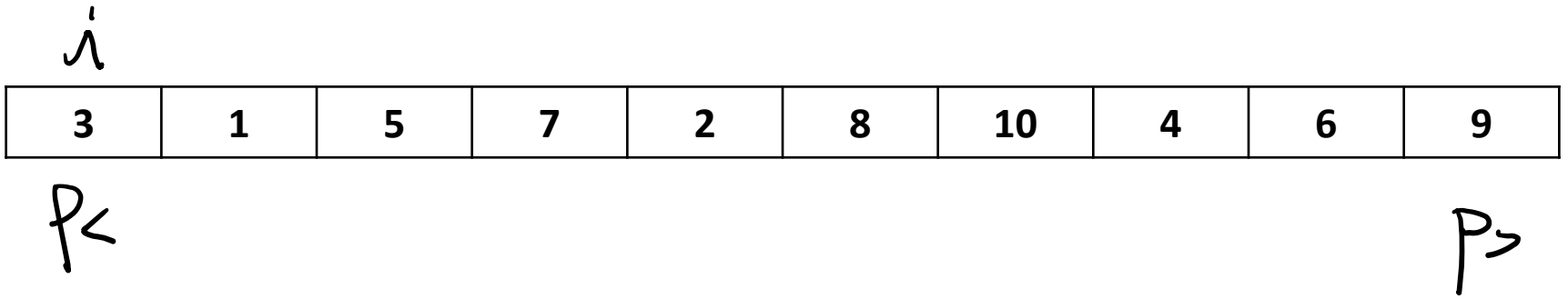
Time complexity = $\Theta(n \log n)$ when applying quick sort to an array where all inputs are the same (with Hoare Partition)

3-Way Partition (Dutch Flag Partition)

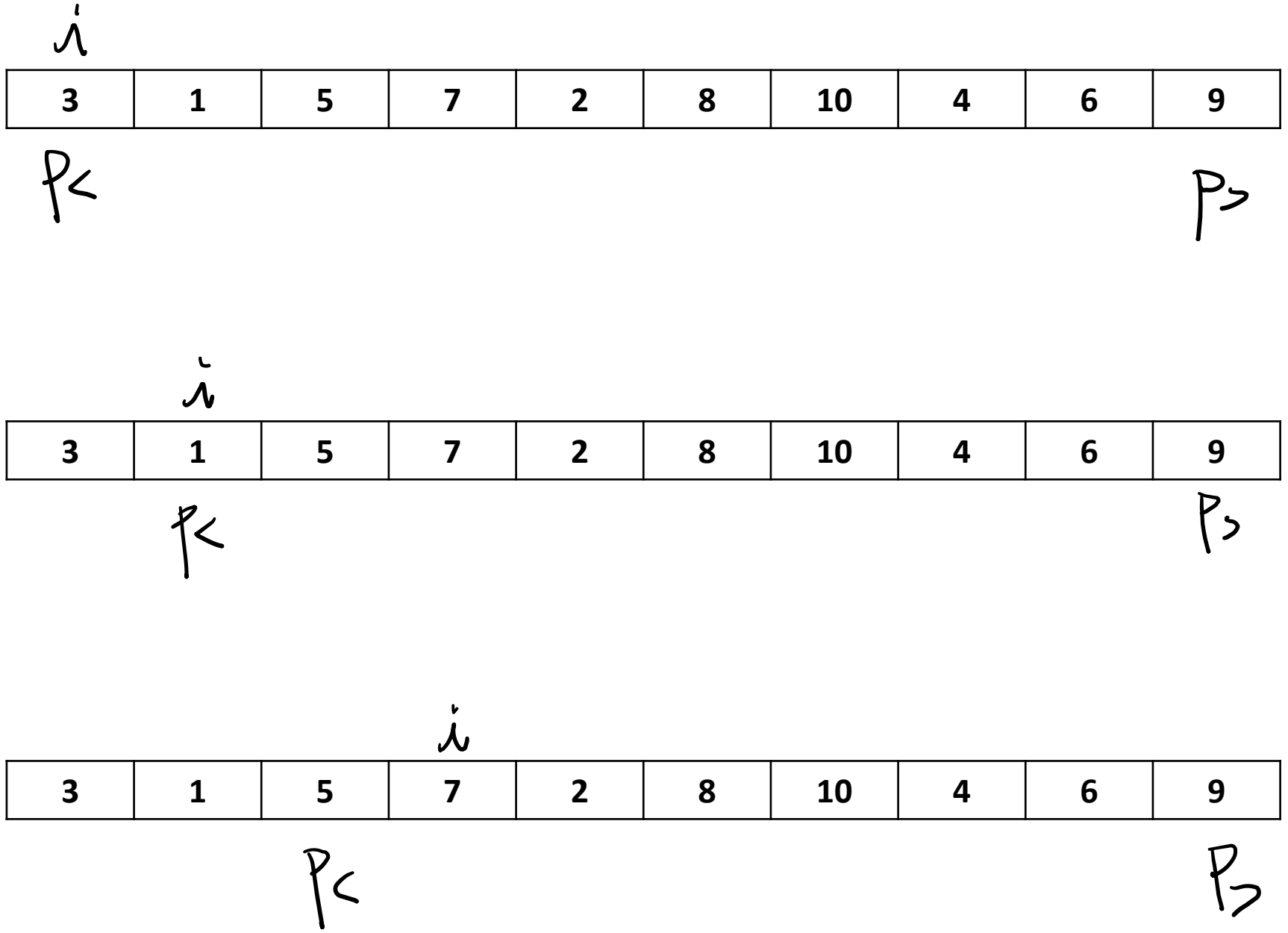
Hint: ① p<: position to store the next < pivot data

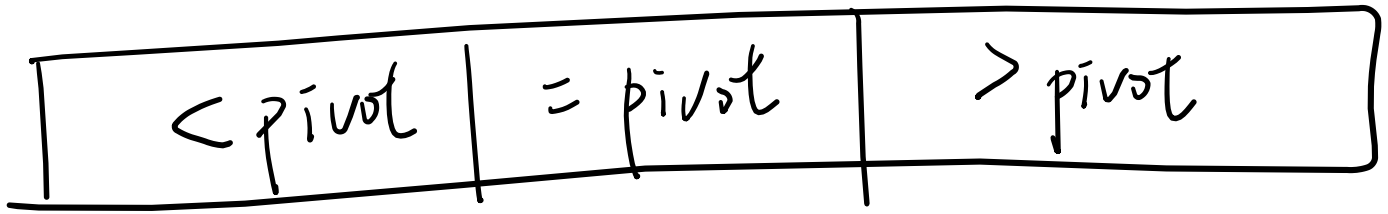
② p>: position to store the next > pivot data

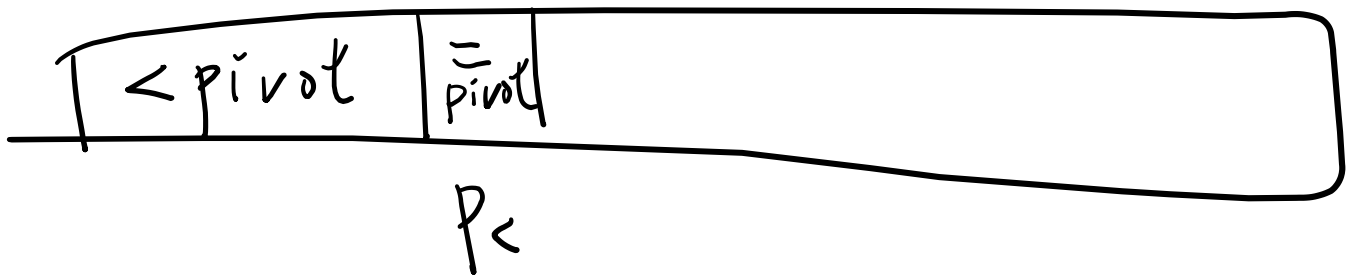
③ i : 0 ~ p>



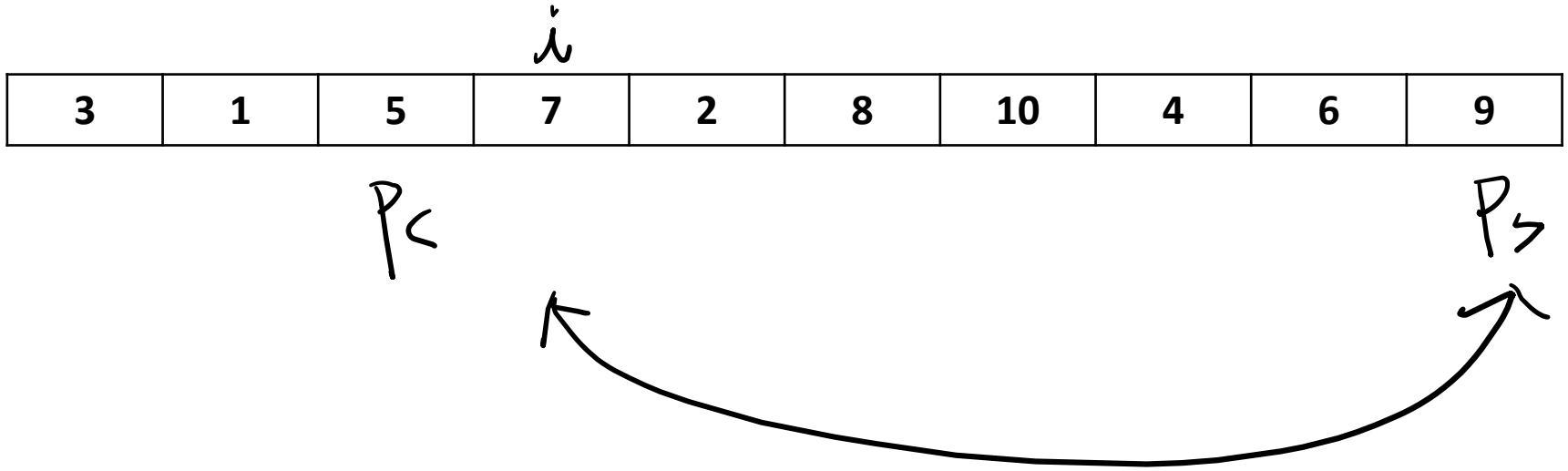
pivot = 5

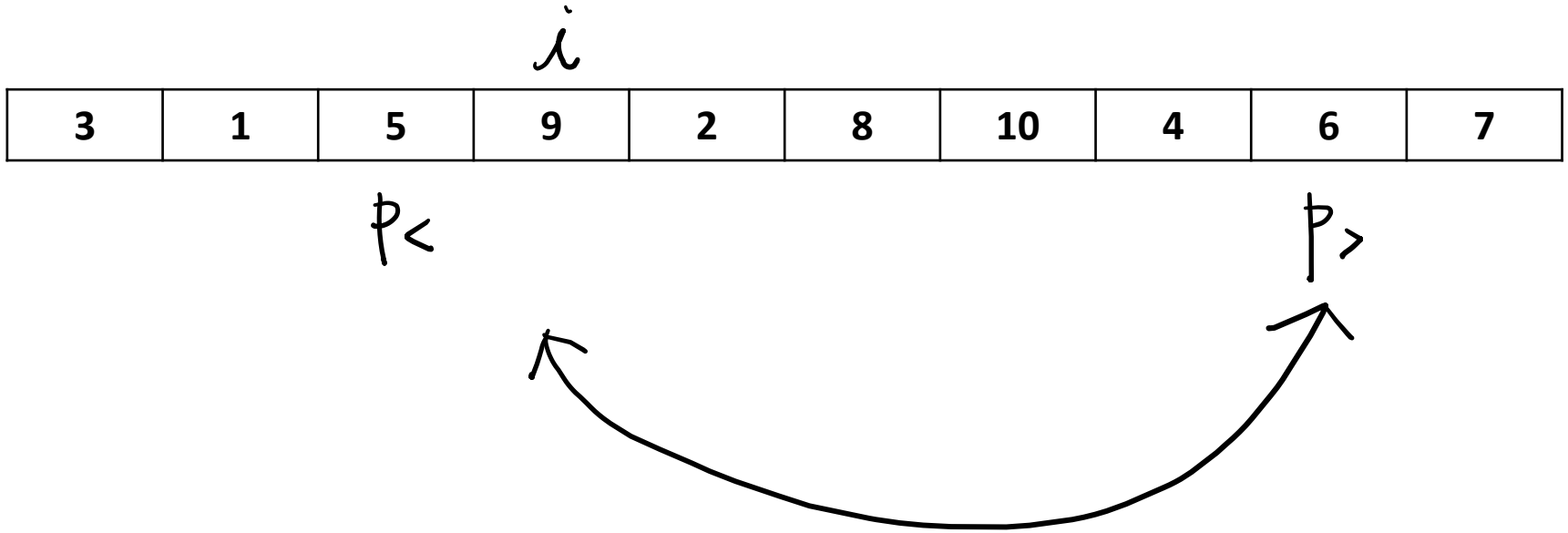


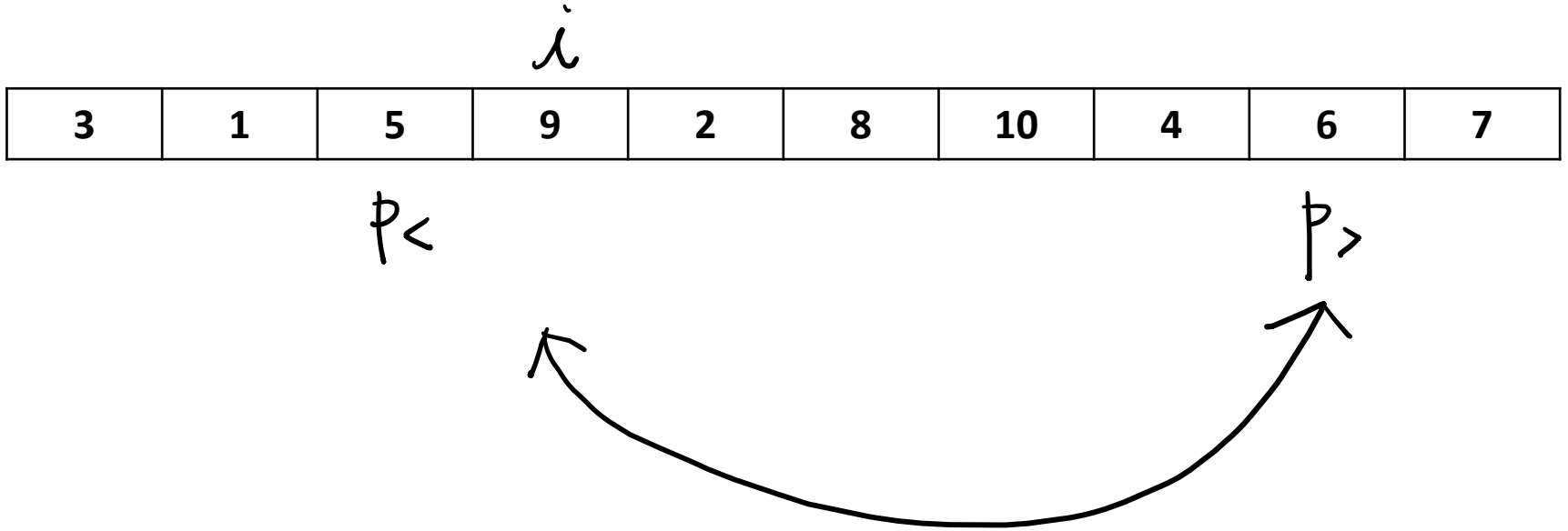


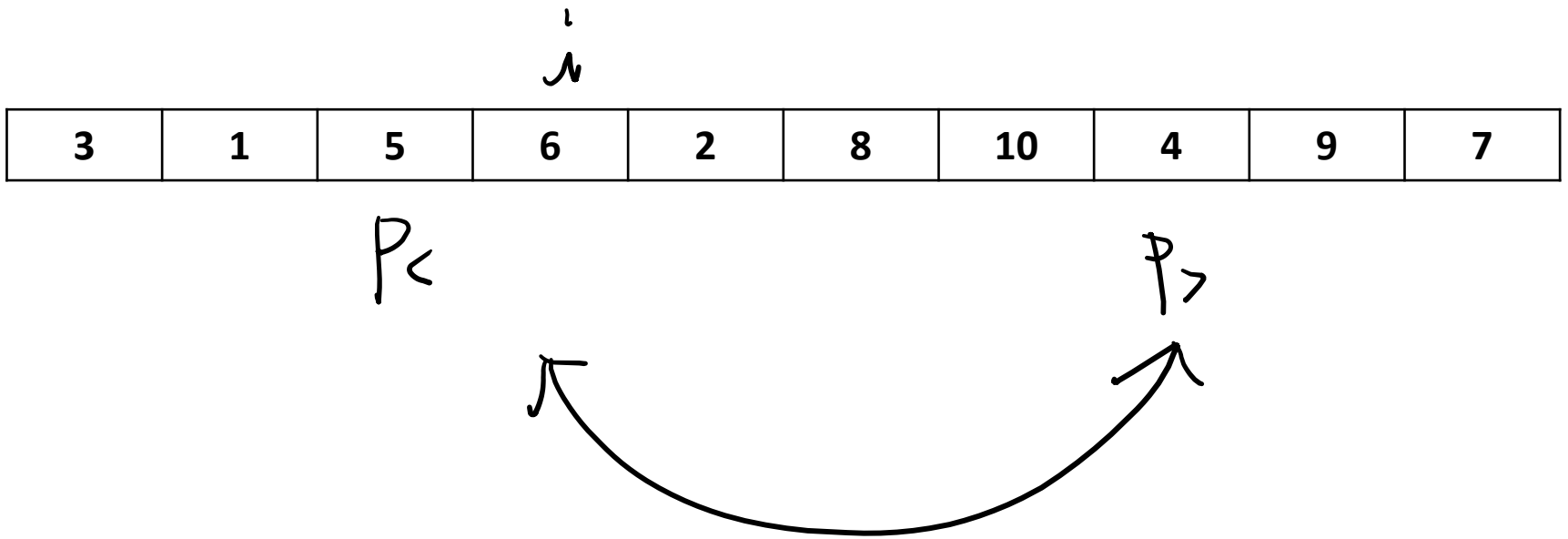


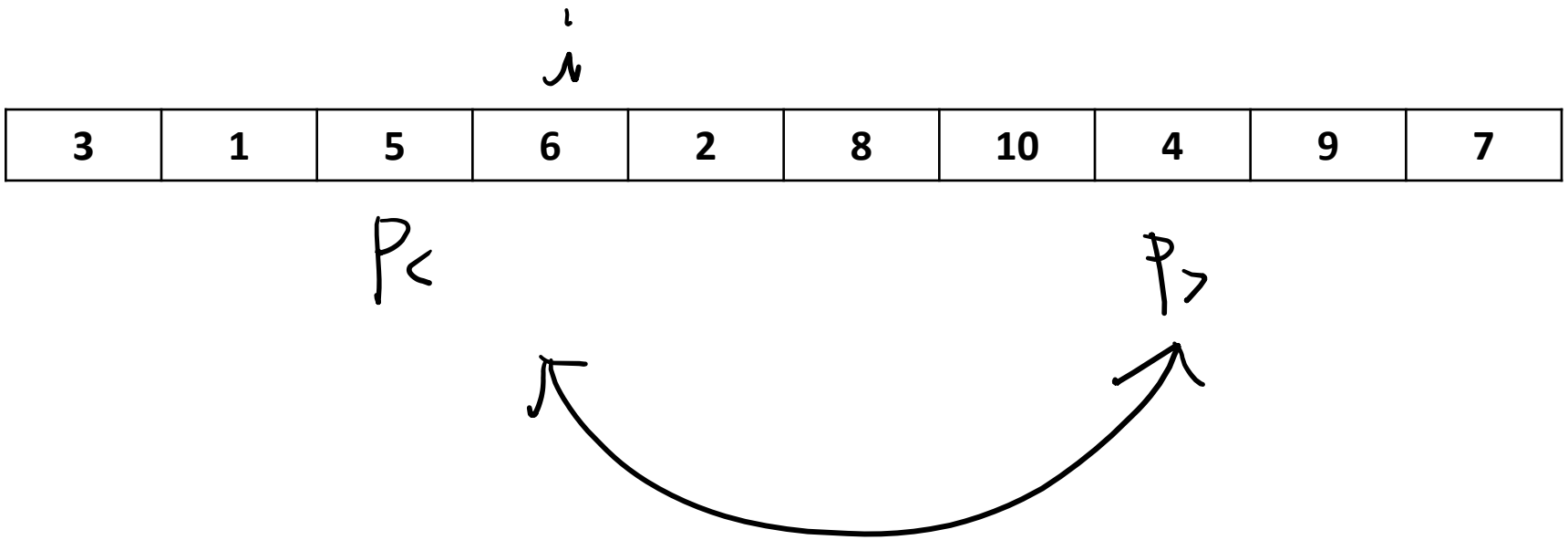
pivot = arr [rand() % n]

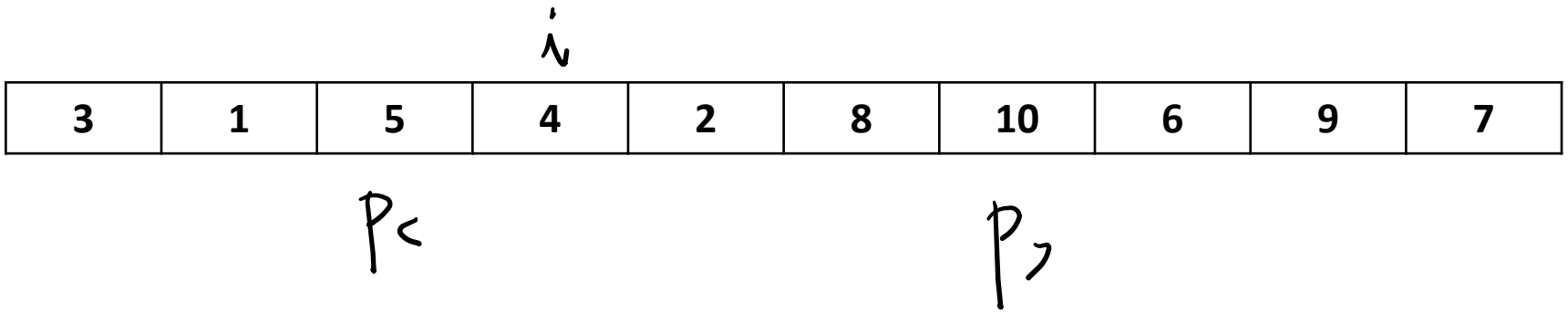

i
3
1
5
7
2
8
10
4
6
9
P<
P>


i
3
1
5
9
2
8
10
4
6
7
P<
P>

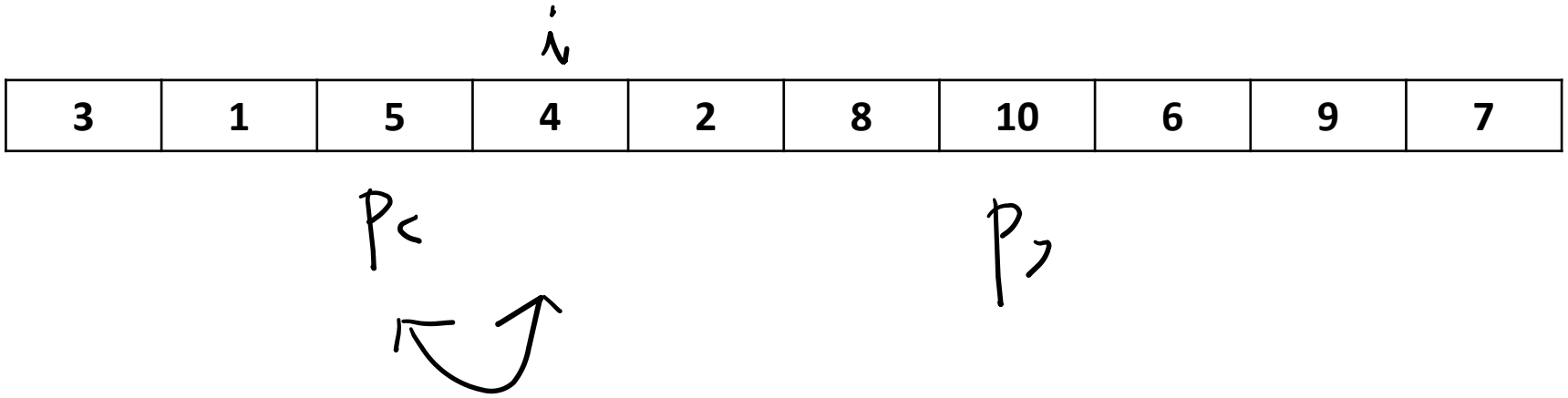

i
3
1
5
9
2
8
10
4
6
7
P<
P>

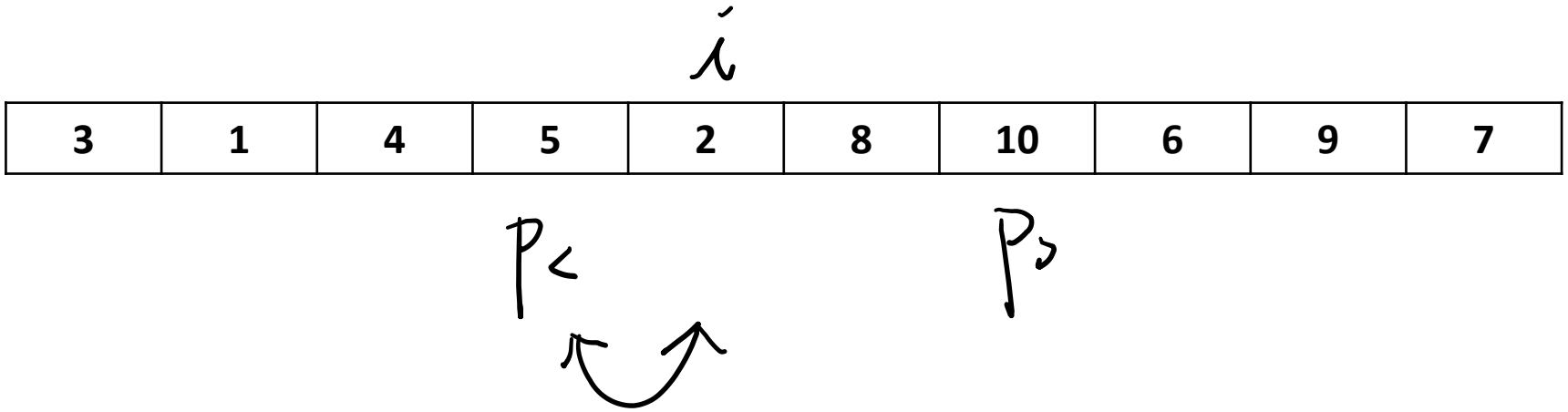

i
3
1
5
6
2
8
10
4
9
7
P<
P>

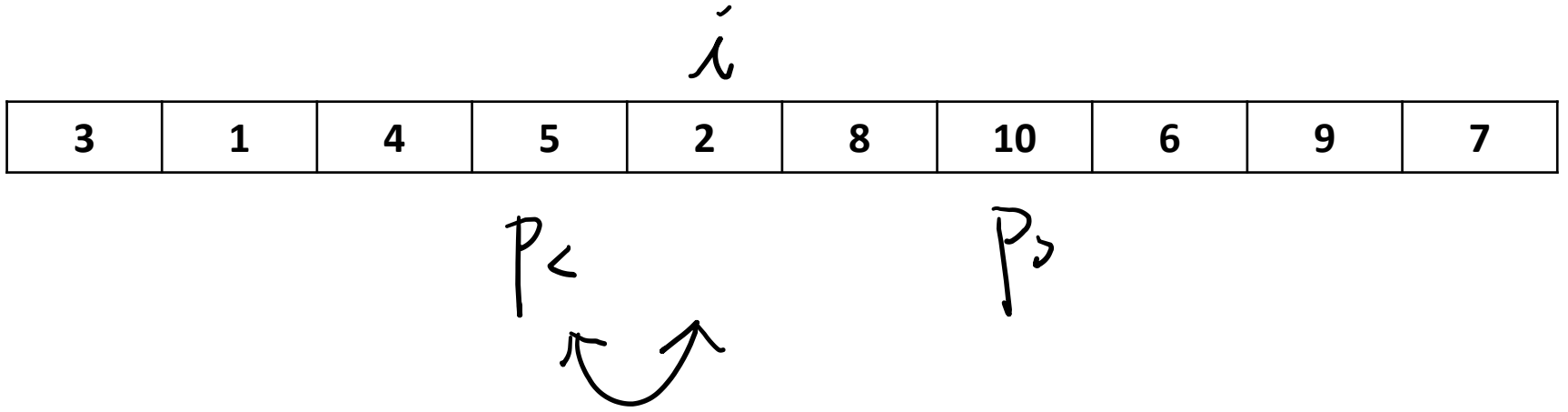

i
3
1
5
6
2
8
10
4
9
7
P<
P>

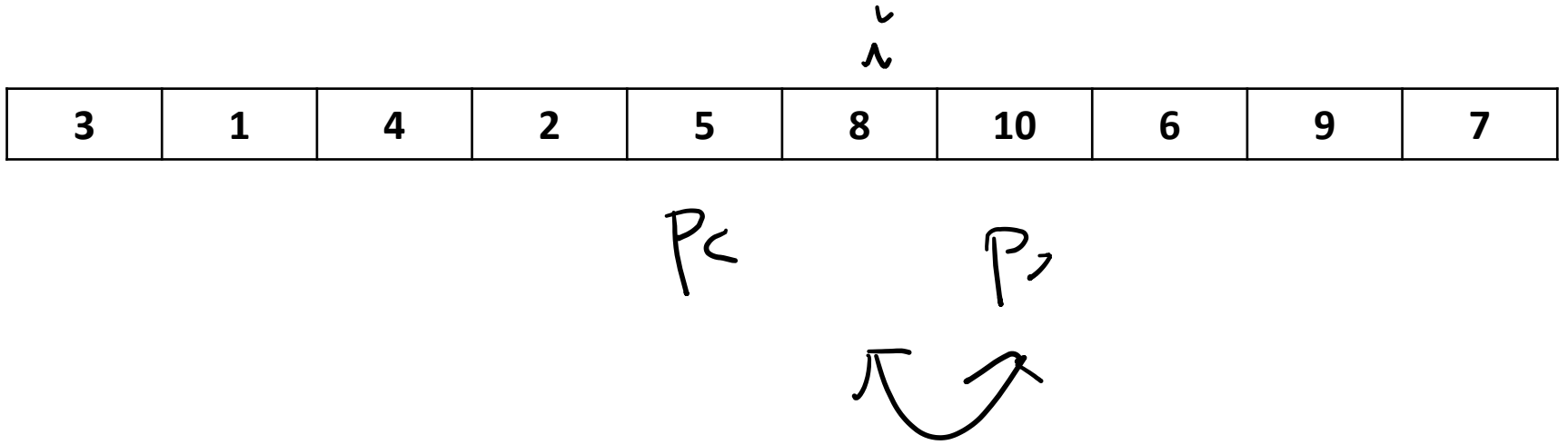

i
3
1
5
4
2
8
10
6
9
7
P<
P>

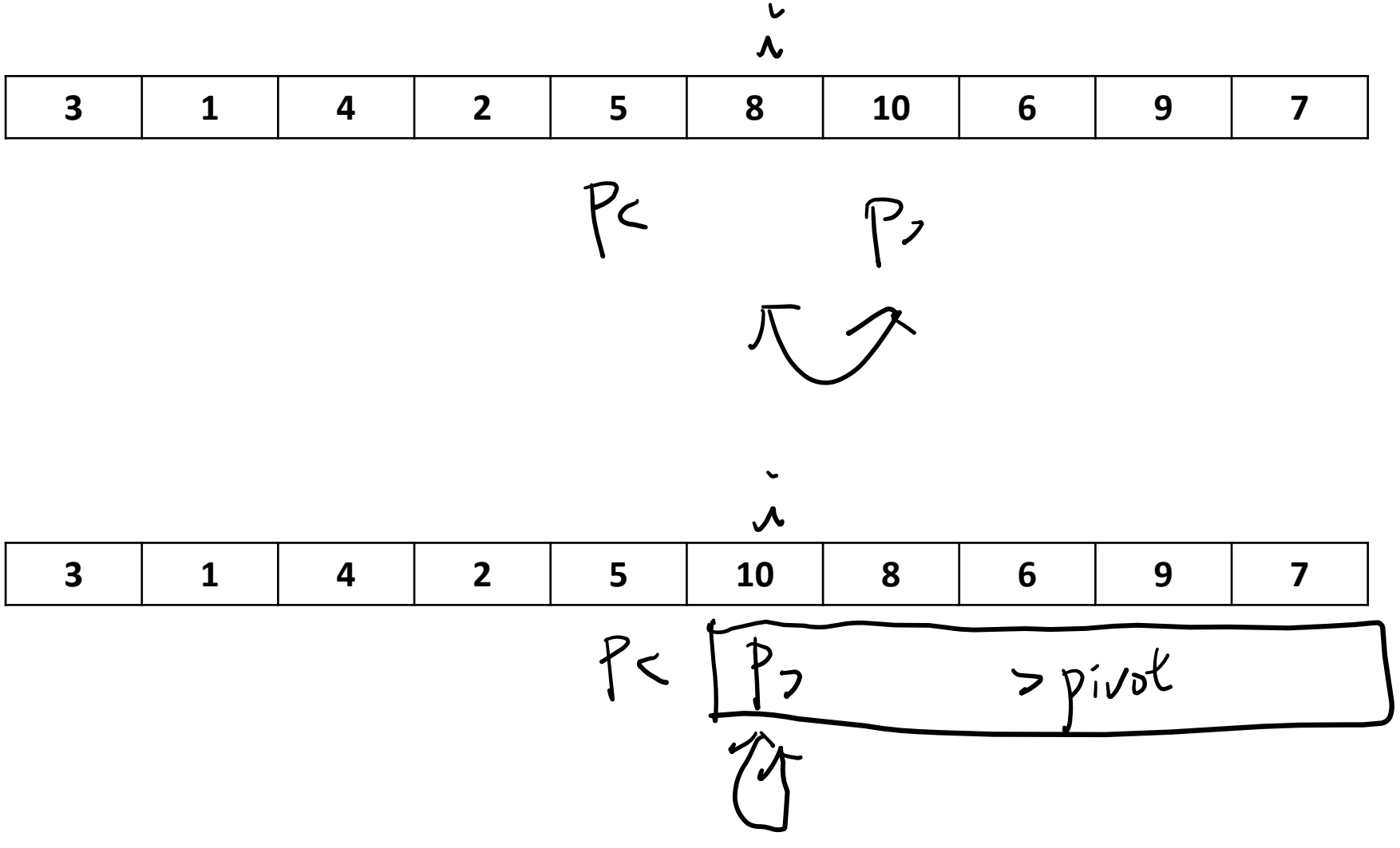
pivot = 5

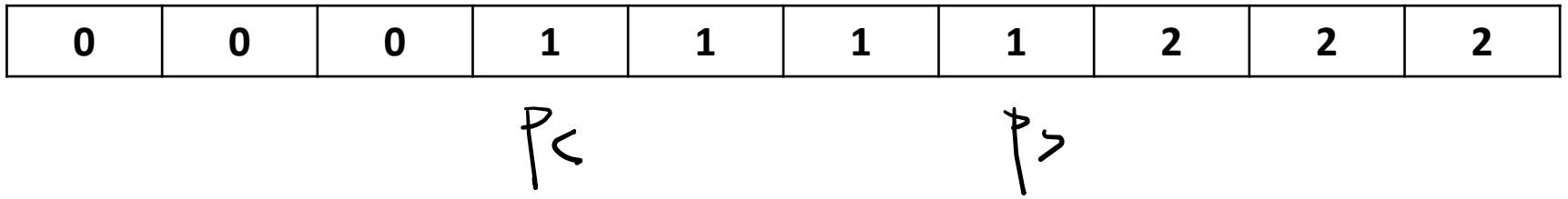





i
3
1
4
5
2
8
10
6
9
7
P<
P>


i
3
1
4
2
5
8
10
6
9
7
P<
P>


3
1
4
2
5
8
10
6
9
7
P<
P>
3
1
4
2
5
10
8
6
9
7
P<
P>
> pivot



Arr is partitioned into

① Arr [0 ~ p< - 1] : < pivot

② Arr [p< ~ p>] : = pivot

③ Arr [p> + 1 ~ n-1] : > pivot

Case 1: Arr[i] < pivot

swap (Arr[p++], Arr[i++])

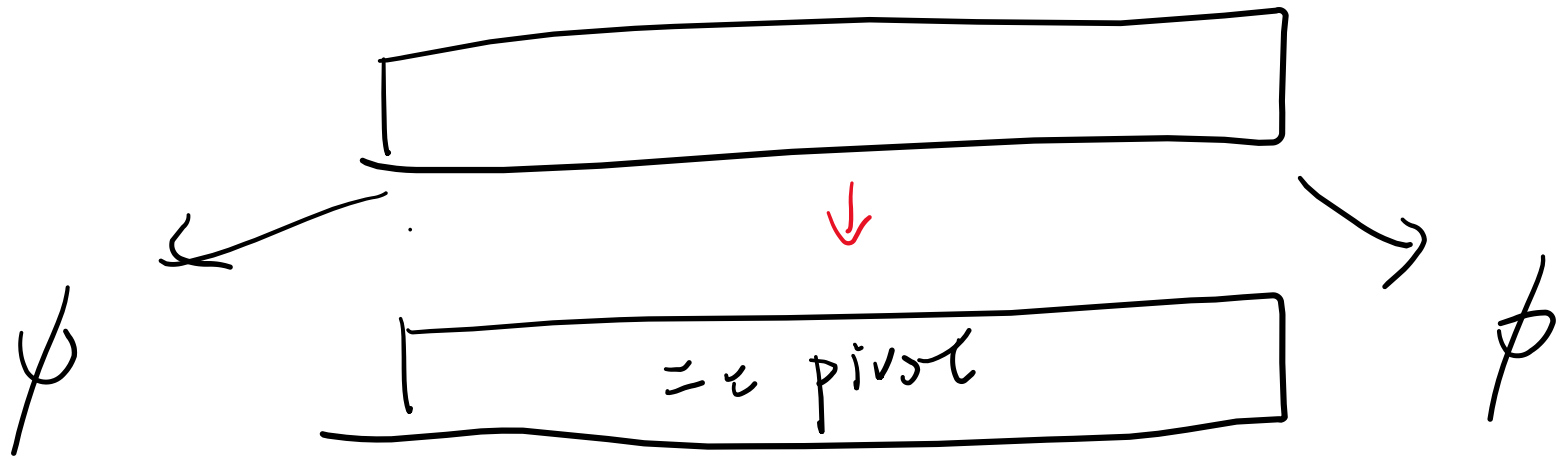
Case 2: Arr[i] = pivot

i++

Case 3: Arr[i] > pivot

swap (Arr[j--], arr[i])

Time complexity = $\Theta(n)$ when applying quick sort to an array where all inputs are the same (with 3-Way Partition)

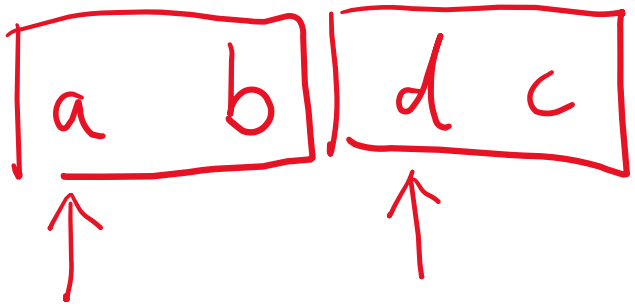


No more sorting is needed

Comparison-Based Sorting Algorithm

a b c d

Sort an array by comparing 2 numbers.



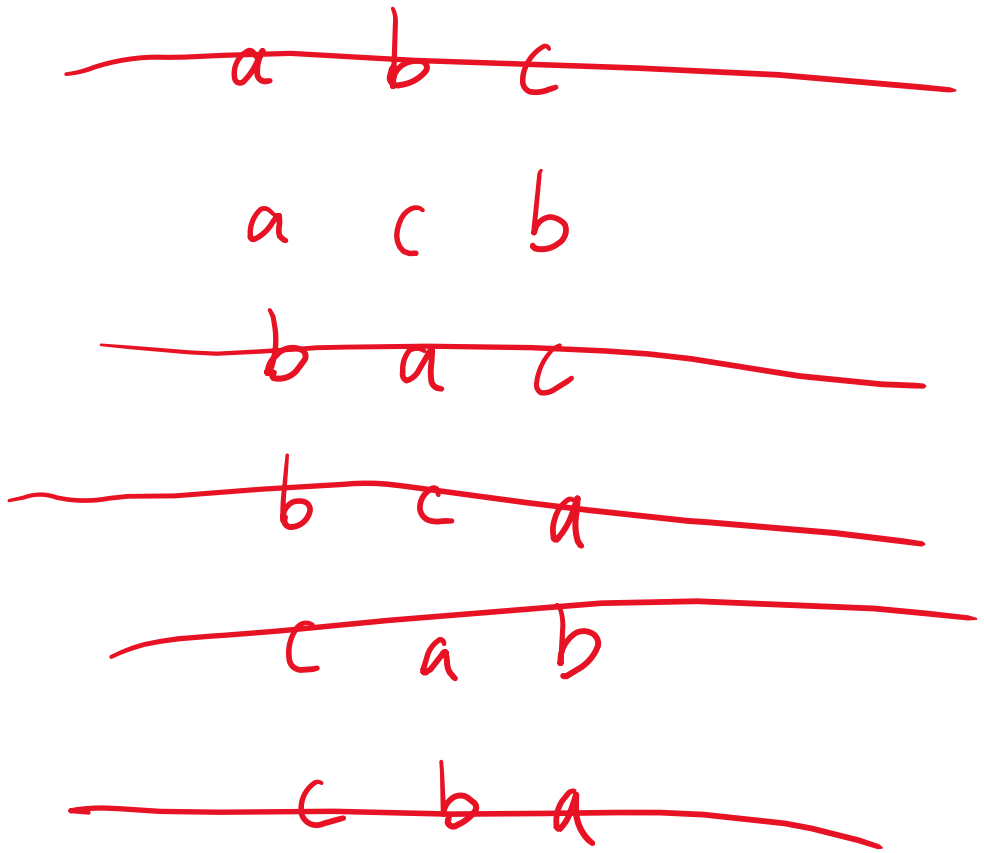
d a b c

a, b, c

$a \leq b$? Yes

$b \leq c$? No

$a \leq c$? Yes



questions needed to find the correct order

$$= \log_2(n!) = \log n + \log(n-1) + \cdots + \log 1$$

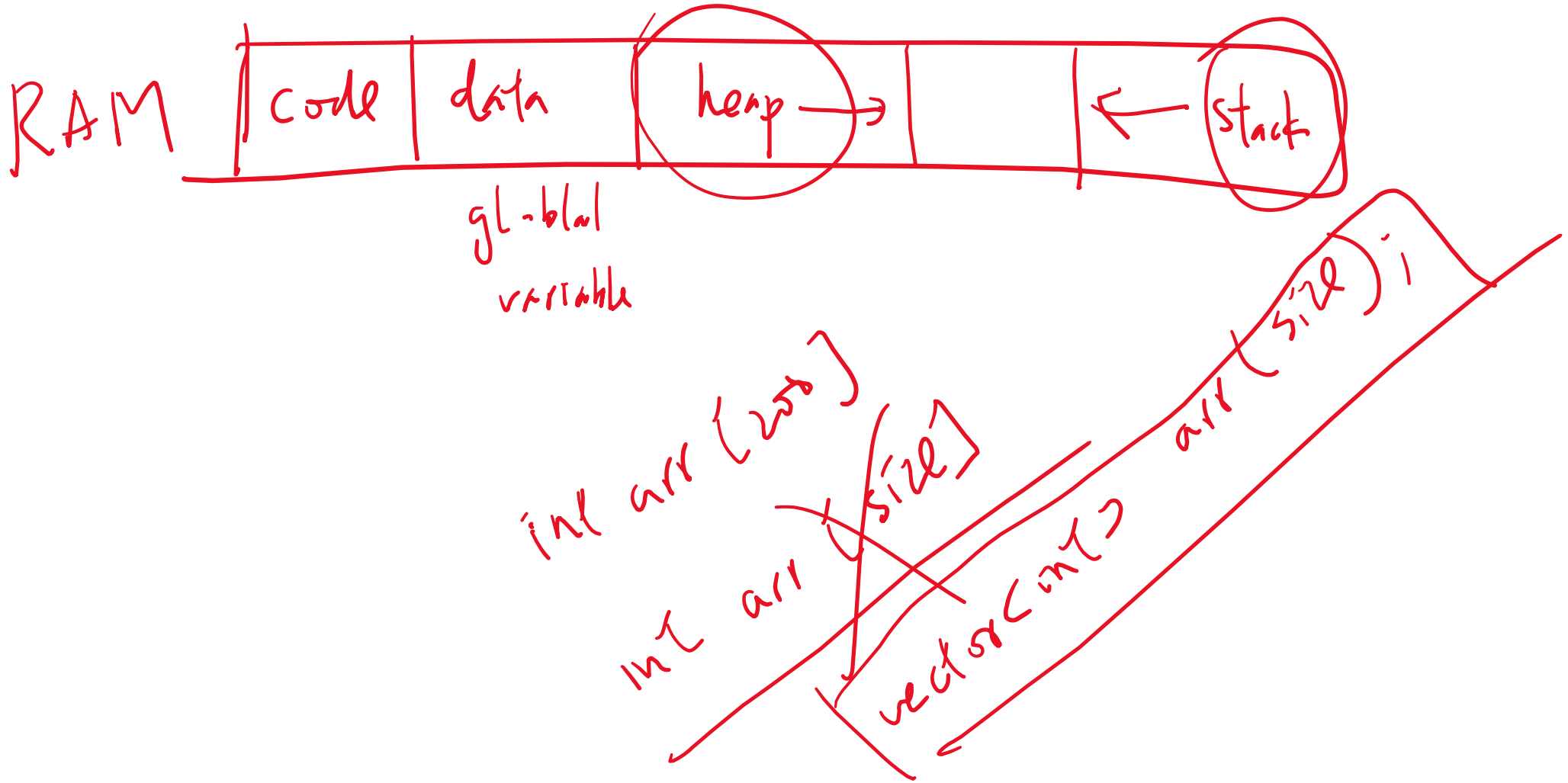
$$\geq \log n + \log(n-1) + \log(n-2) + \cdots + \log\left(\frac{n}{2}\right)$$

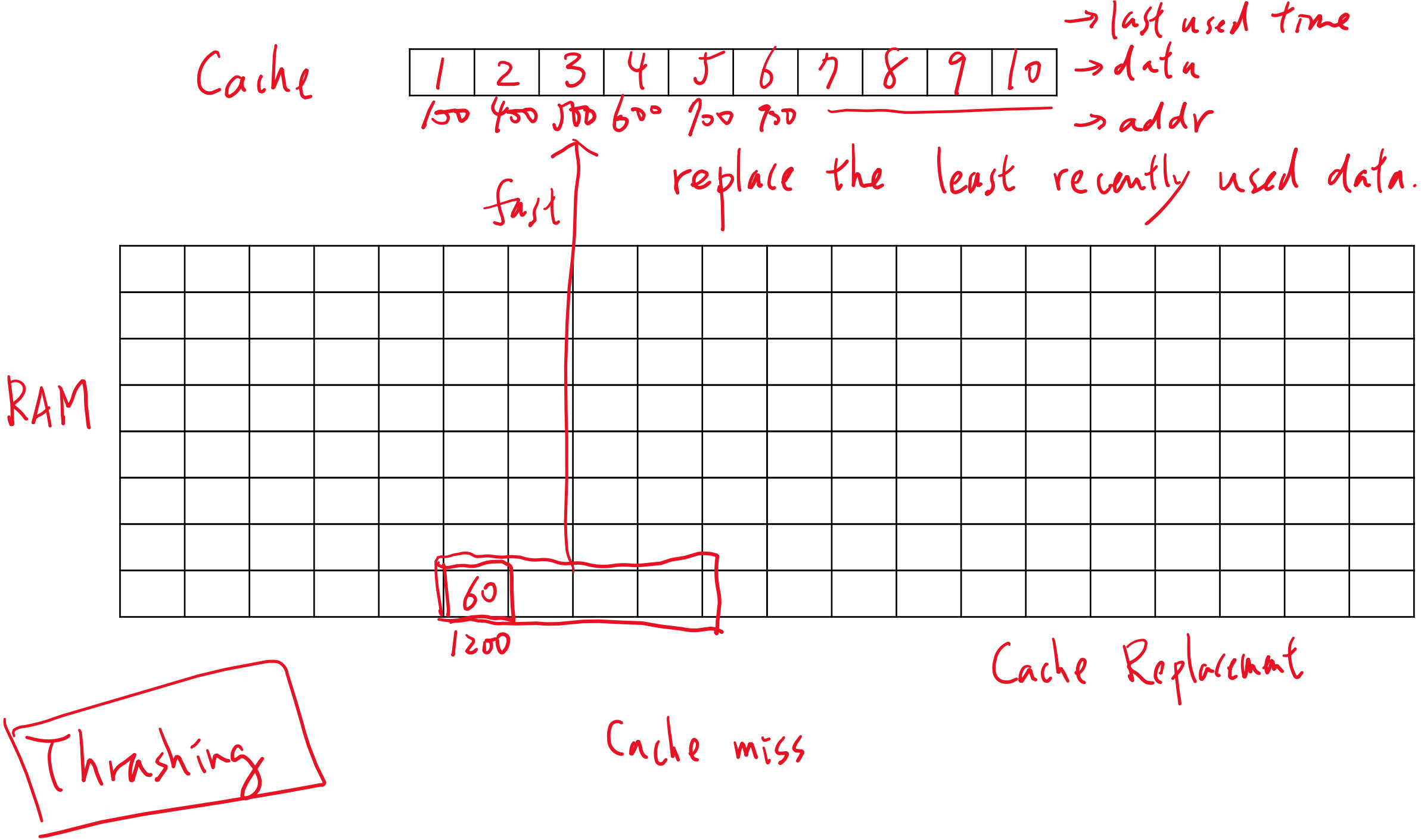
$$\geq \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2}\right) + \cdots + \log\left(\frac{n}{2}\right)$$

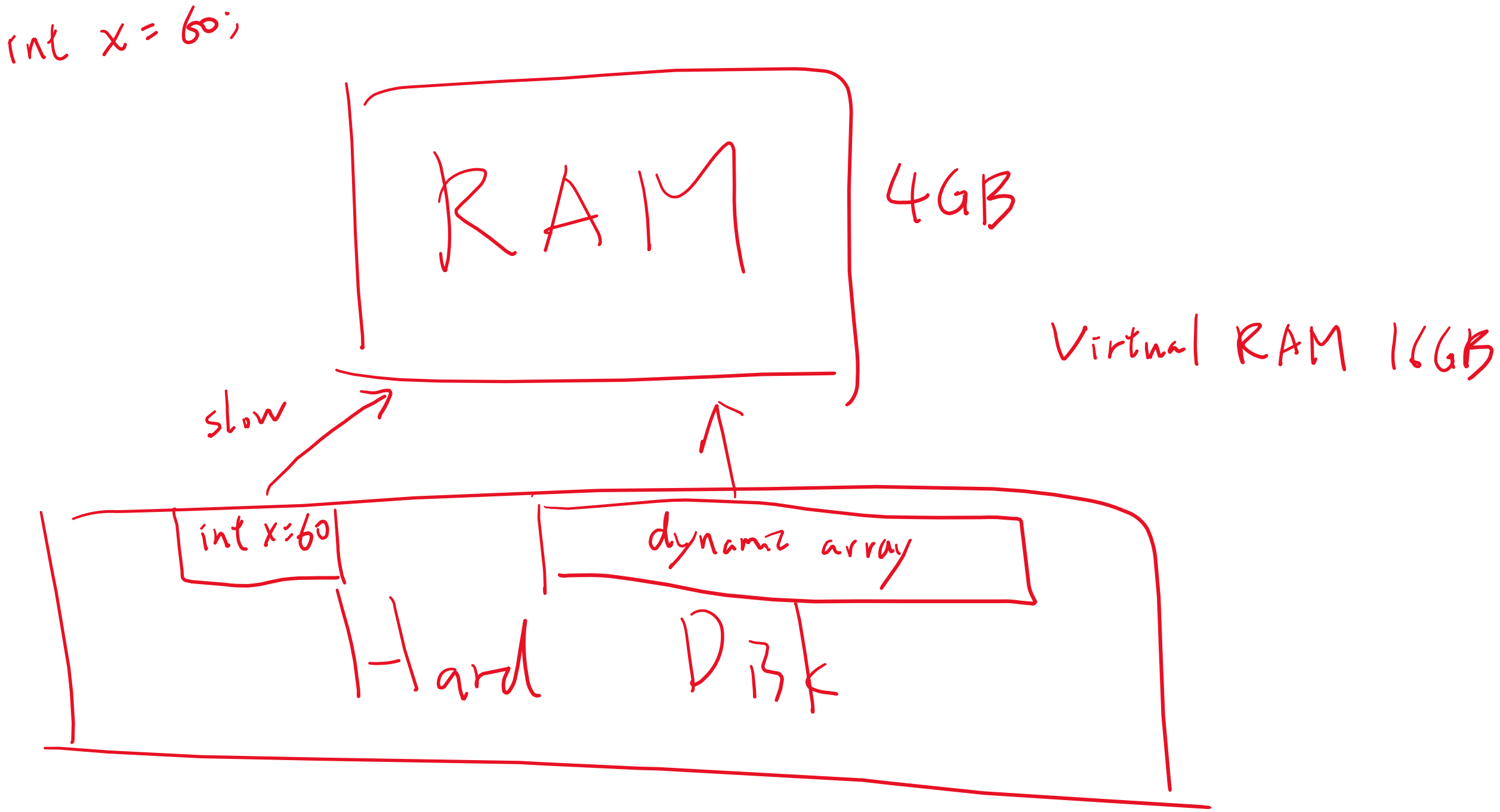
$$\geq \frac{n}{2}\log\left(\frac{n}{2}\right) = \boxed{\Omega(n \log n)}$$

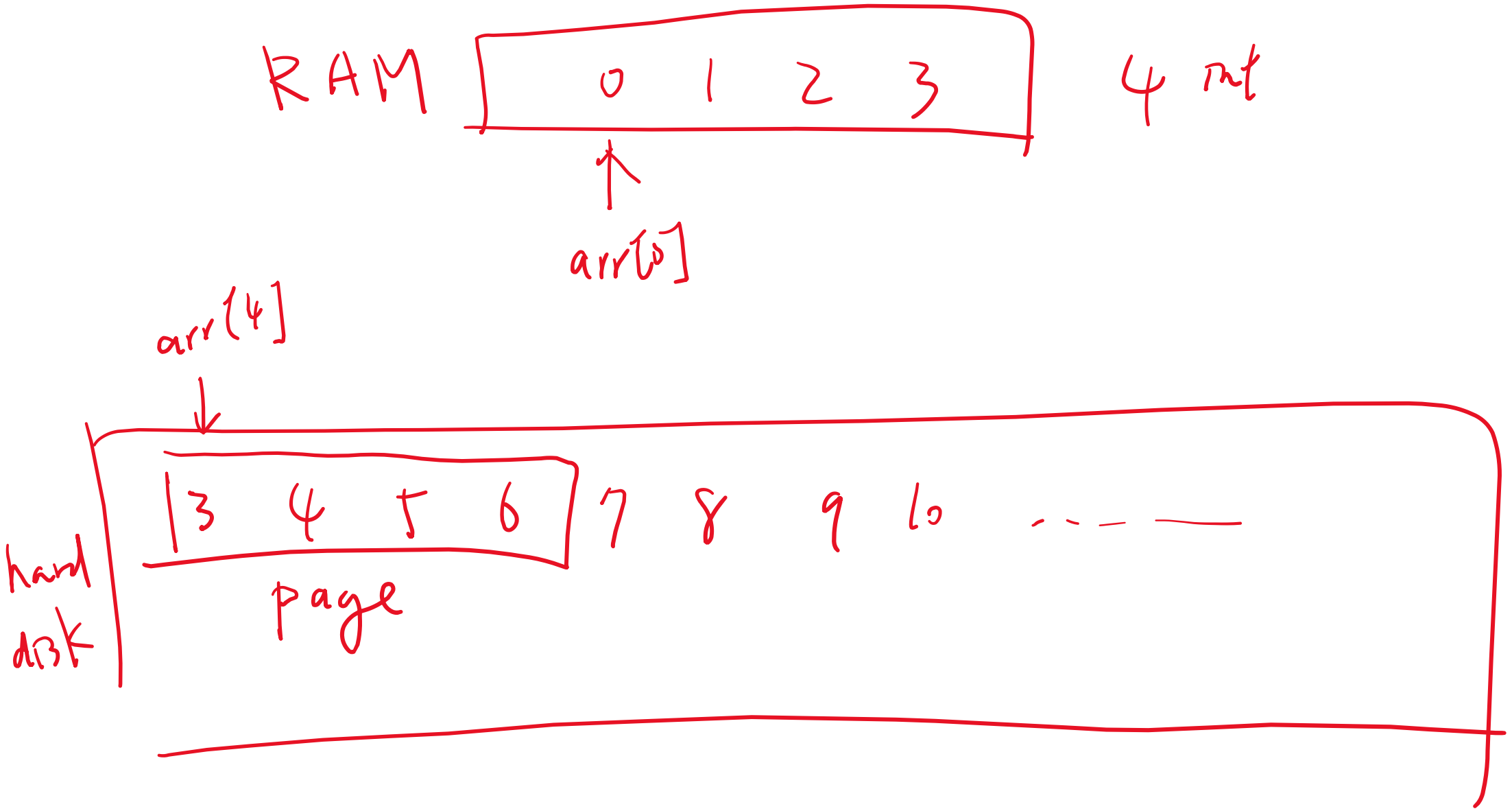
$\log \frac{n}{2} \cdot \log \frac{n}{2}$

$\therefore \log^2 \frac{n}{2}$


RAM
code
data
heap
stack
gl-blal
variable
int arr[size]
int arr(size)
vector<int> arr(size)


Cache
1 2 3 4 5 6 7 8 9 10
→ last used time
→ data
→ addr
100 400 500 600 700 750
fast
replace the least recently used data.
RAM
60
1200
Cache Replacement
Thrashing
Cache miss


int x = 60;
RAM
4GB
Virtual RAM 16GB
slow
int x:60
dynamic array
Hard Disk


RAM
0 1 2 3
4 nt
arr[0]
arr[4]
3 4 5 6
7 8 9 10
page
hard disk

$2^{27} \times 4$ bytes

$= 2^7 \times 4 \times 2^{20}$

$= 2^7 \times 4$ MB

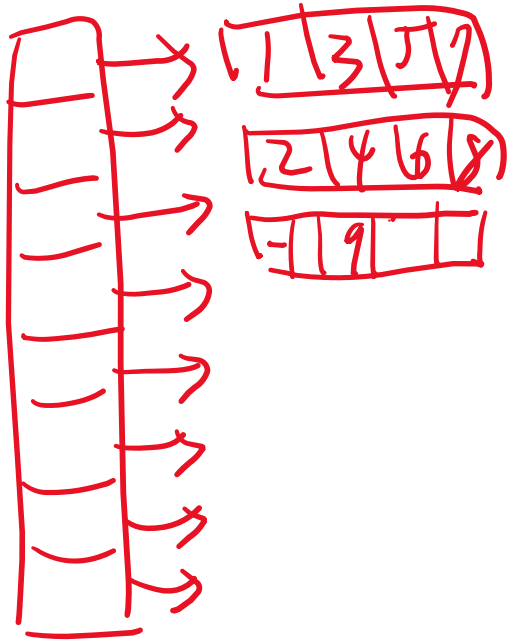
$= 512$ MB

Data structures that support fast insert, search, and sort

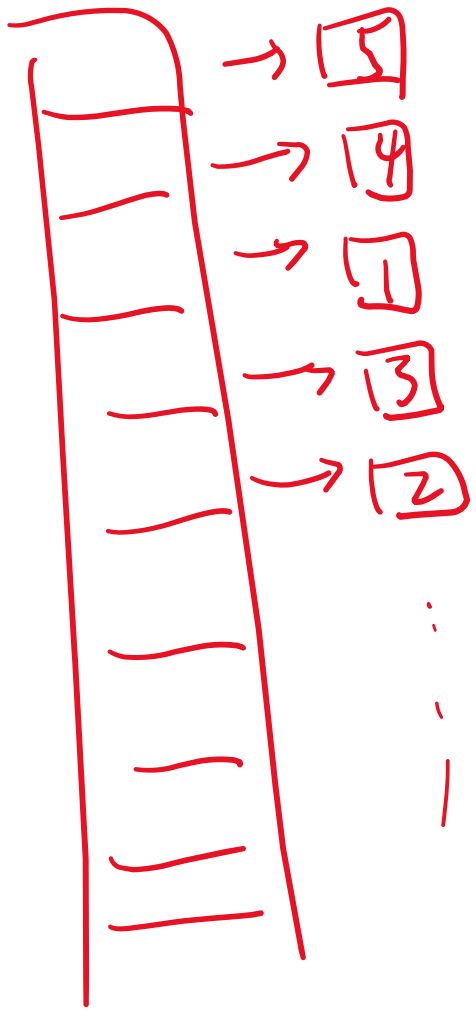
① Array of sorted arrays (HW3)

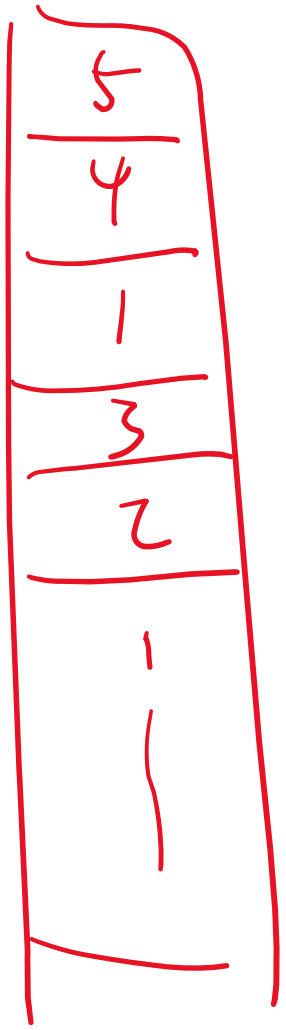
② Skip list

Array of Sorted Arrays



- Pros and Cons of sorted arrays
  - pros: fast search ($O(\log n)$)
  - cons: slow insert ($O(n)$)
- To speedup insert, data should be inserted into small arrays.
- Data structure cannot be a set of small arrays
  $\Rightarrow$ search is slow


5
4
1
3
2


5
4
1
3
2

Option 1: 1 large sorted array: slow insert
fast search

Option 2: $\Theta(n)$ small sorted arrays of size $O(1)$:
fast insert
slow search

Requirement:

① + ② ⇒ search in $O((\log n)^2)$ time

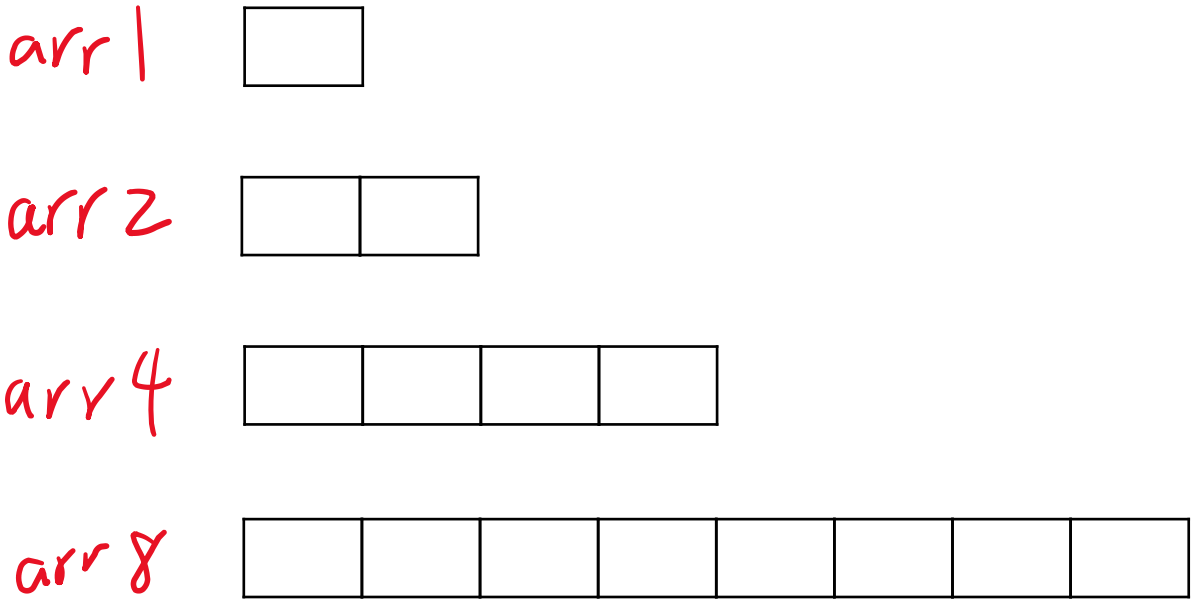
① All arrays are sorted.

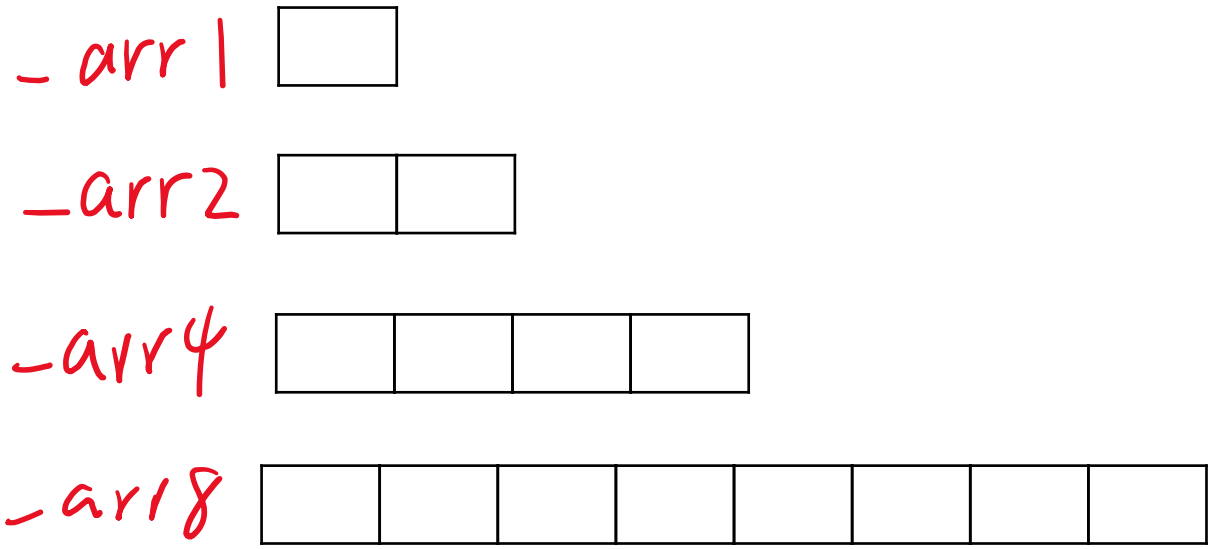
② 1 size-1 array + 1 size-2 array + 1 size-4 array + ...

$O(\log n)$ arrays

③ In most insertions, we insert the data to small arrays.

$O(\log n)$ amortized





To merge 2 size-k arrays, we have to insert 2k numbers.

After merge, these 2 size-k arrays are clear.

Time Complexity of Insertion: O(time complexity of all merges)

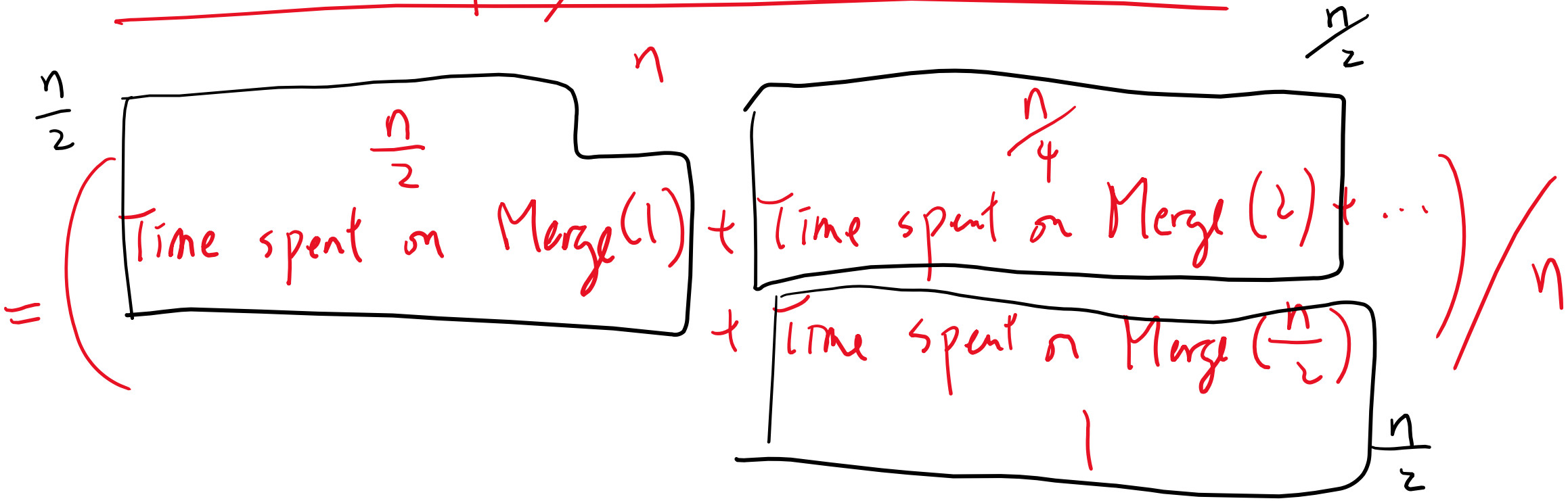
- Worst-Case: Merge(1) + Merge(2) + Merge(4) + Merge(8)
  + ... + Merge($\frac{n}{2}$)

  $= O(1) + O(2) + O(4) + O(8) + \dots + O(\frac{n}{2}) = O(n)$

Time complexity to merge 2 size-k sorted arrays → Merge(k) = O(k)

- amortized:

Time complexity of n insertions



$= \left(\text{Time spent on Merge}(1) + \text{Time spent on Merge}(2) + \cdots + \text{Time spent on Merge}\left(\frac{n}{2}\right) \right) / n$

$= \underline{O\left(\frac{n}{2} \times \log n\right)} = \underline{O(n \log n)} = O(\log n)$

$n \qquad n$

$$\frac{a(1-r^n)}{1-r}$$

$$1 + 2 + 4 + 8 + \cdots + \frac{n}{2}$$

$$\| $$

$$a + ar + ar^2 + \cdots + ar^{n-1}$$

$$n = 2^k$$

$$r = 2$$

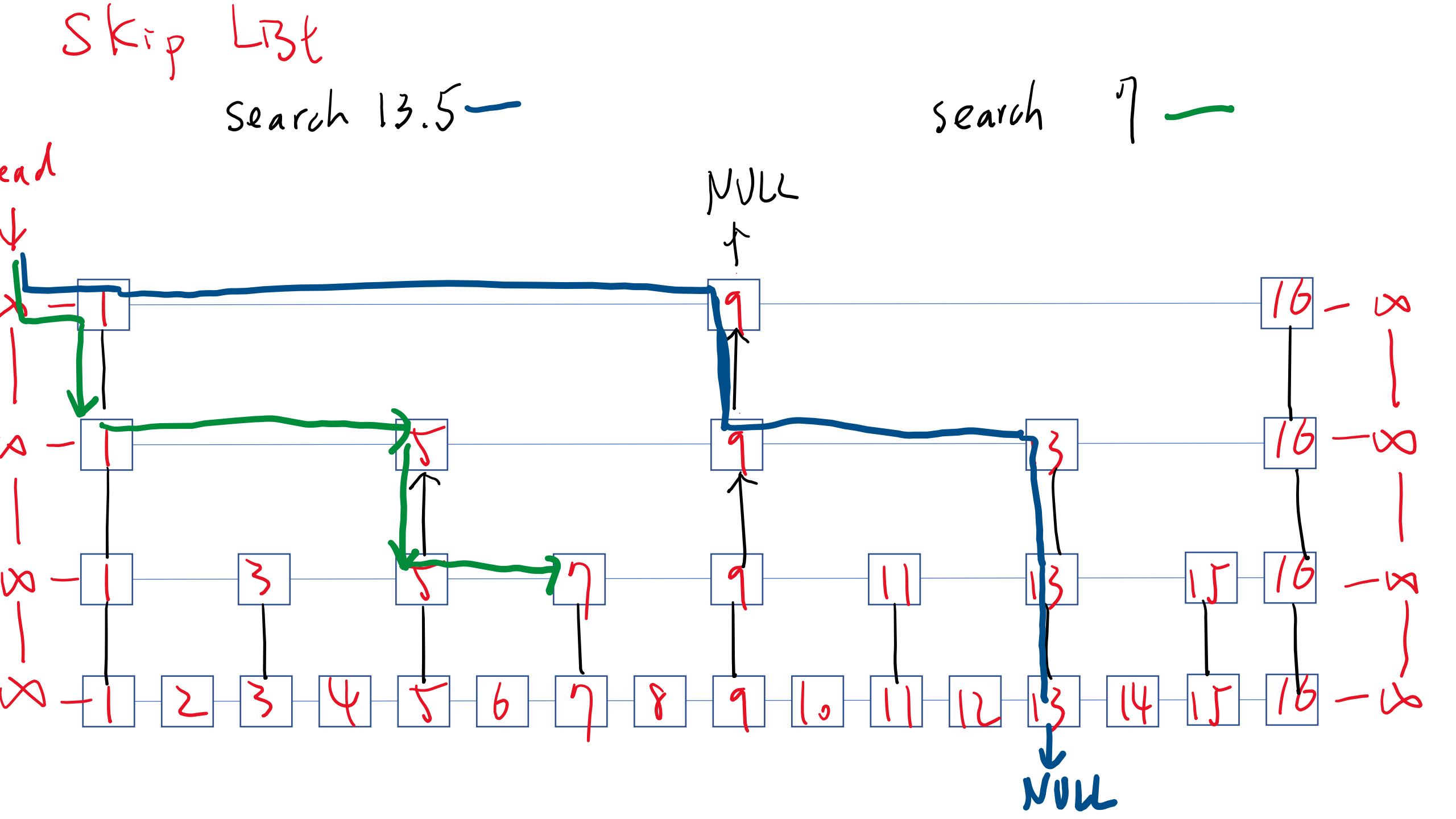
$$2^0 + 2^1 + \cdots + 2^{k-1}$$

$$\hookrightarrow r = 2$$

$$n = k = \log_2 n$$

$$\boxed{2^{\log_2 n} = n}$$

$$r^n = 2^{\log_2 n}$$


Skip List
search 13.5
search 7
NULL
NULL

```
struct Node{
    int data;
    struct Node* up;
    struct Node* down;
    struct Node* right;
    struct Node* left;
}
```

search 4

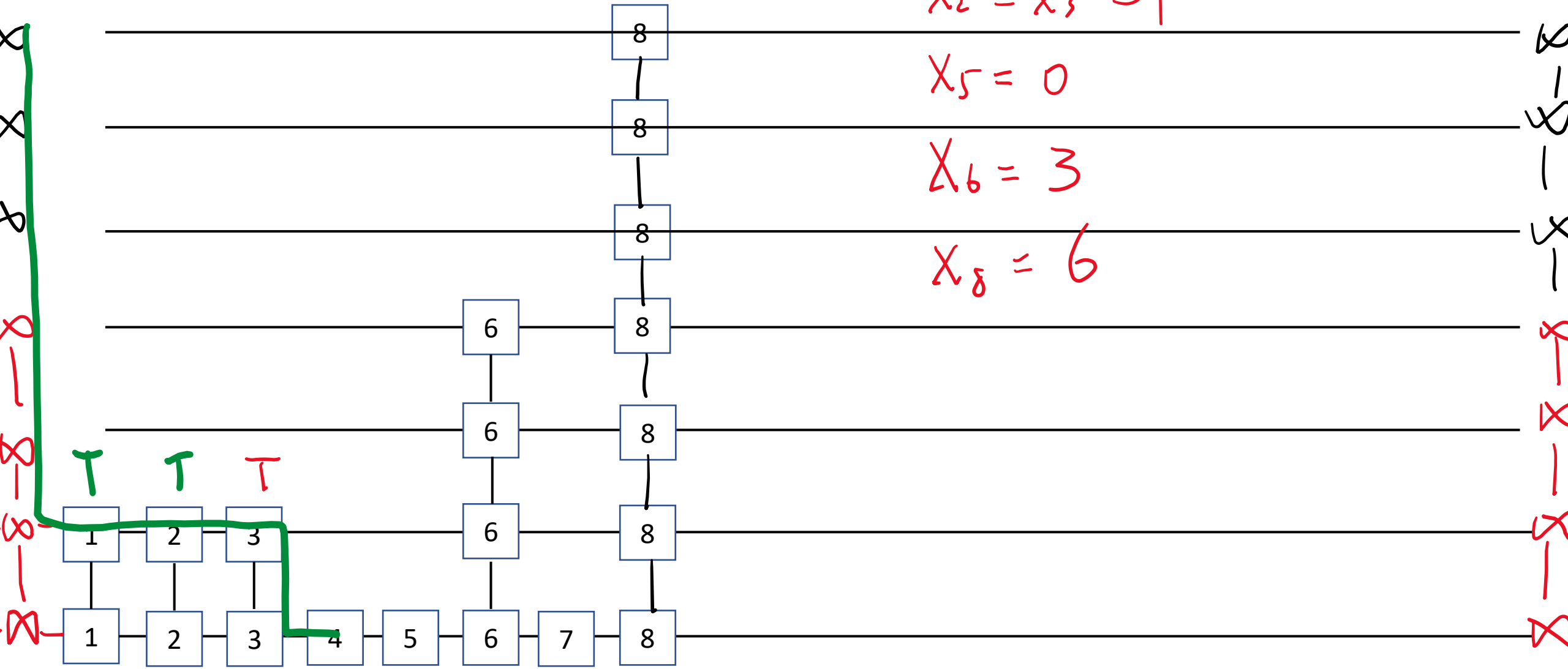
$X_1 = 1$

$X_2 = X_3 = 1$

$X_5 = 0$

$X_6 = 3$

$X_8 = 6$



```
search ( x ) {
    p = start
    while (true) {
        if ( p→ data == x ) return p
        if ( p → right → data ≤ x ) {
            p = p→right
        } else {
            if (p → down) p = p → down
            else return NULL
        }
    }
```

O(1) time

Insert (x) {

1° search to find the position to insert x

2° create a copy in the upper list if a head comes up.

}

X: # coin tosses to get the 1st tail

$E[X] = 2$

$$E[X] = 1 \times \frac{1}{2} + 2 \times \frac{1}{2^2} + 3 \times \frac{1}{2^3} + 4 \times \frac{1}{2^4} + \cdots$$

$$\frac{1}{2}E[X] = 1 \times \frac{1}{2^2} + 2 \times \frac{1}{2^3} + 3 \times \frac{1}{2^4} \cdots$$

$$\frac{1}{2}E[X] = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots = \frac{\frac{1}{2}}{1 - \frac{1}{2}} = 1$$

$E[X] = 2$

$$E[X] = 1 \times Pr[1^{st} \text{ coin toss} = \text{tail}]$$

$$+ (1 + E[X]) \times Pr[1^{st} \text{ coin toss} = \text{head}]$$

$$\Rightarrow E[X] = 1 \times \frac{1}{2} + (1 + E[X]) \times \frac{1}{2}$$

$$E[X] = 2$$

$$E[X] = E[X \mid 1^{st} \text{ coin toss} = \text{head}] \times Pr[1^{st} \text{ coin toss} = \text{head}]$$

$$1 + E[X]$$

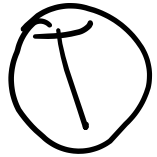
$$+ E[X \mid 1^{st} \text{ coin toss} = \text{tail}] \times Pr[1^{st} \text{ coin toss} = \text{tail}]$$

$$E[X \mid \text{case } 1] \times Pr[\text{case } 1]$$

$$+ E[X \mid \text{case } 2] \times Pr[\text{case } 2]$$

$$= E[X]$$

H, H, H, T $X = 4$



$X = 1$

Expected Time Complexity of Search

- Consider the "reversed" search path.

- Whenever there is ↑, we take it.

- E[search time] = E[search time in the 1st list]
  + E[— — — — — 2^{nd} —]
  + E[— — — — — — 3^{rd} —]
  + ...
  + E[last list]

Time complexity of search $= O(\#\text{ lists})$

$= \Theta$ (length of the search path)

$= \Theta$ (length of the search path on the 1st list

+ — — — — — — — — — 2nd last

+ — —

+ — — — — — last list)

$$E[\text{length of the search path on a list}] = O(1)$$

$$= O\Big(E[\#\text{ link traversals to encounter the first } \uparrow]\Big)$$

$$\Big(= 0 \times \Pr[1^{st} \text{ node has } \uparrow] + E[\#\text{ coin tosses to get the } 1^{st} \text{ head}] \Big)$$

$$\Big(= 0 \times \Pr[1^{st} \text{ node has } \uparrow] + \underbrace{E[\#\text{ coin tosses to get the } 1^{st} \text{ head}]}_{\| \; 2} \Big)$$

$$E[\#\text{ additional copies of a data}]$$

$$= E[\#\text{ coin tosses } \textbf{before} \text{ we get a tail}]$$

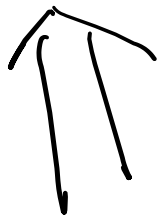
$$= E[\#\text{ coin tosses } \textbf{to} \text{ get the 1st tail}] - 1$$

$$= 1 = O(1)$$

$$E[\text{time complexity of insert}] = E[\text{time complexity of search}] + O(1)$$

Pr[# lists $= O(\log n)$] is high

very



additional

Pr[# lists $\leq 100 \log n$] is very high

X_i : # additional copies for the i^{th} insertion

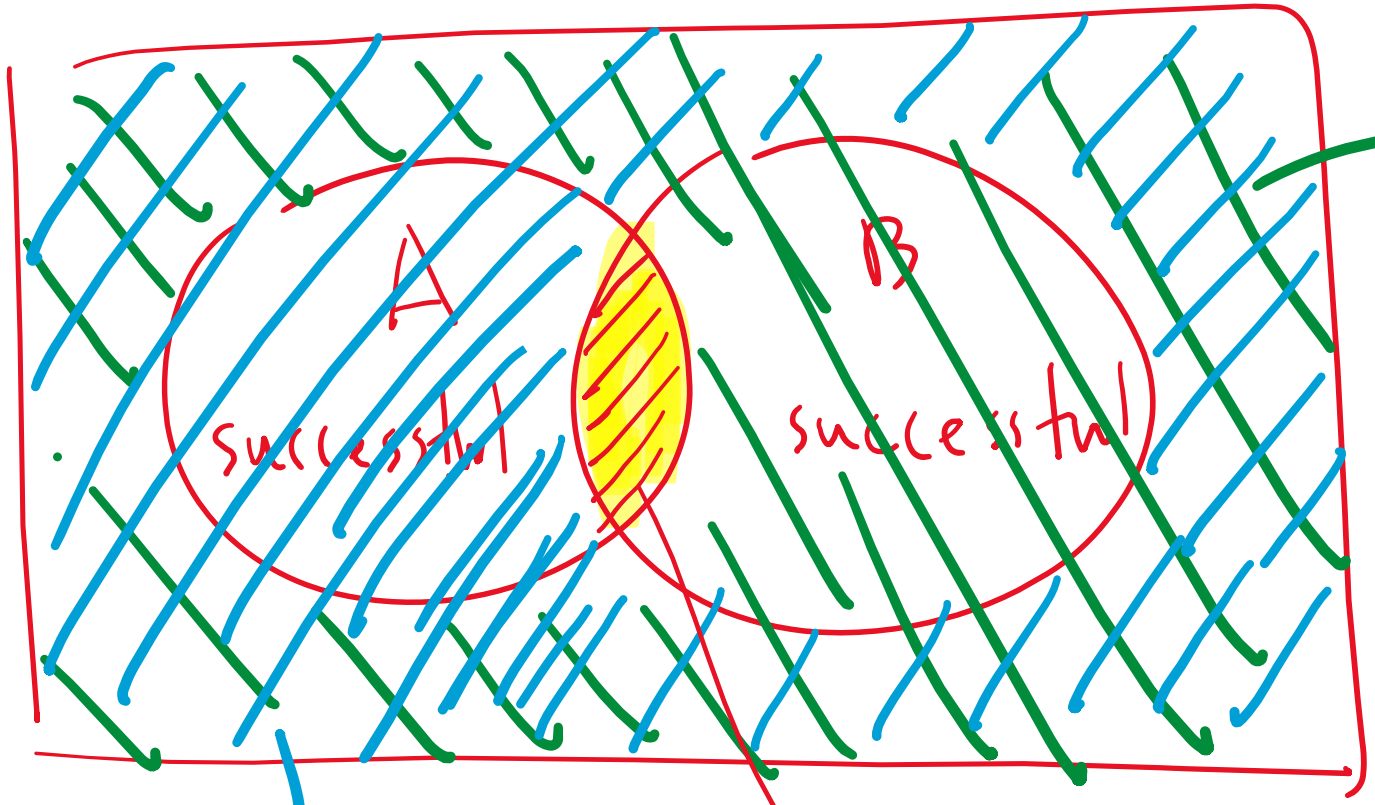
additional lists $= \max_i X_i$

$$Pr\left[\max_i X_i \leq 100 \log n\right]$$

$$= Pr\left[X_1 \leq 100\log n \text{ and } X_2 \leq 100\log n \text{ and } \cdots \text{ and } X_n \leq 100\log n\right]$$

$$= 1 - \Big(\underbrace{Pr\left[X_1 > 100\log n\right]}_{A} \text{ or } \underbrace{X_2 > 100\log n}_{B} \text{ or } \cdots \text{ or } \underbrace{X_n > 100\log n}_{C}\Big)$$

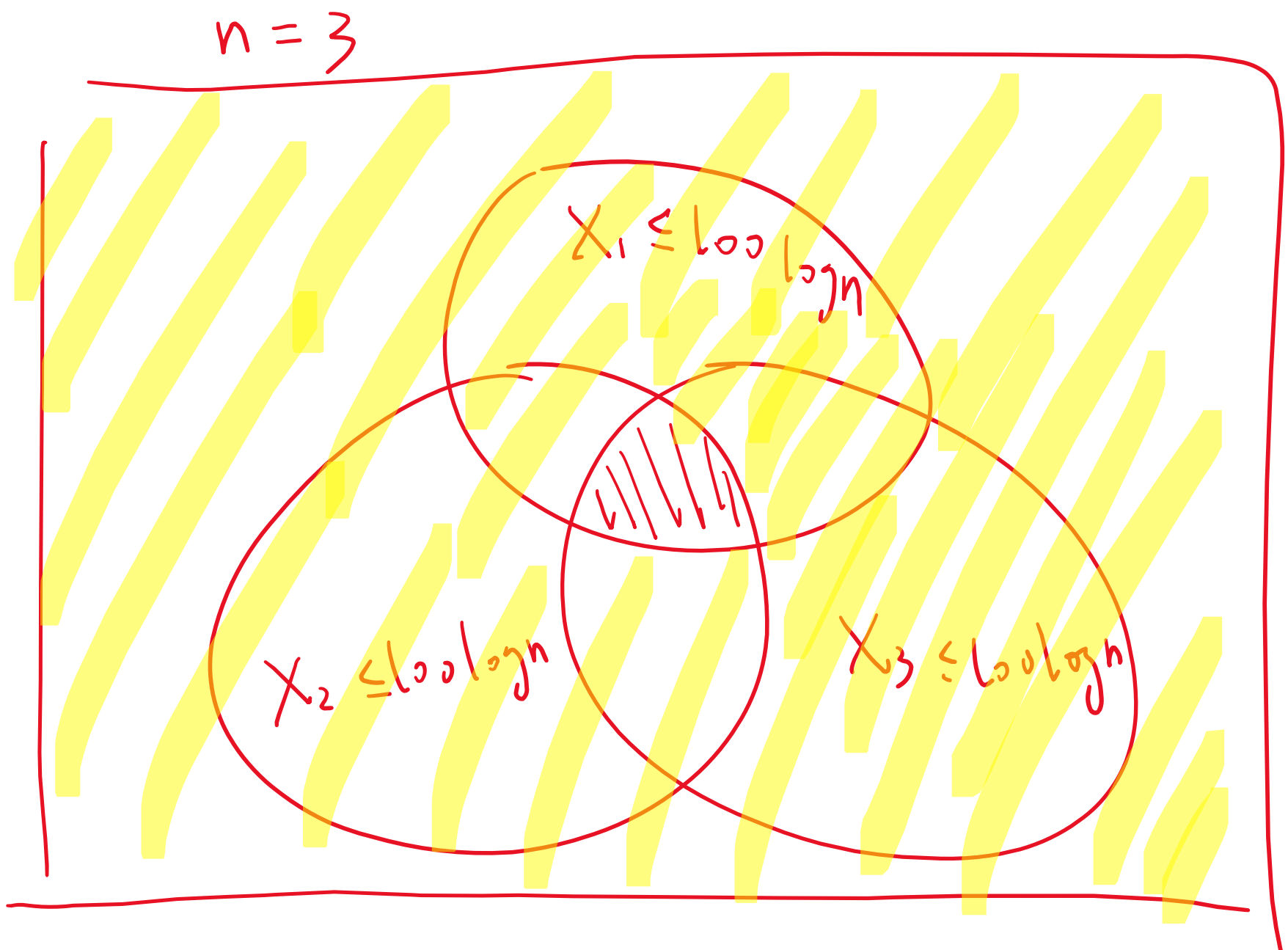
$$\geq 1 - \left(\underbrace{Pr\left[X_1 > 100\log n\right]}_{A} + \underbrace{Pr\left[X_2 > 100\log n\right]}_{B} + \cdots + \underbrace{Pr\left[X_n > 100\log n\right]}_{C}\right)$$



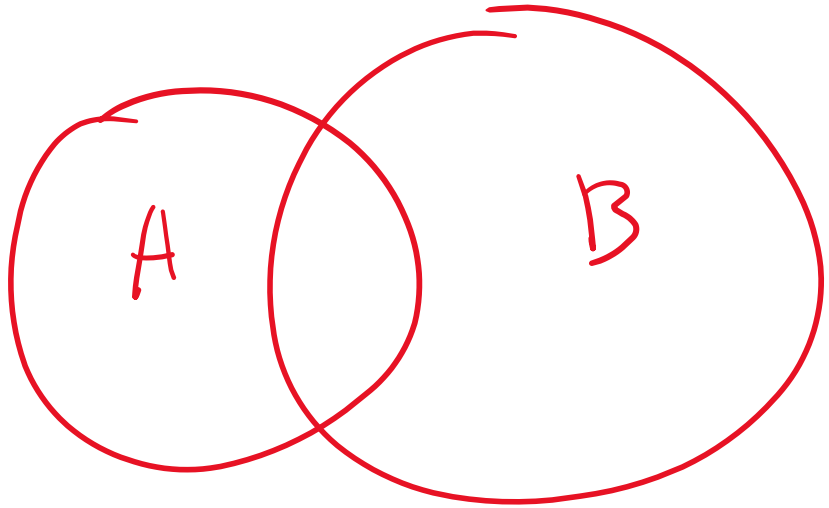
A fail

B fail

1 − Pr[A fail ∪ B fail]


n = 3
$X_1 \leq 100 \log n$
$X_2 \leq 100 \log n$
$X_3 \leq 100 \log n$

$$Pr[A \cup B] \leq Pr[A] + Pr[B]$$



$$-Pr[A \cup B] \geq -(Pr[A] + Pr[B])$$

(fail prob.) $Pr[X_i > 100\log n]$

$$= \frac{1}{2^{100\log n + 1}}$$

$$= \frac{1}{2 \times 2^{100\log n}} = \frac{1}{2(2^{\log_2 n})^{100}} = \frac{1}{2 \times n^{100}} \text{ (very small)}$$

$Pr[X_i > 3]$

$$= \frac{1}{2^4}$$

$$\boxed{2^{\log_2 n} = n}$$

$$Pr\left[\max_i X_i \leq 100 \log n\right] \approx 1$$

$$\geq 1 - \left(Pr[X_1 > 100 \log n] + Pr[X_2 > 100 \log n] + \ldots + Pr[X_n > 100 \log n]\right)$$

$$= 1 - \left(n \times \frac{1}{2 \times n^{100}}\right)$$

$$= 1 - \boxed{\frac{1}{2 \times n^{99}}} \approx 1$$

$\rightarrow$ very small

$\Pr\left[\max_i X_i > 100 \log n\right]$ is very small!

$= \Pr[X_1 > 100 \log n \text{ or } X_2 > 100 \log n \text{ or } \cdots$

$\text{or } X_n > 100 \log n]$

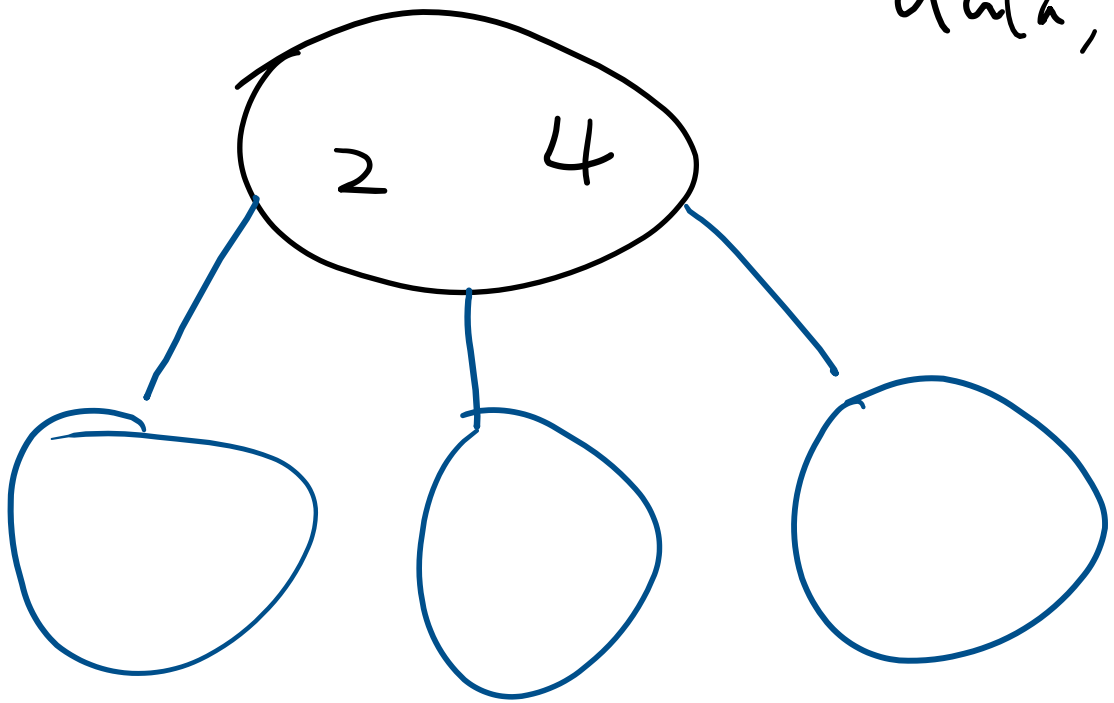
Summary:

With high probability, search, insert can be done in $O(\log n)$ ti

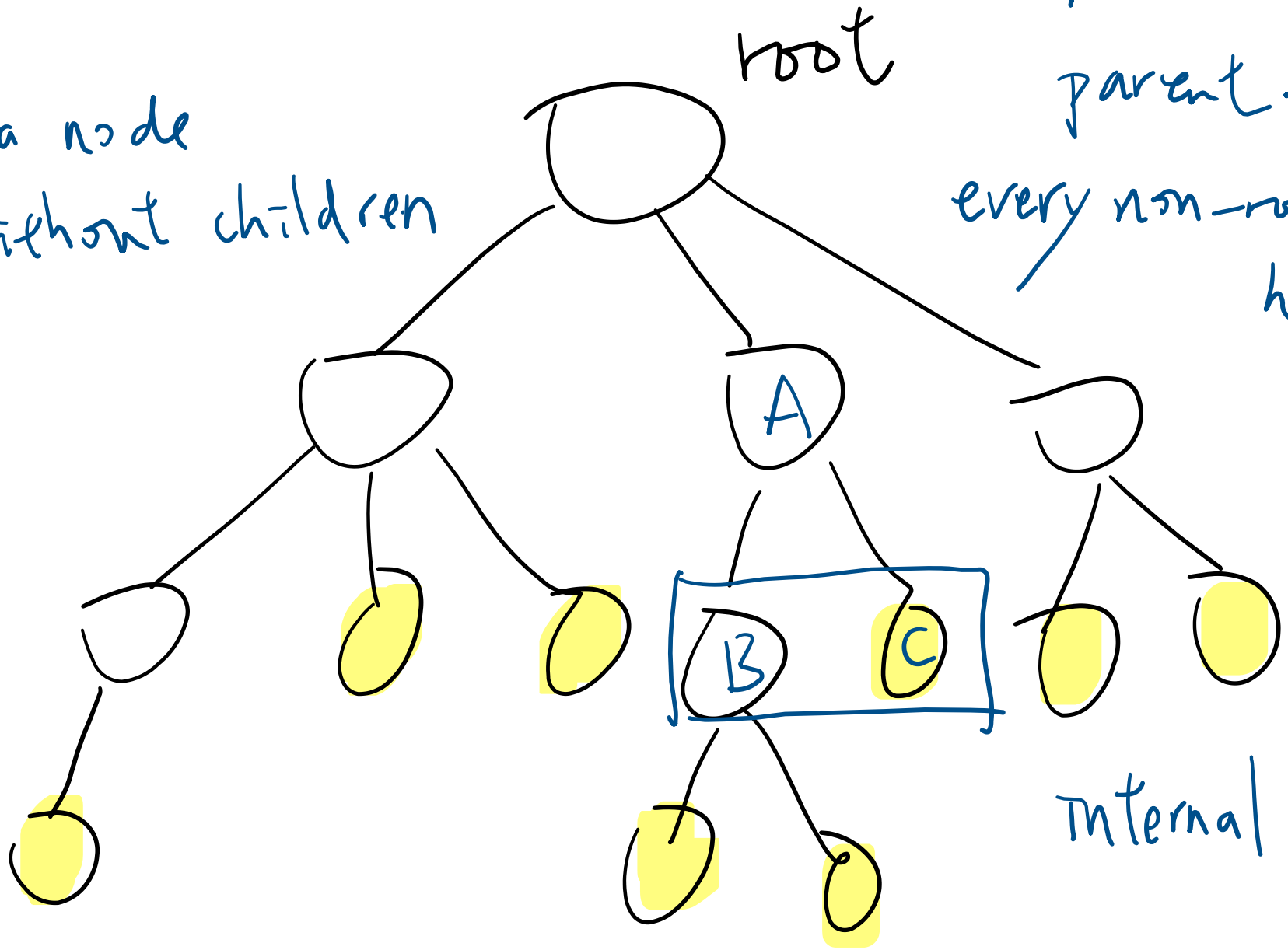
B-tree

- Key property : ① A node contains 1 or 2 data

② If an internal node has k data, it has k+1 ~~children~~ subtrees



Tree



leaf: a node without children

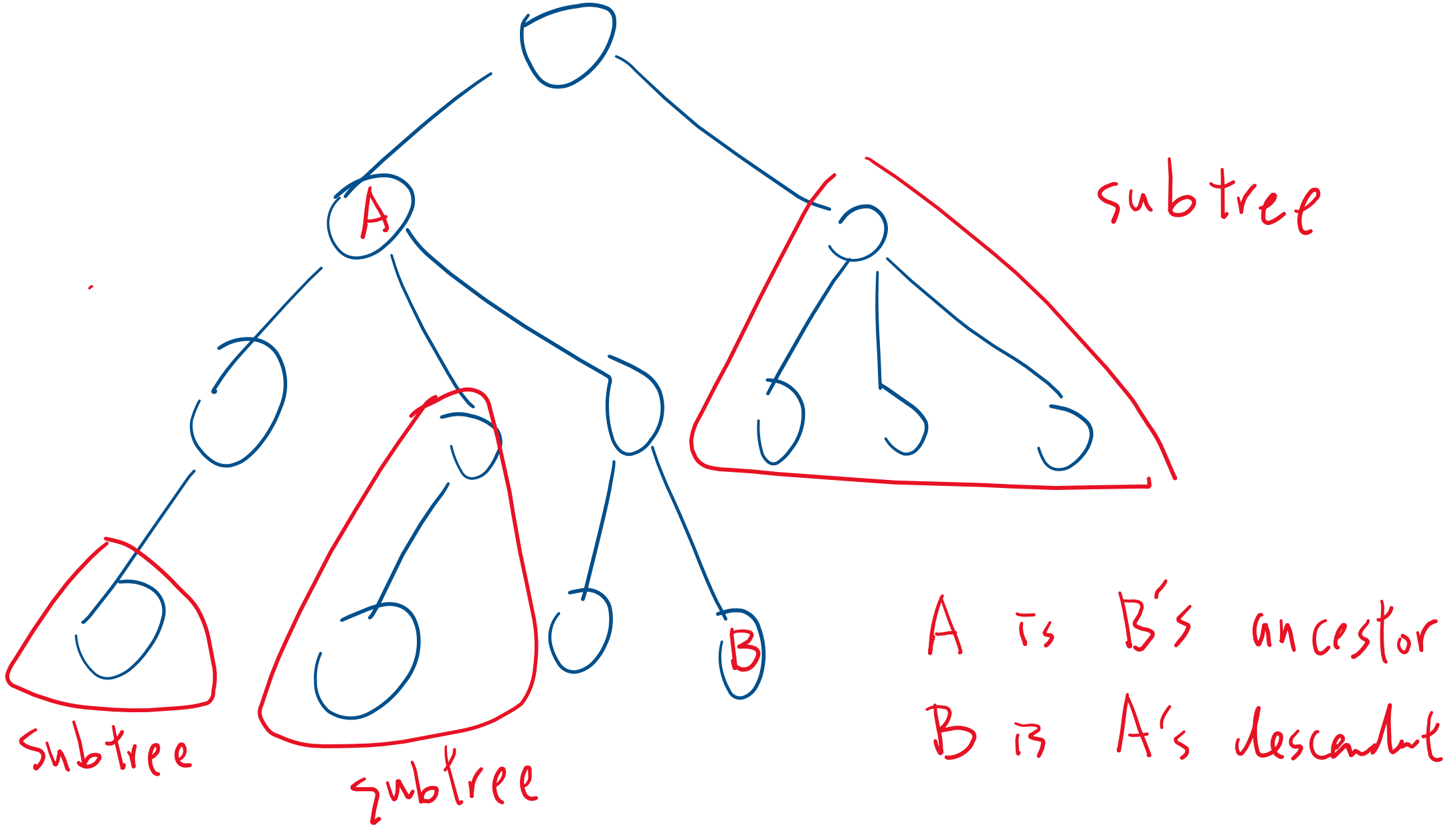
A's children: B, C

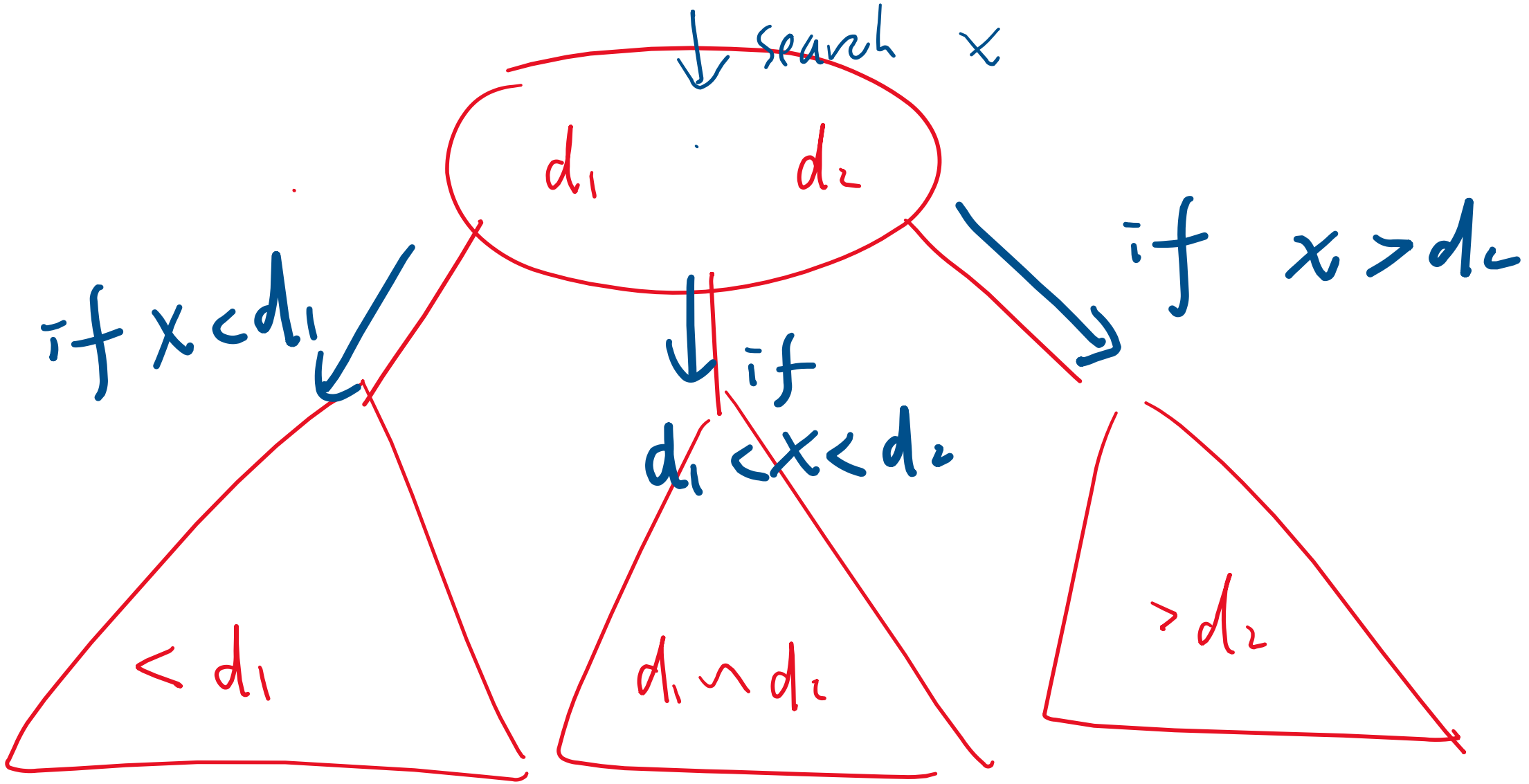
parent of B (and C): A

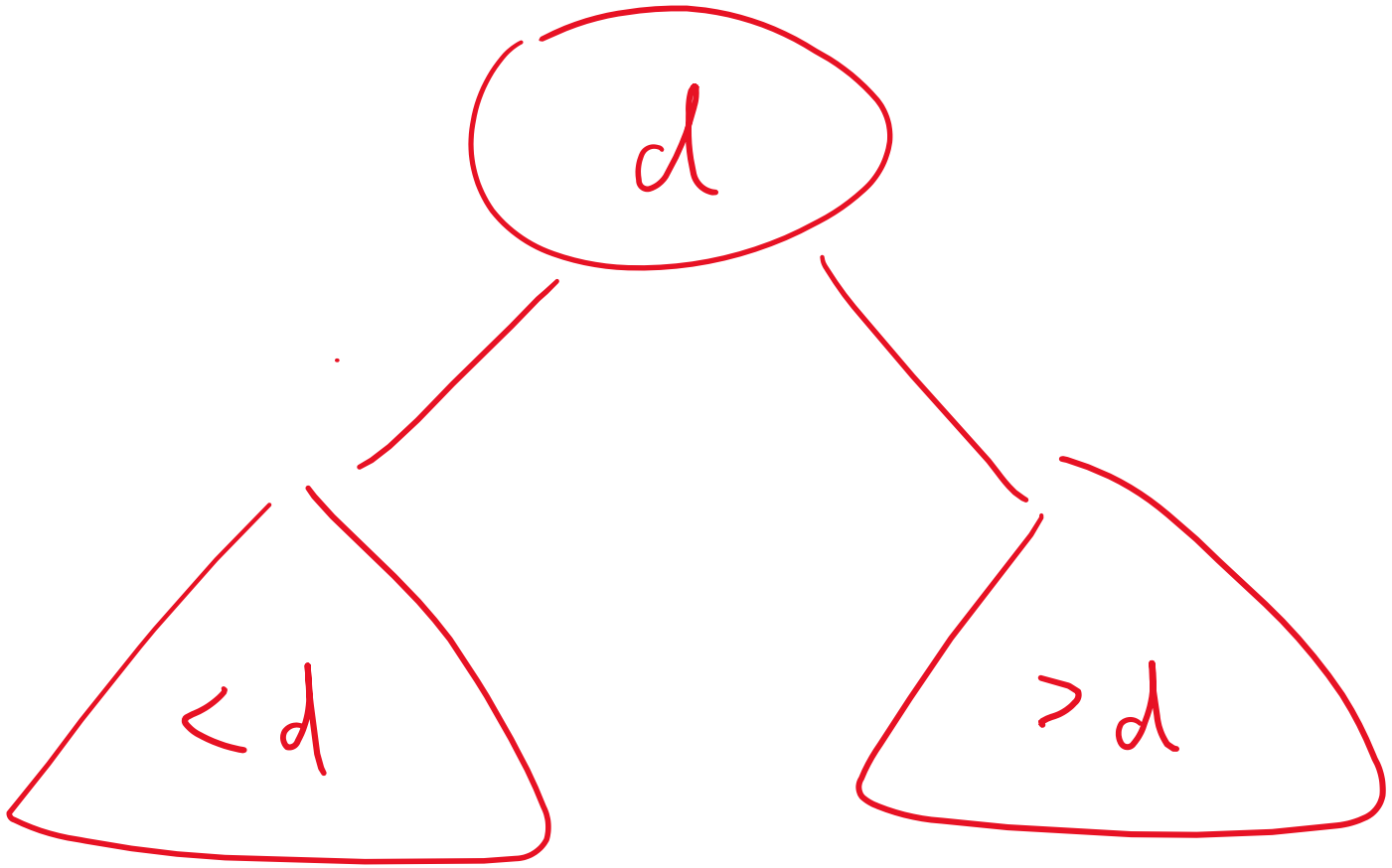
every non-root node has a parent

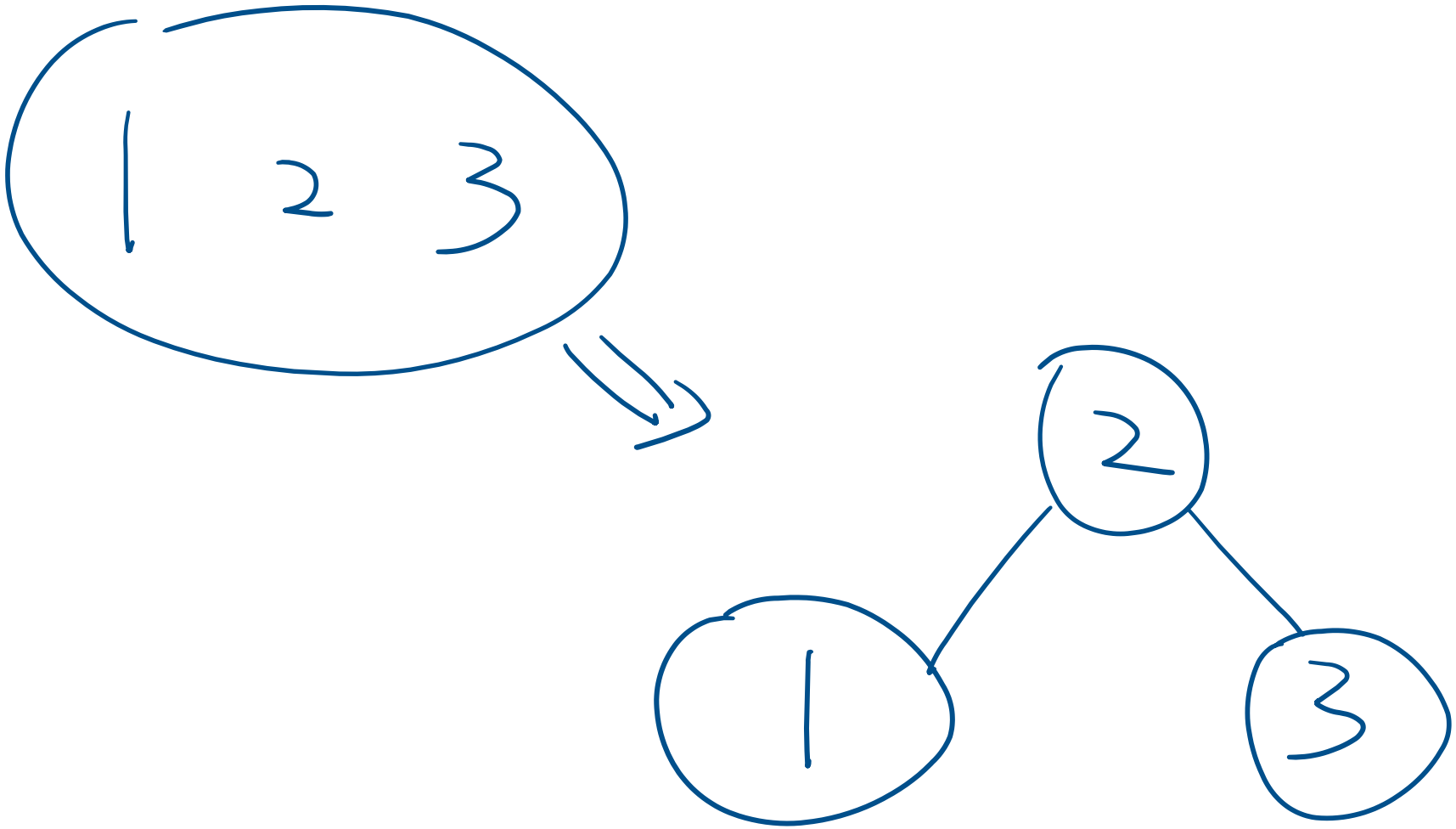
leaf

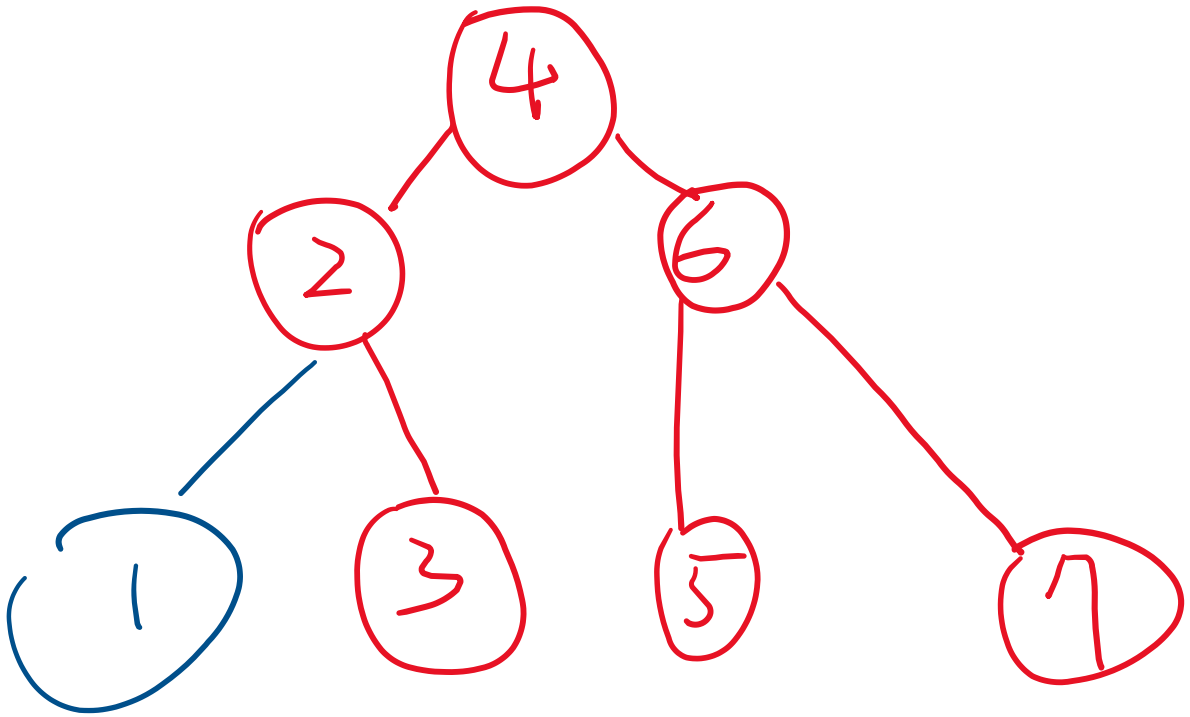
internal node: non-leaf node

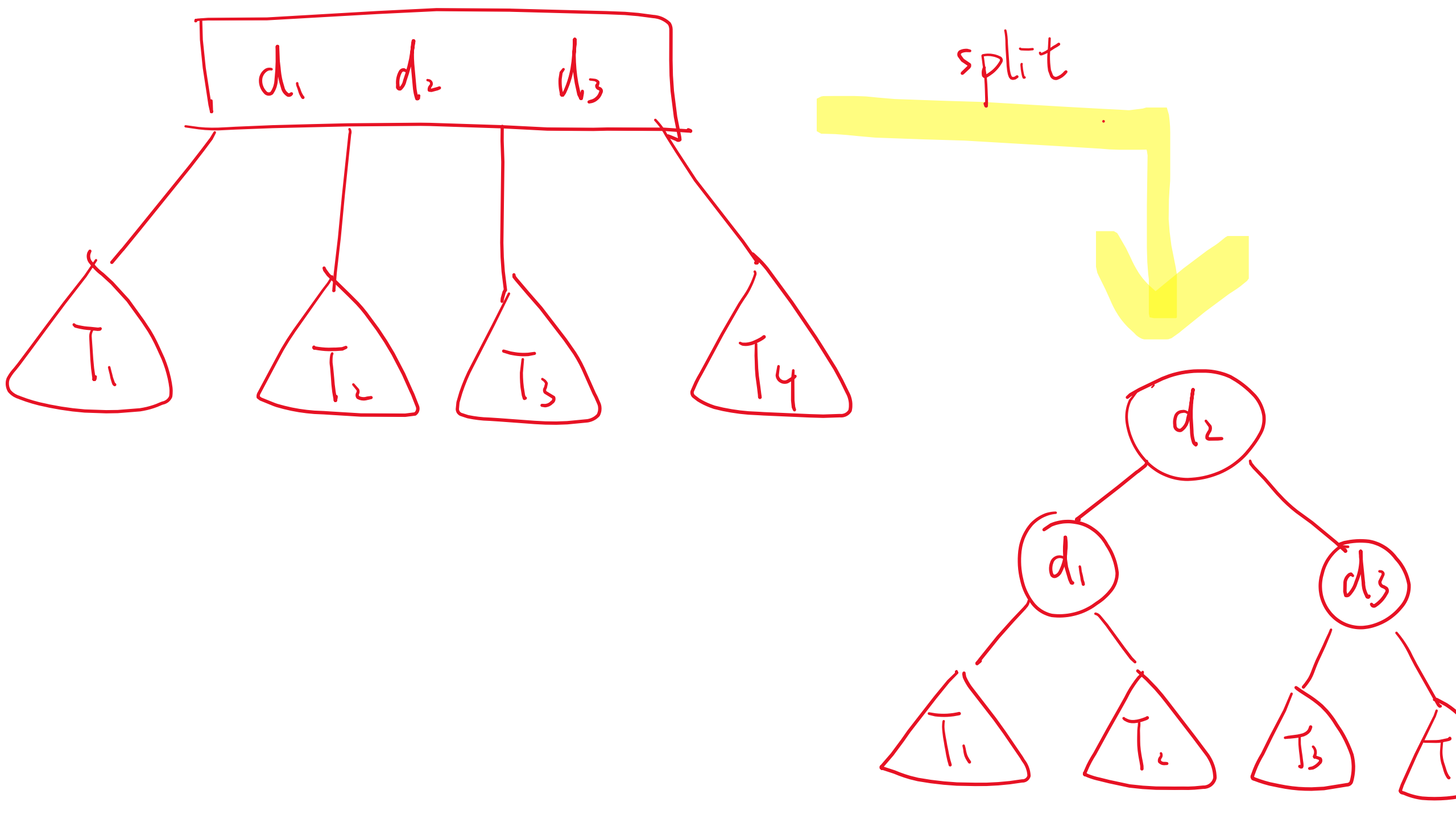

A
B
subtree
subtree
subtree
A is B's ancestor
B is A's descendant


search x
d_1 d_2
if $x < d_1$
if $d_1 < x < d_2$
if $x > d_2$
$< d_1$
$d_1 \sim d_2$
$> d_2$


d
< d
> d


1 2 3
2
1
3


4
2
6
1
3
5
7


d_1 d_2 d_3
T_1 T_2 T_3 T_4
split
d_2
d_1 d_3
T_1 T_2 T_3

Time complexity of search = O(tree height)

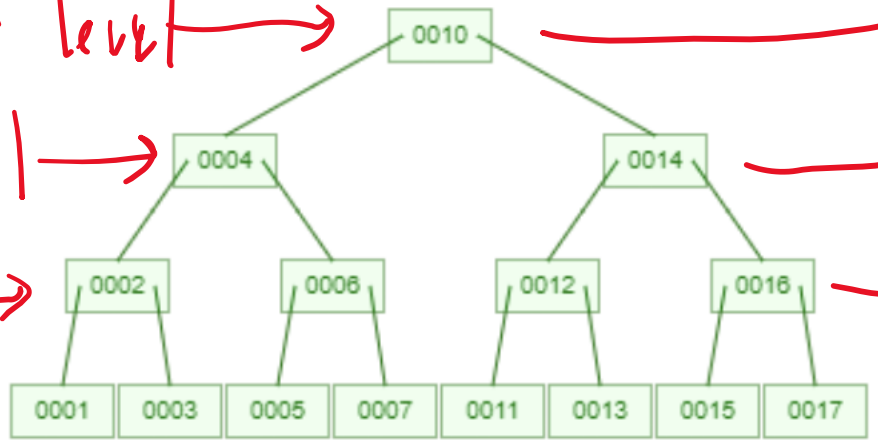
Time complexity of insertion = O(tree height)

Claim: tree height = $O(\log n)$

nodes = O(n)

$2^0 + 2^1 + 2^2 + \dots + 2^x \geq n$

$x = O(\log n)$



the 0th level → 1 — 2^0

1st level → 2 — 2^1

2nd level → 4 — 2^2

8 — 2^3

16

32

⋮

|

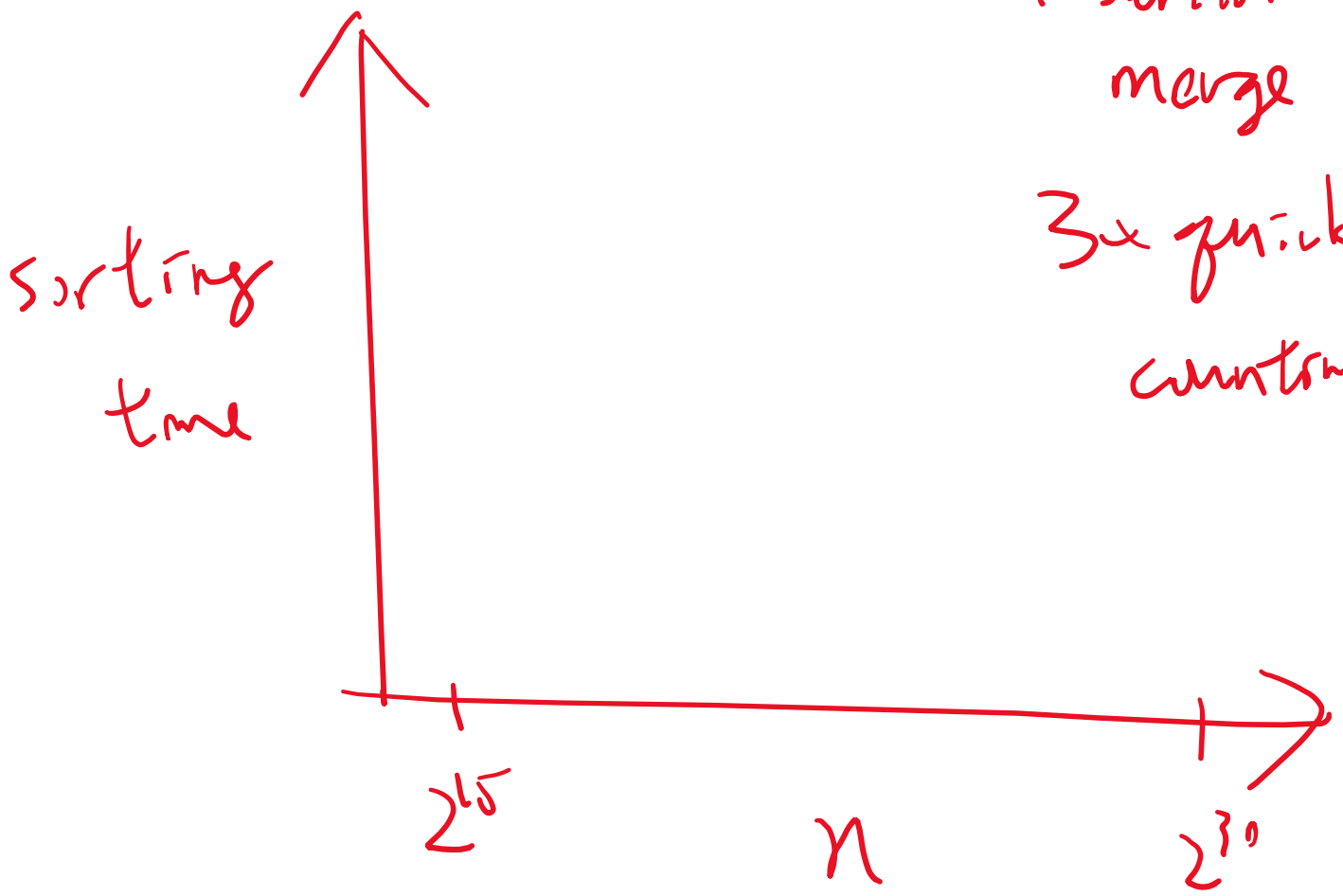
nodes in the i^{th} level $\geq 2^i$

insertion sort

merge sort

3x quick sort

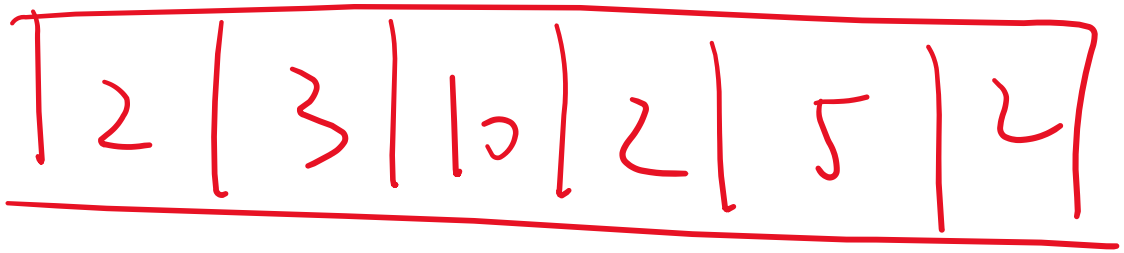
counting sort

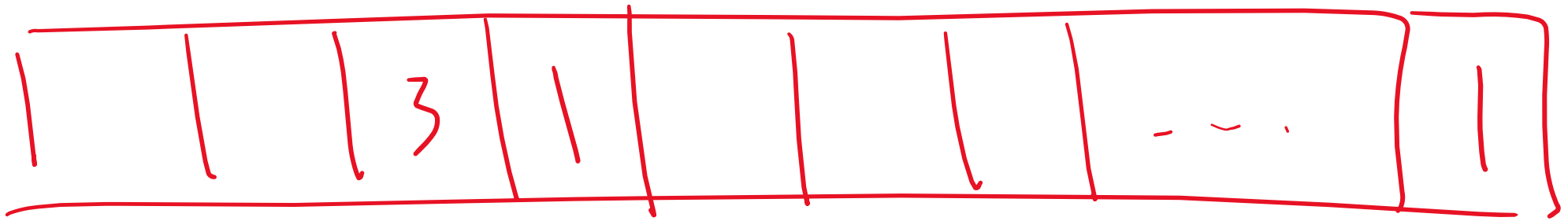


1M/sec

max: k

all data 0 or positive int





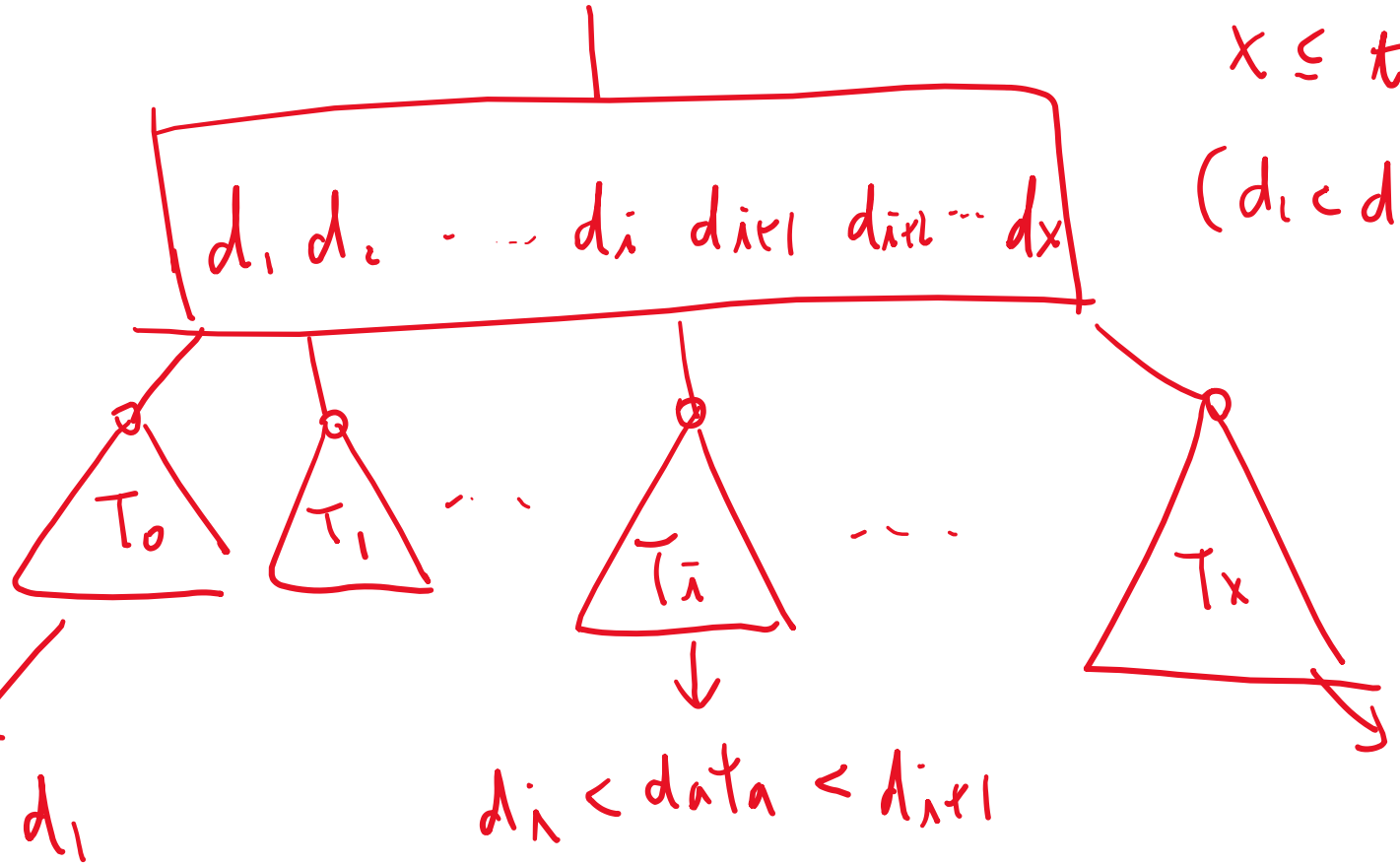
B-tree

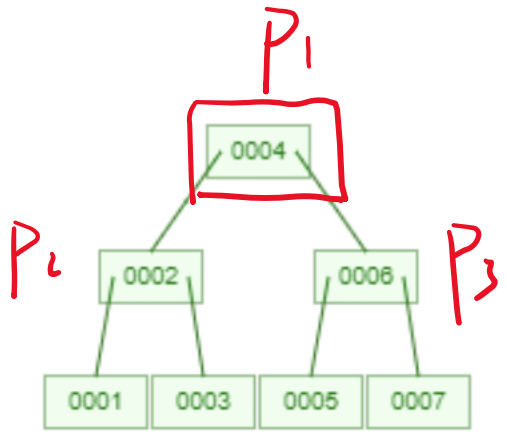
$t = O(1)$ (e.g. $t = 2000$)

- A node can store up to $t = 2k$ data ($k \in \mathbb{N}$)
- A node that stores x data looks like

$x \leq t$

$(d_1 < d_2 < \cdots < d_x)$



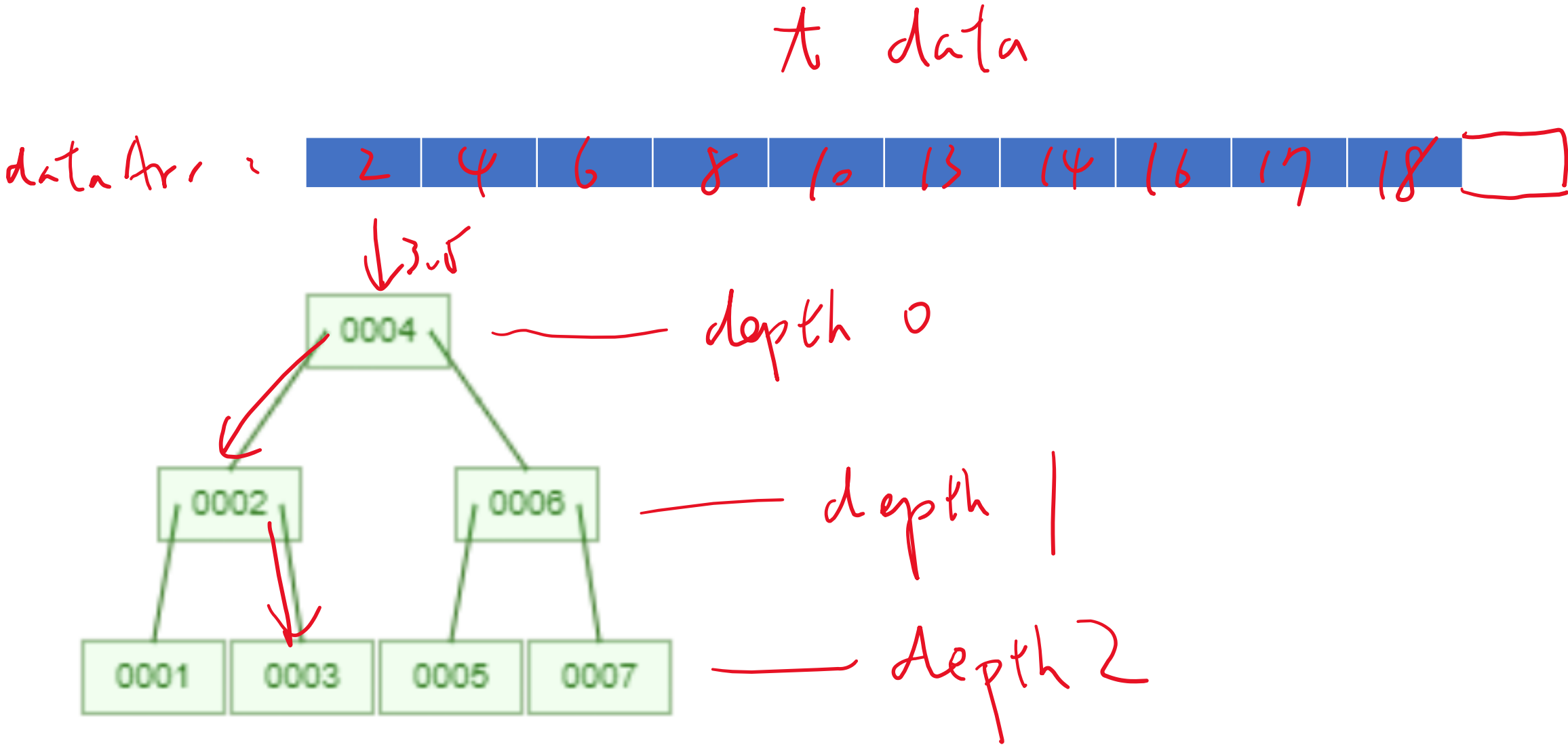


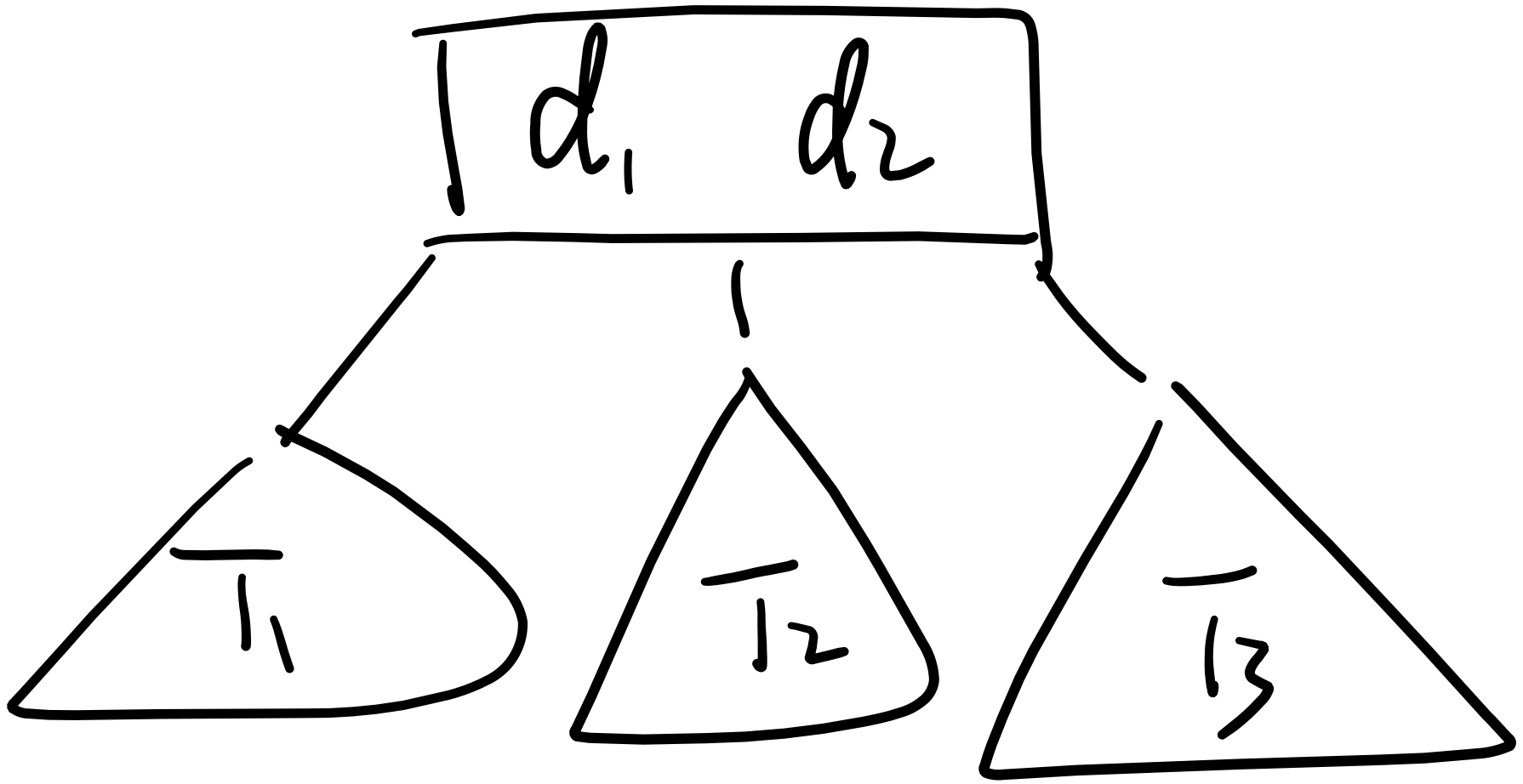

```
struct node {
    int* arrData; // a sorted array of data
    struct node** children // an array of pointers to the children
}
```

root

{4},

{P2, P3}


data
dataArr :
2
4
6
8
10
13
14
16
17
18
3.5
0004
depth 0
0002
0006
depth 1
0001
0003
0005
0007
depth 2


d_1 d_2
T_1
T_2
T_3

Search (x) :

Struct Node *p = root;

while (true) {

if (x in p->arrData) {

return p;

}

if (p is leaf) {

return NULL

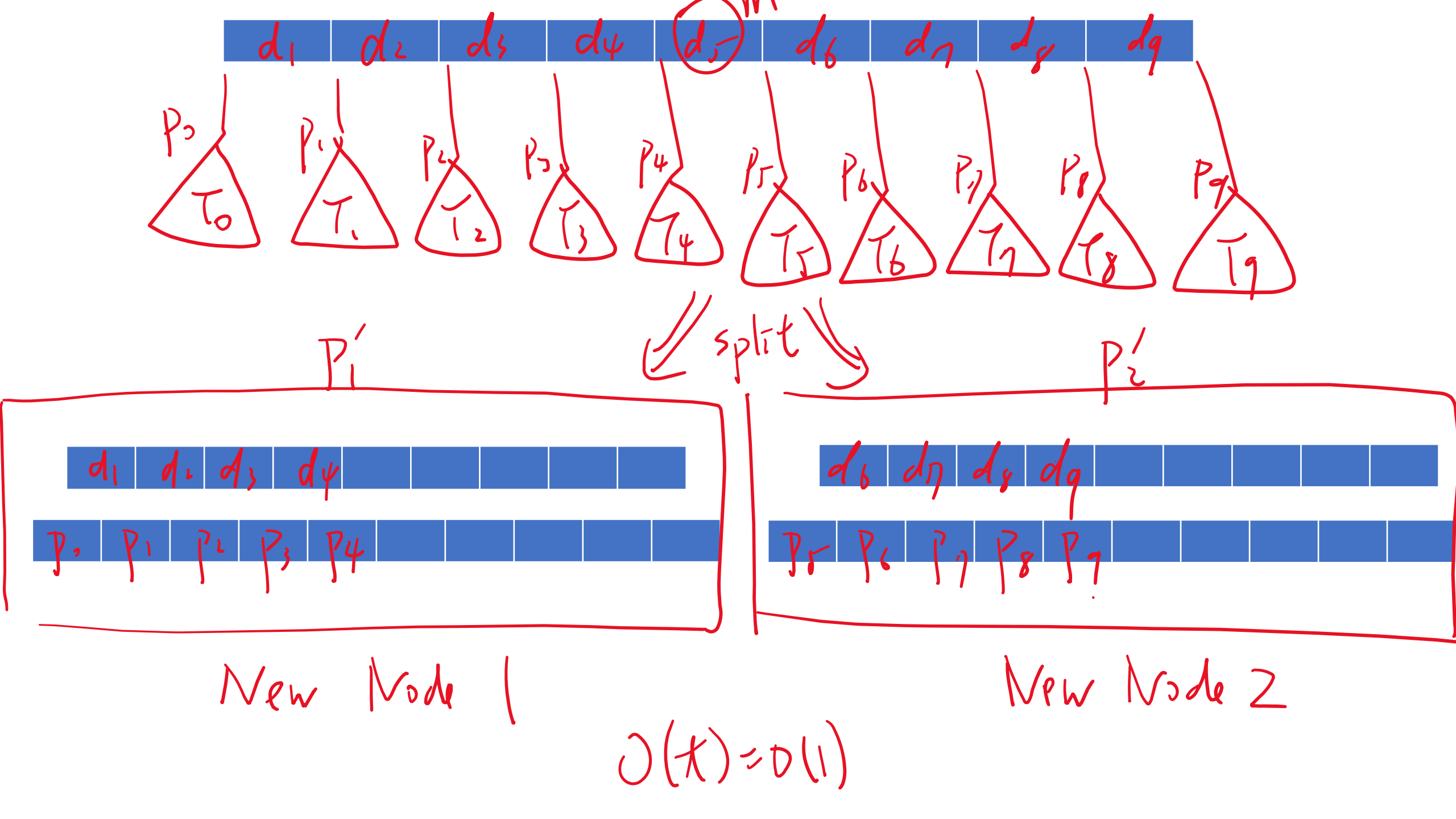
}

p = the root of T_j such that

$d_j < x < d_{j+1}$

O(1) ⟸

→ binary search



$O(k) = O(1)$

Insert (x):

$O(\text{tree height})$ ← 1° search x to find the leaf L that can store x.

$O(t) \to O(1)$ 2° store x in L

3° Whenever a node has $t+1$ data, split the node and store the medium in the parent node (create a new root if parent does not exist)

$O(\text{tree height} \times \text{split time}) = O(\text{tree height})$

- Search & Insert can be done in $O(\text{tree height})$ time

$t \geq 2k$

Property 1: All non-root nodes have $\geq k$ data.

$k = \frac{t}{2}$

- There are 2 ways to create a new node:

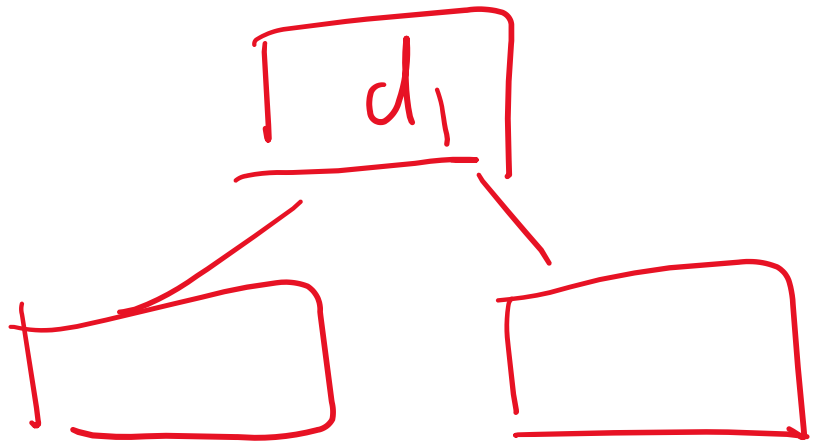
a) new root

b) split ($= k$ data)

Property 2: For each non-leaf node, if it has x data, it has $x+1$ children.

Induction on the # of insertions:

Induction basis: when we have the 1st non-leaf node, the property holds



Assume the property holds after n insertions (induction hypothesis)

Consider the $n+1^{st}$ insertion.

2 types of nodes may be modified:

1° leaf → split (X)

→ no split

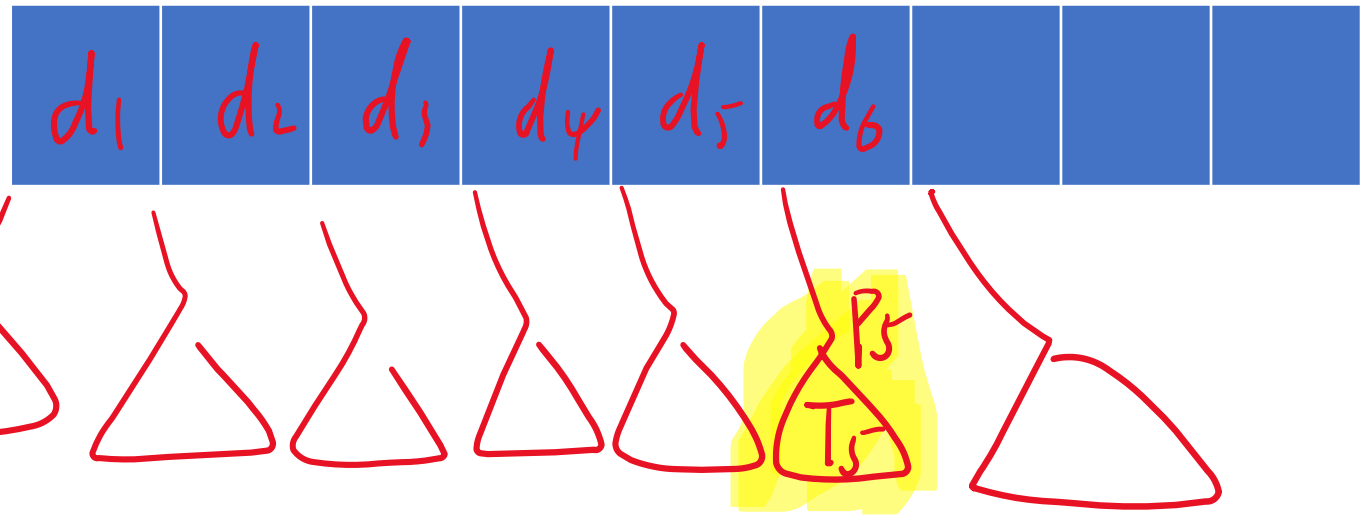
2° internal node → split (✓)

→ no split (✓)

6 data

6+1 children

by induction hypothesis

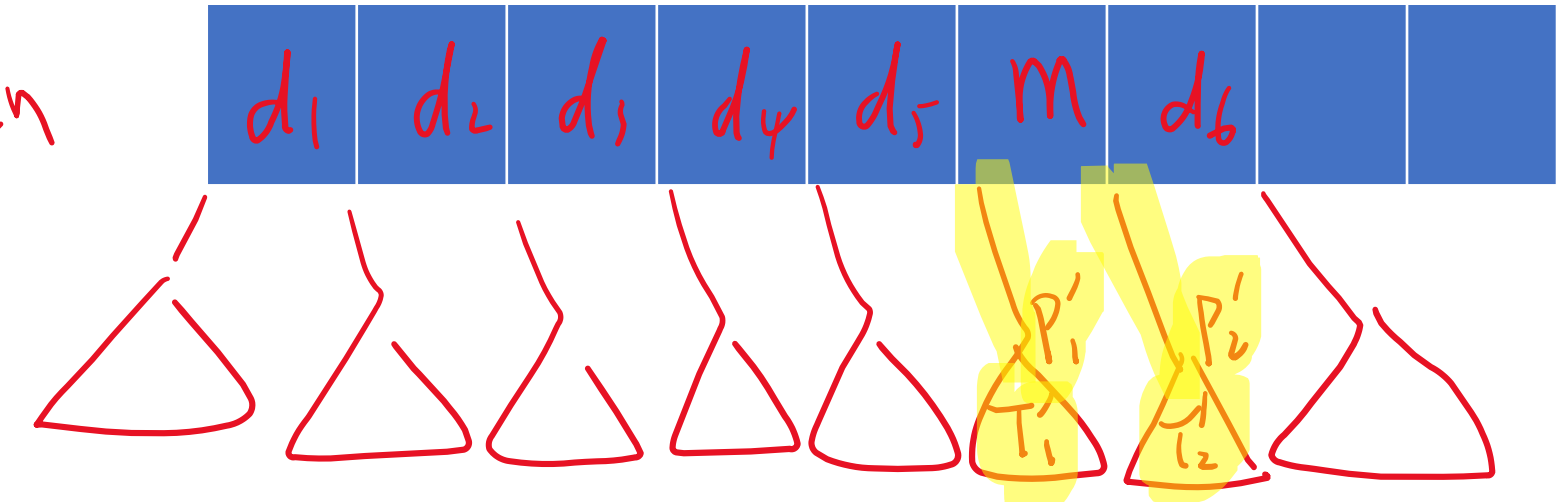


↙ after the n^{th} insertion

In the $n+1^{st}$ insertion, p_5 is full

7 data

7+1 children



Analysis of tree height: n data

1° # of nodes $= O\left(\frac{n}{t}\right)$ (by property 1)

2° For every non-leaf node other than the root, it has $\geq \frac{t}{2}$ children (by properties 1 & 2)

nodes of depth $0 = 1$

nodes of depth $1 \geq 2$

nodes of depth $2 \geq 2\left(\frac{t}{2}+1\right)$

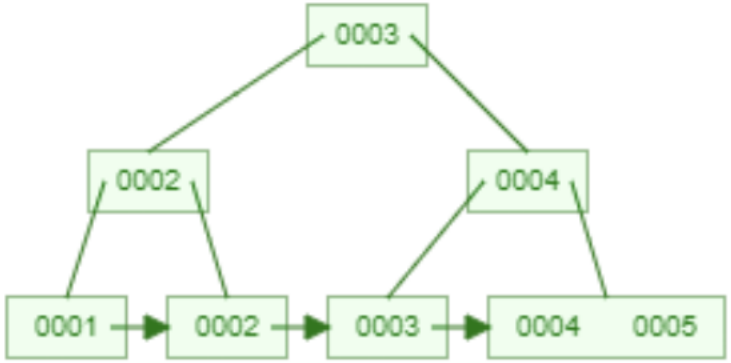
nodes of depth $3 \geq 2\left(\frac{t}{2}+1\right)^2$

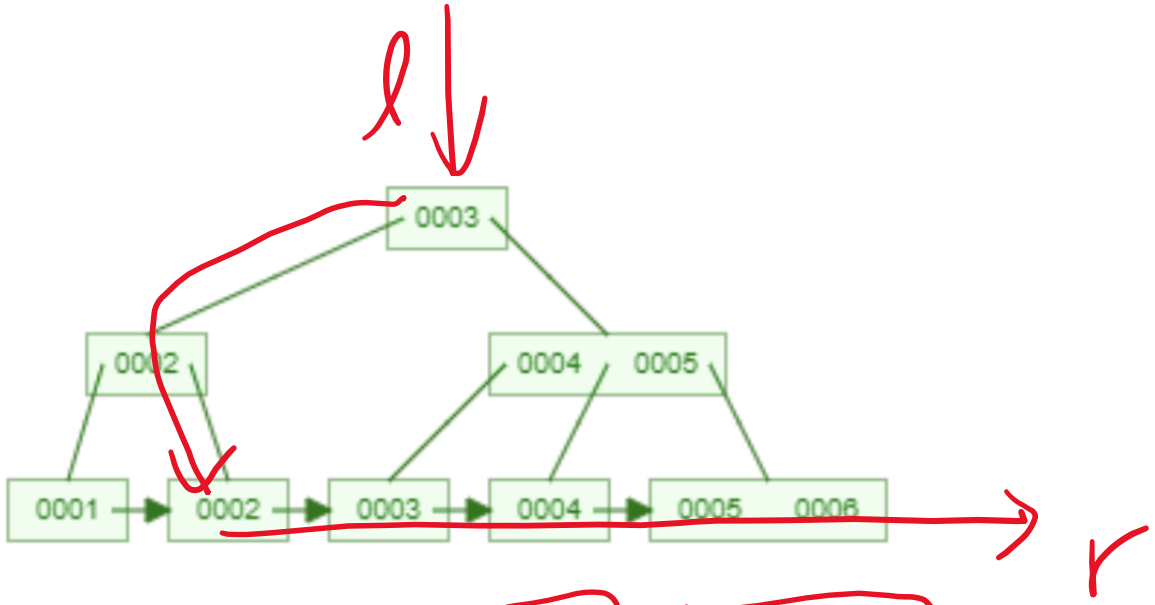
Let h be the height of the tree

$$\#\text{ nodes} \geq 1 + 2 + 2\left(\frac{t}{2}+1\right) + 2\left(\frac{t}{2}+1\right)^2 + \cdots + 2\left(\frac{t}{2}+1\right)^{h-1} = O\left(\frac{n}{t}\right.$$

$$h = O(\log_t n)$$

B+ tree





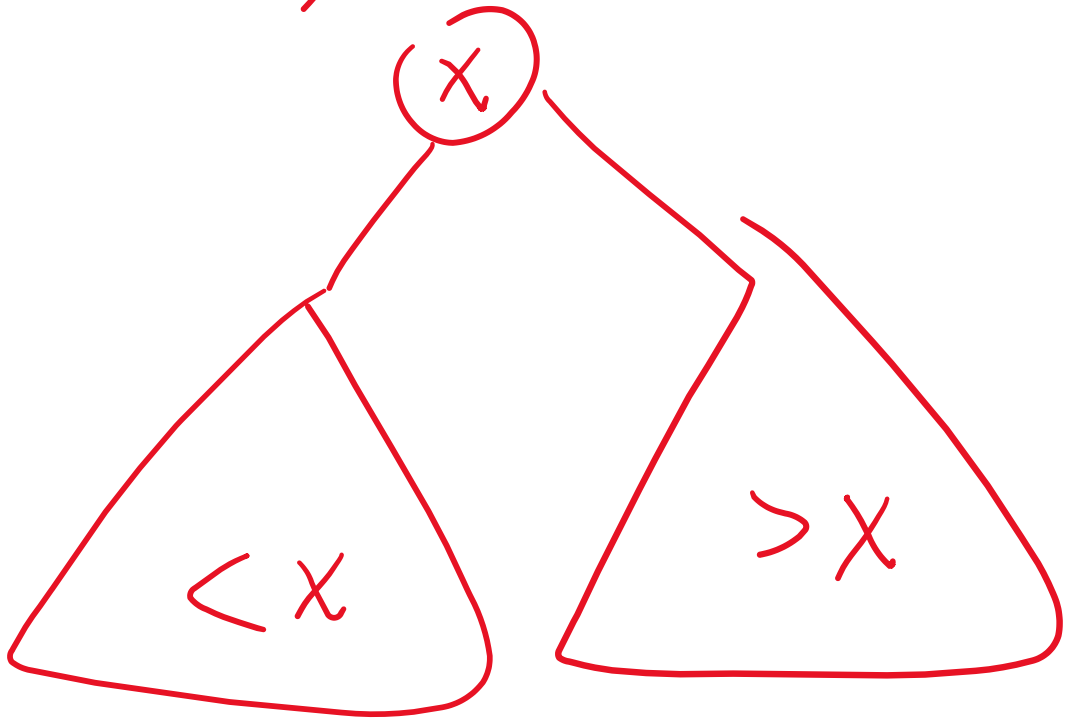
4	5	6

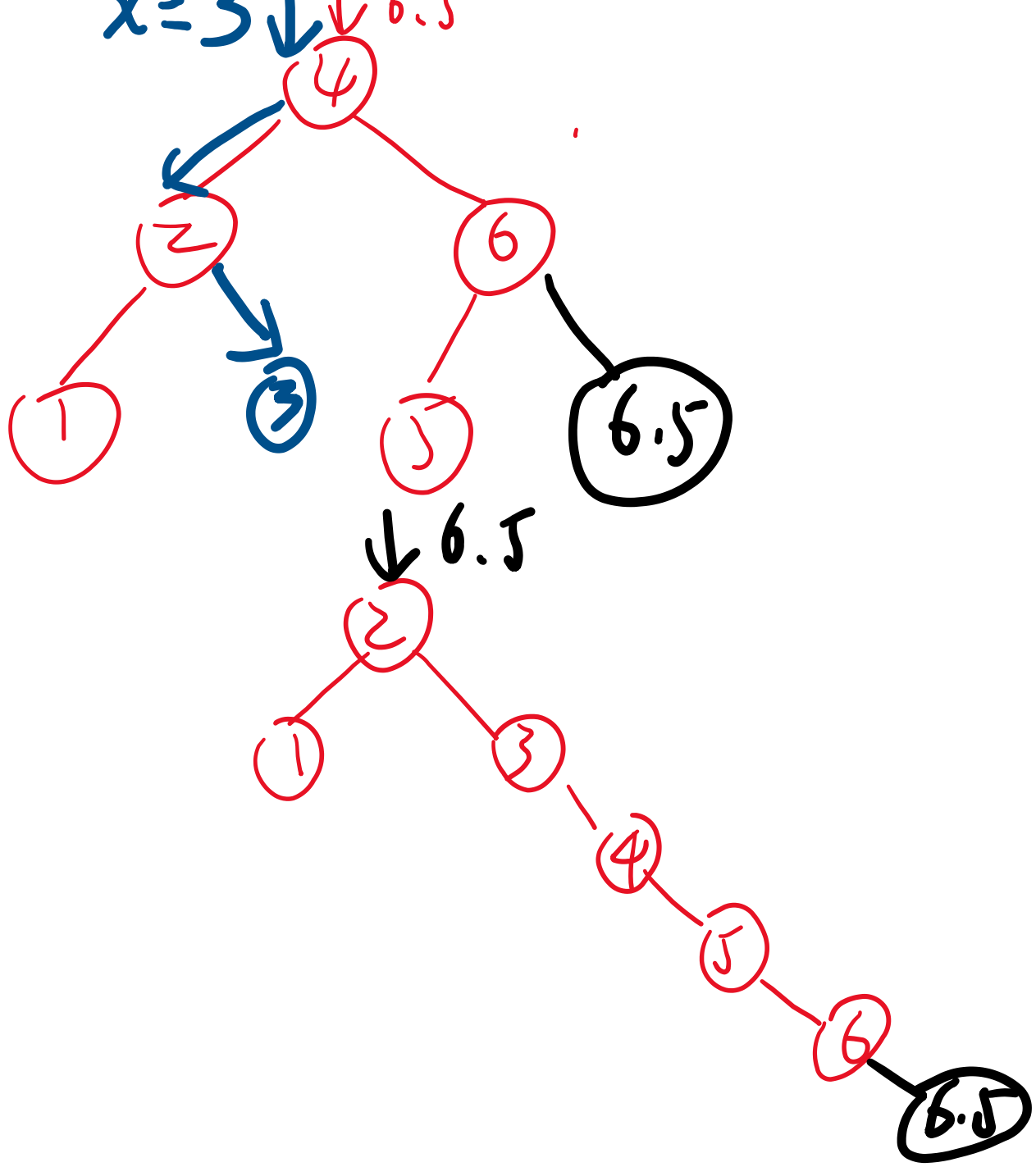
4	5	6

range query (l, r):

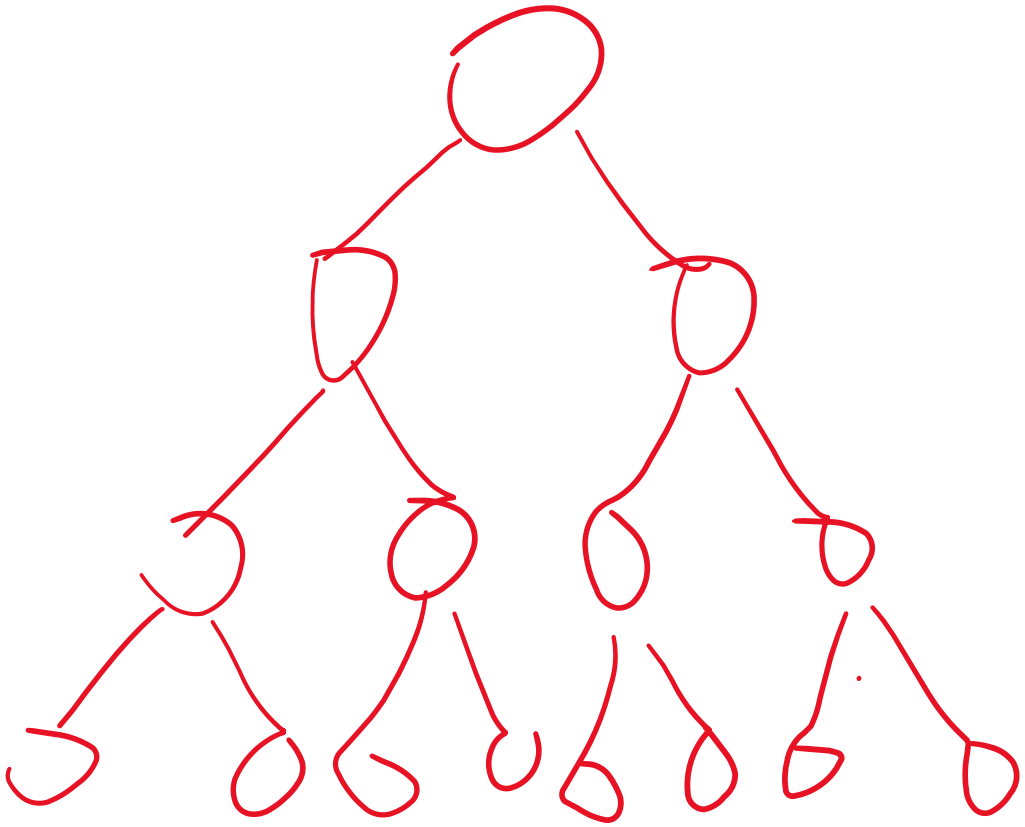
return all data between l and r

Binary Search Tree

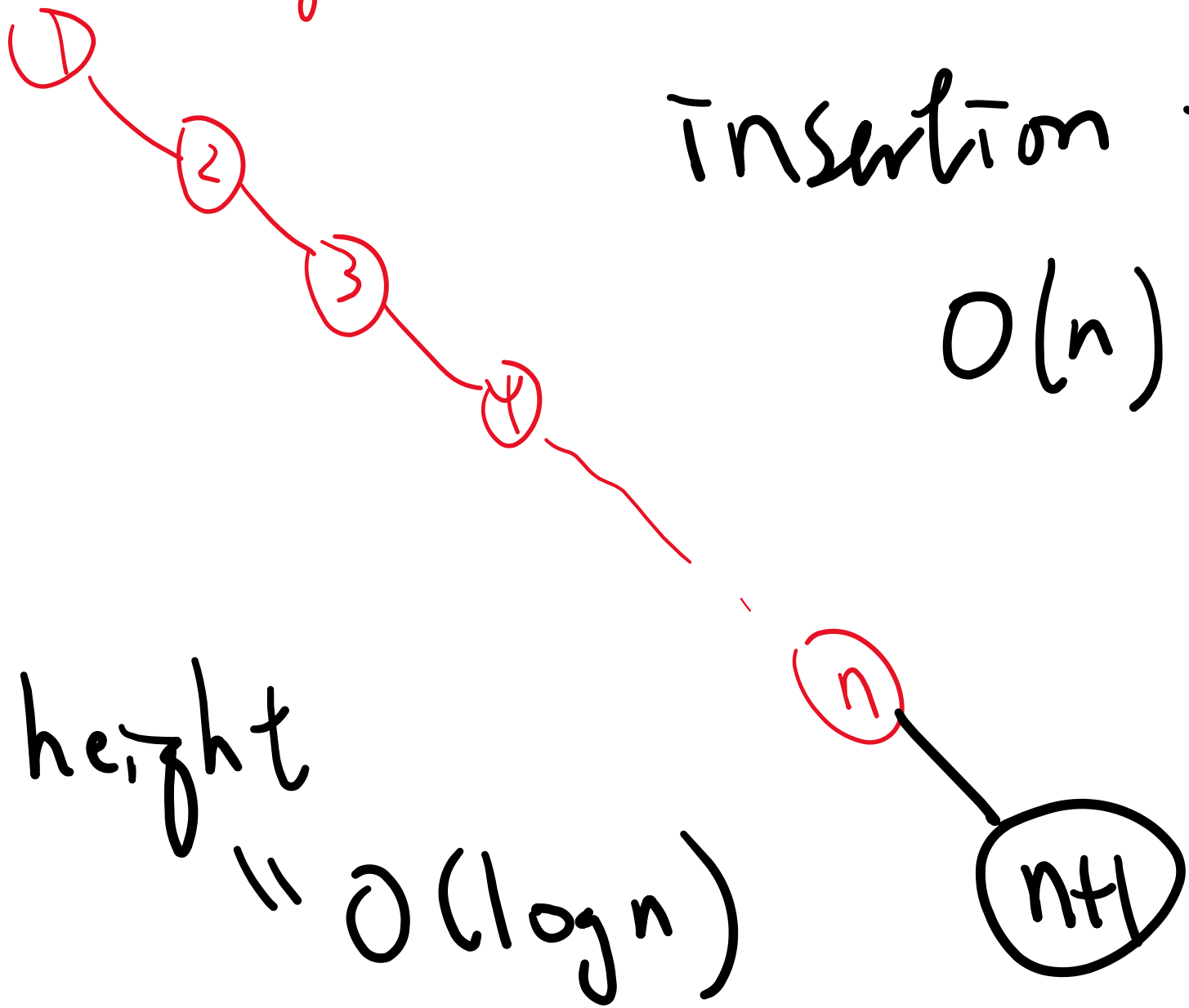




① BST of height $\Theta(\log n)$



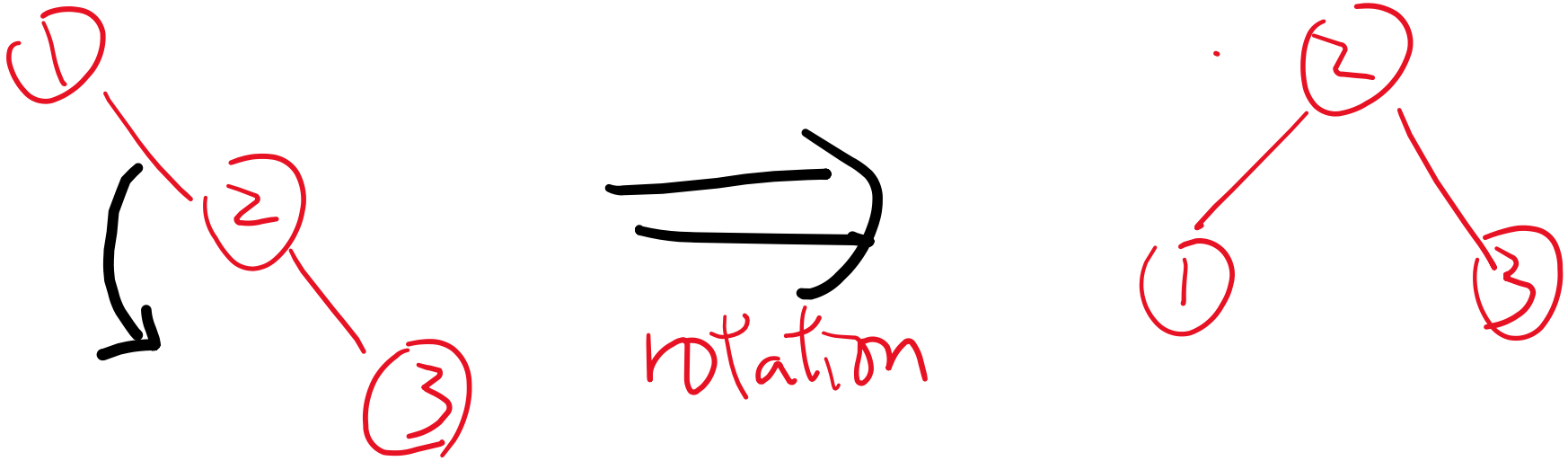
② BST of height n

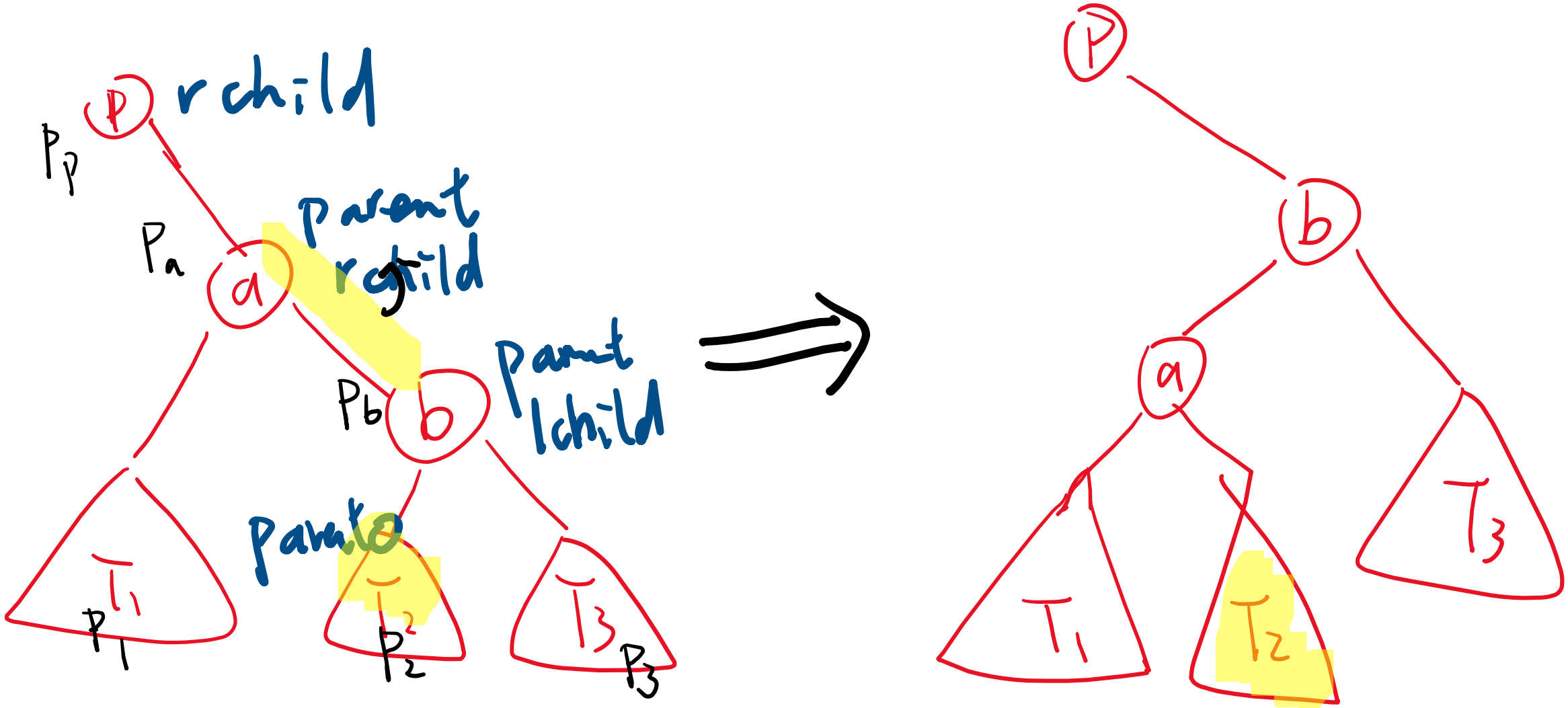


insertion takes O(n) time

Goal: height = O(log n)

Basic Idea: Rotation

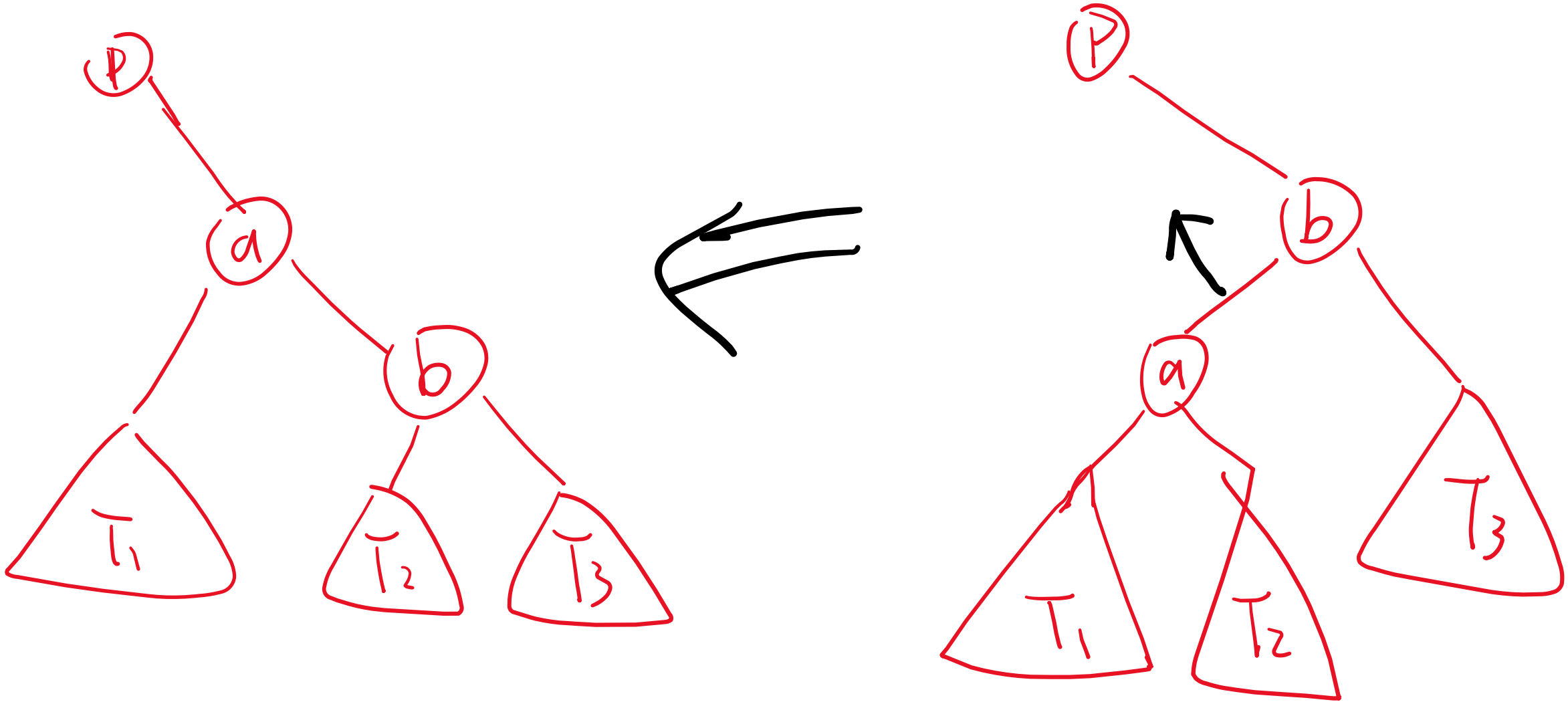




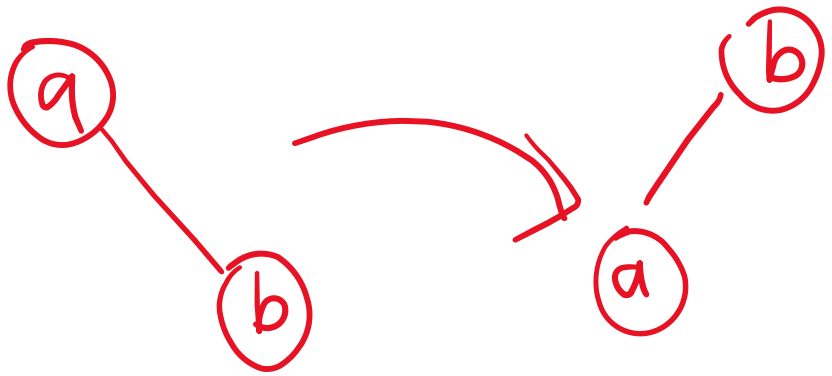
$p_b < p_a$

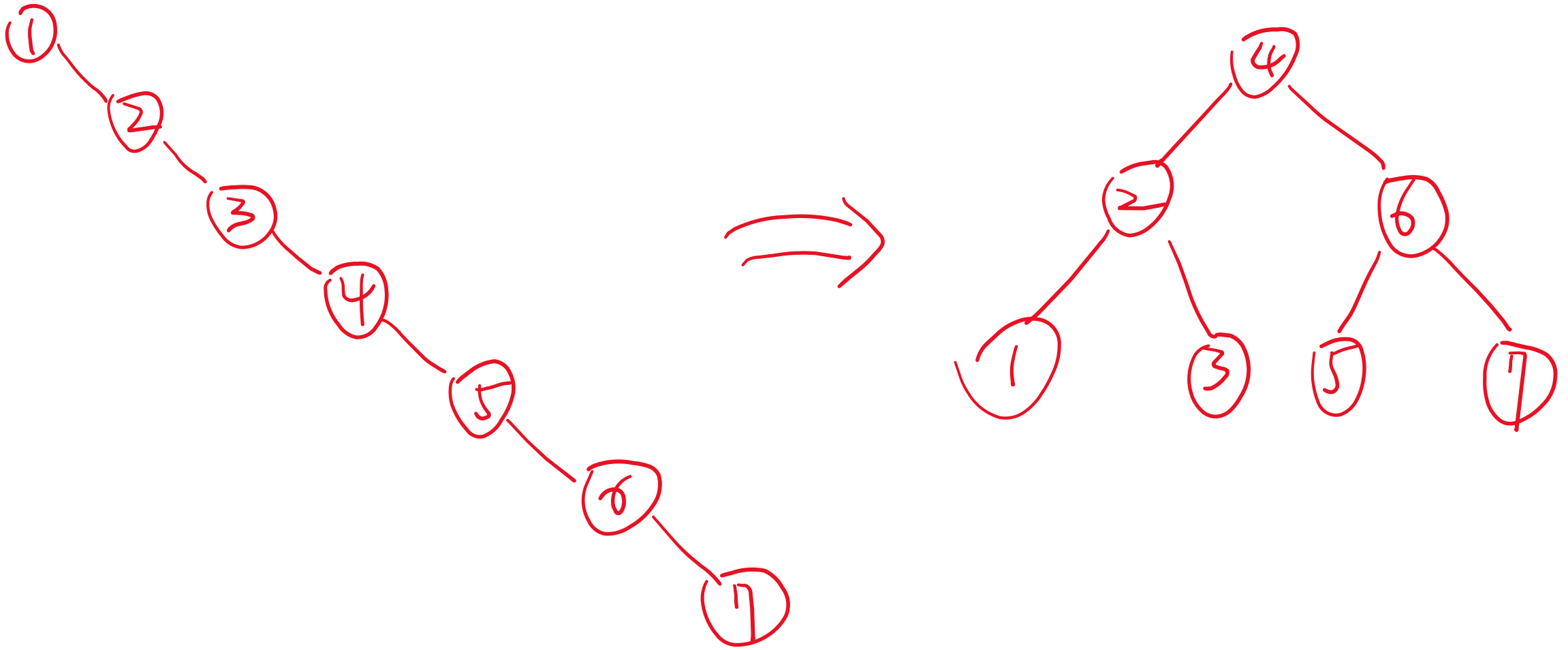
$p_a < p_1$

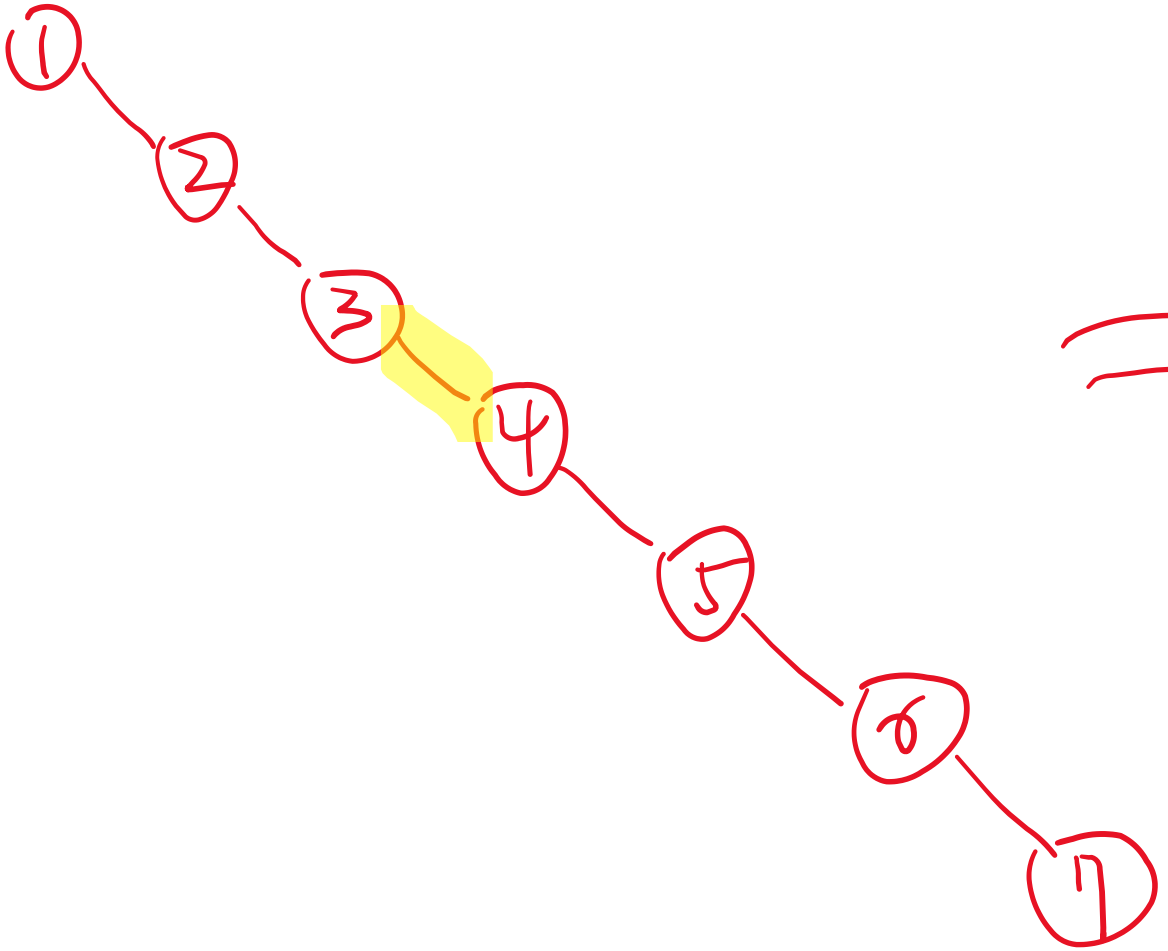
$O(1)$ time.

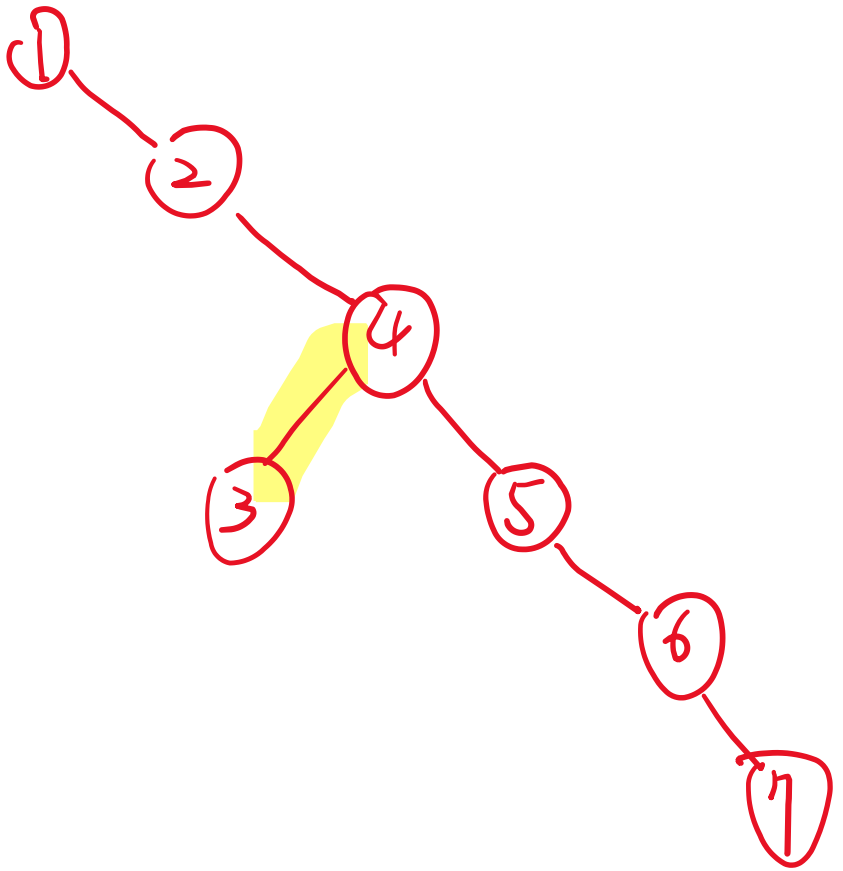

P
a
b
T1
T2
T3
P
b
a
T1
T2
T3

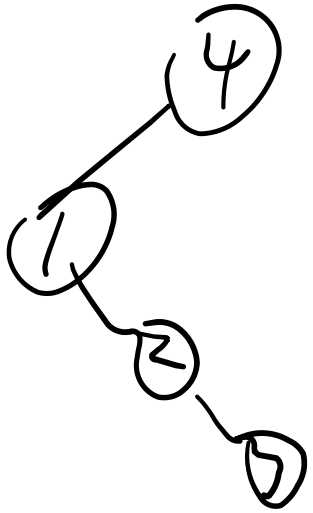
```
struct node {
    int data;
    struct Node* lchild;
    ----          rchild;
    --  --        ------ parent;
};
```

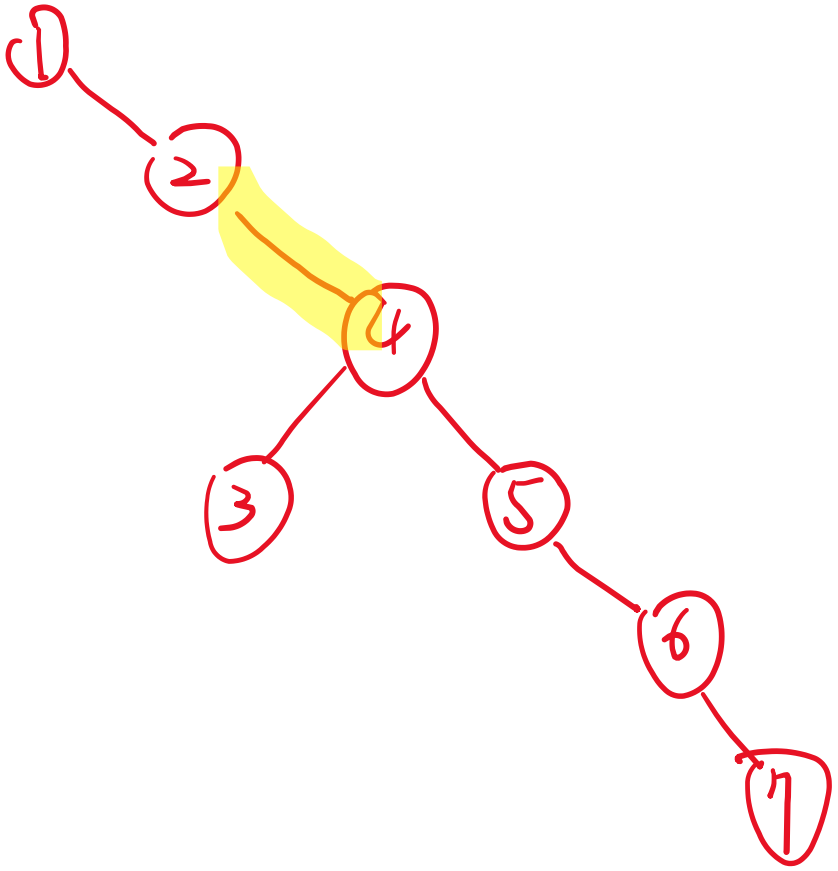


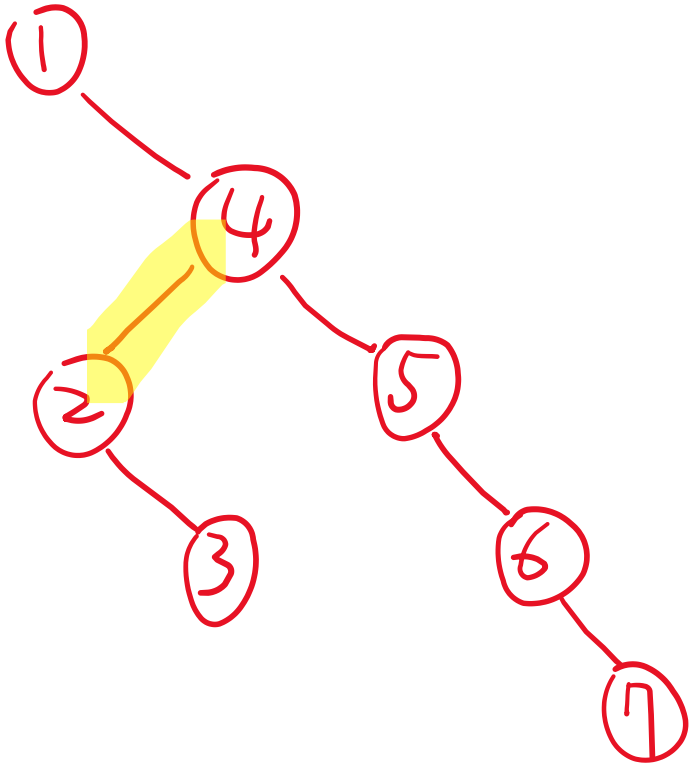

1
2
3
4
5
6
7
4
2
6
1
3
5
7

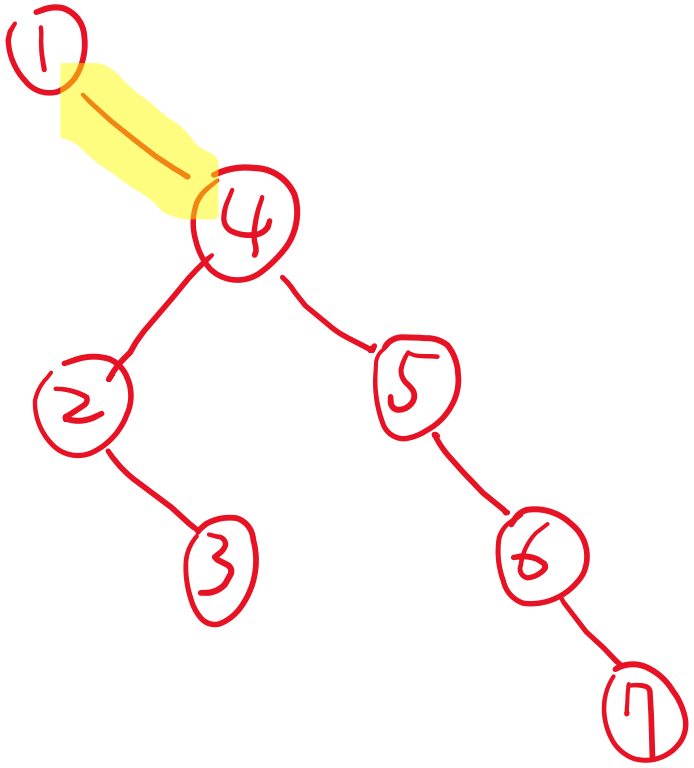

1
2
3
4
5
6
7

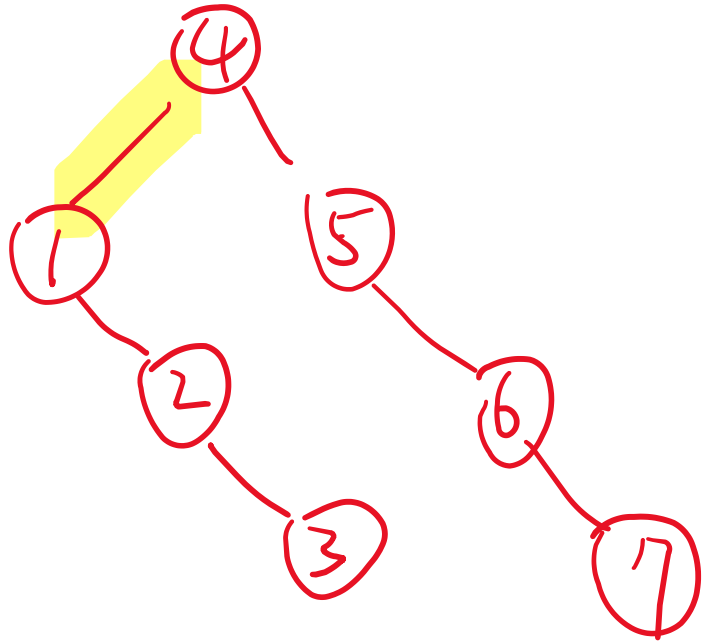

1
2
4
3
5
6
7

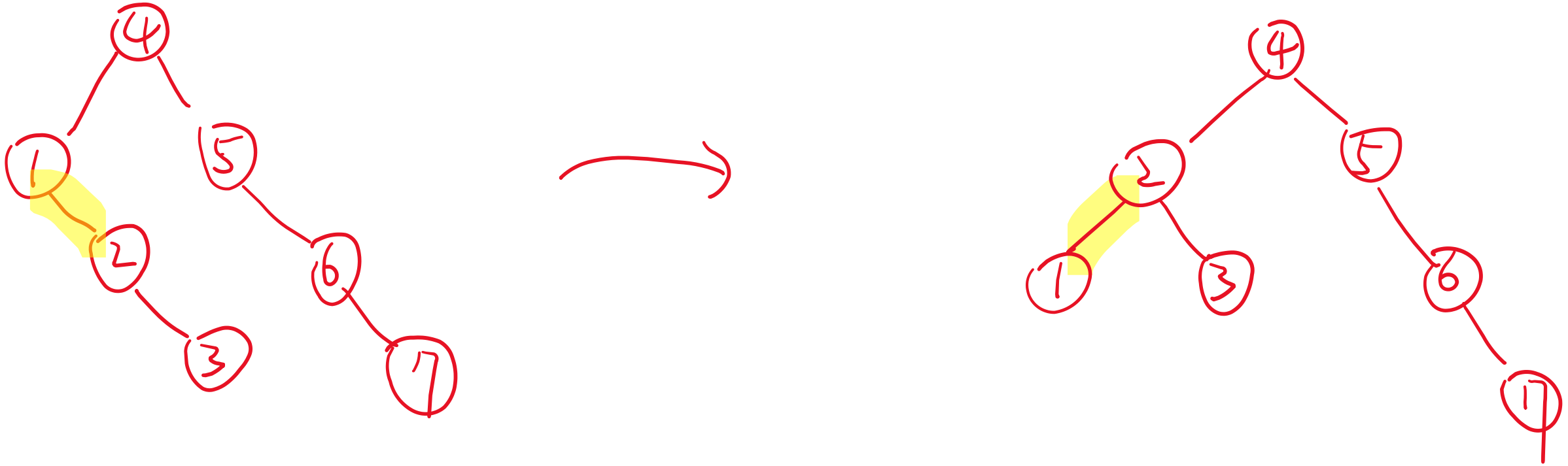

4
1
2
3

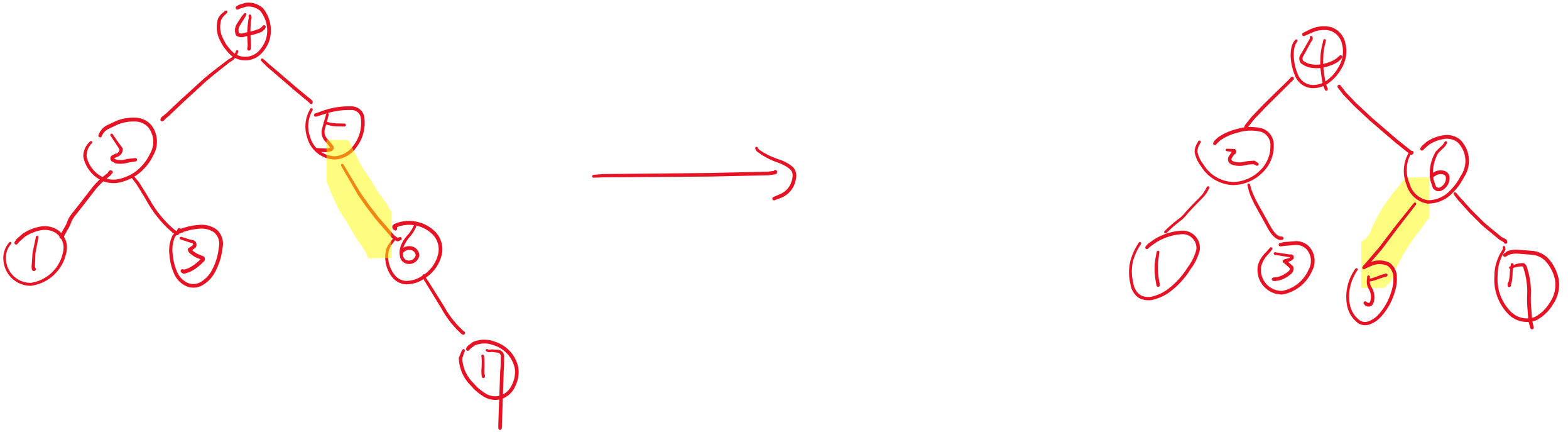

1
2
4
3
5
6
7


1
4
2
5
3
6
7


1
4
2
5
3
6
7

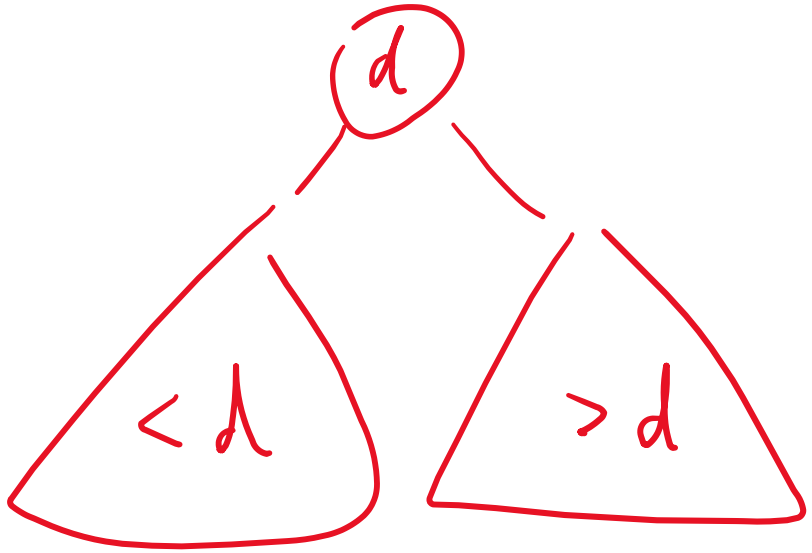

4
1
5
2
6
3
7


4
1
5
2
6
3
7
4
2
5
1
3
6
7


4
2
5
1
3
6
7
4
2
6
1
3
5
7

Treap (randomized BST) (BS)Tree + heap

Observation: ① root in BST is similar to pivot in quick sort



BST

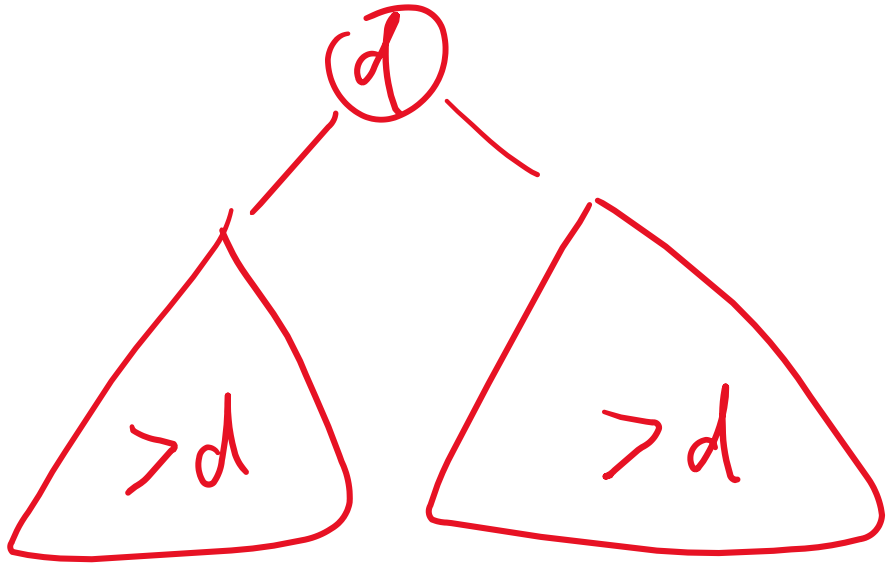
quick sort

② Intuitively, if root is chosen randomly,

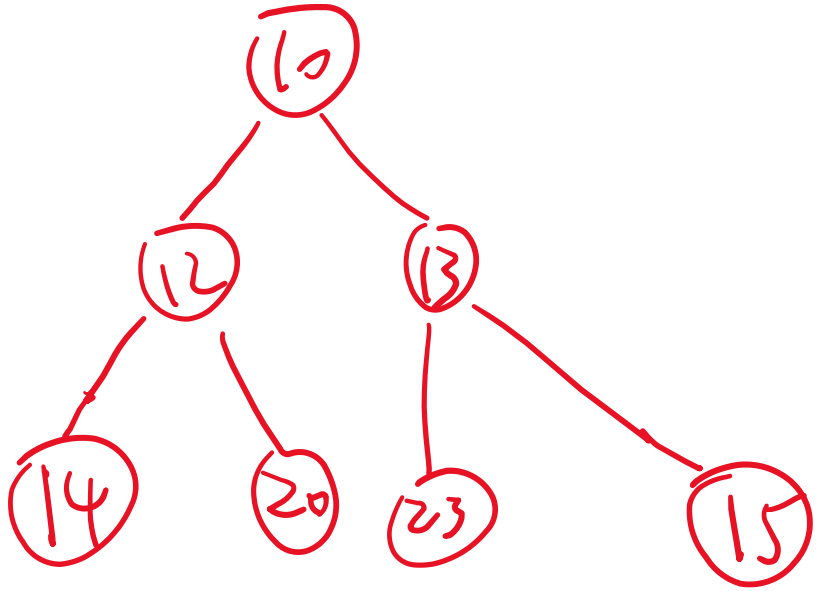
$\approx$ size of right subtree
$\approx$ size of left subtree

heap

root has the smallest priority



(min-heap)

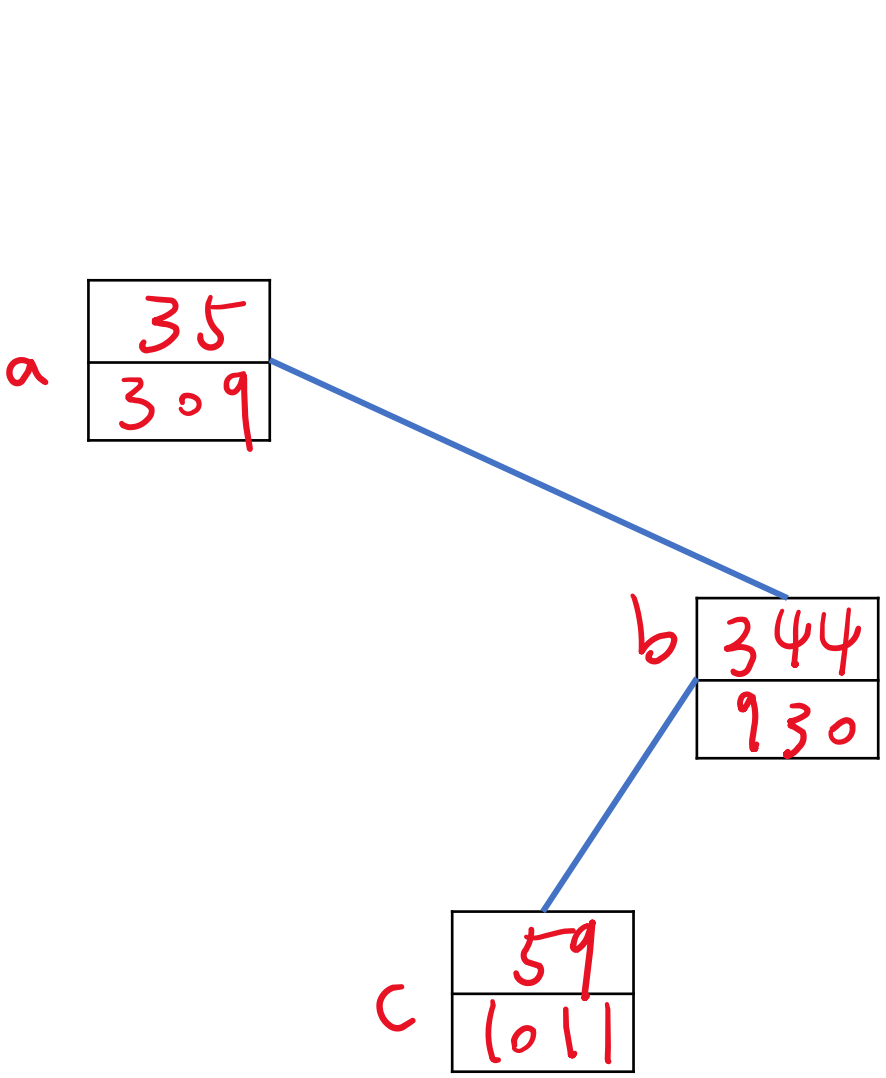


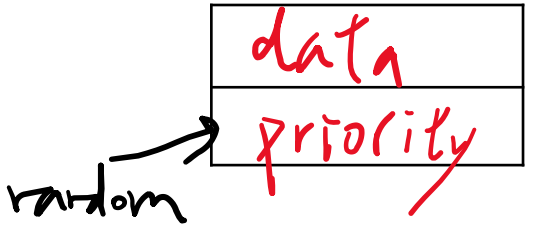
Key Idea: 1° For each data, create a corresponding priority randomly.

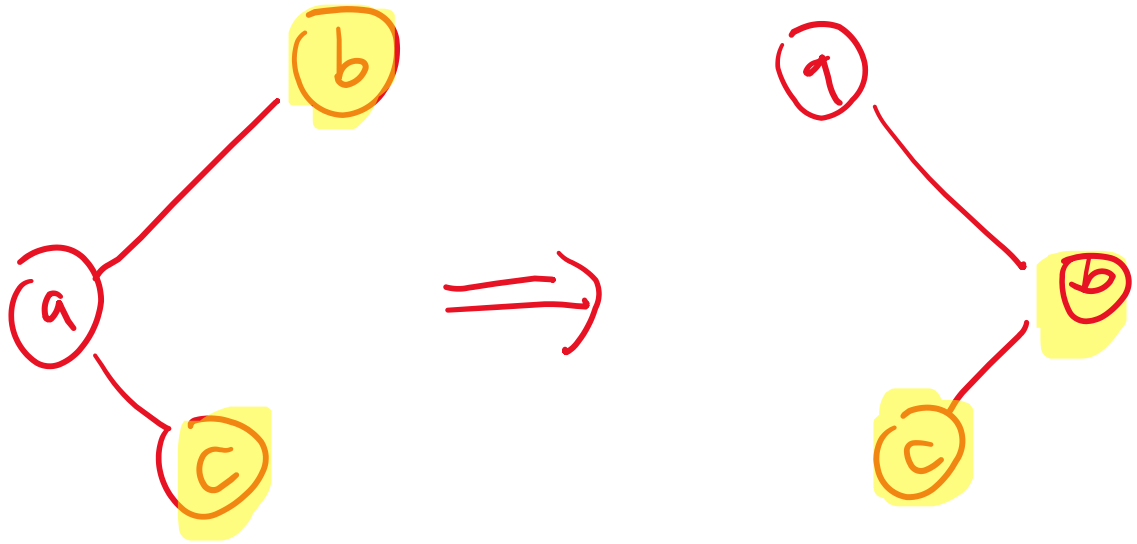
↳ rand()

2° Insert data according to the rule of BST

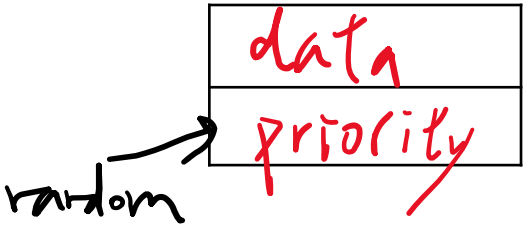
3° Modify the tree according to the rule of heap. (by rotation)

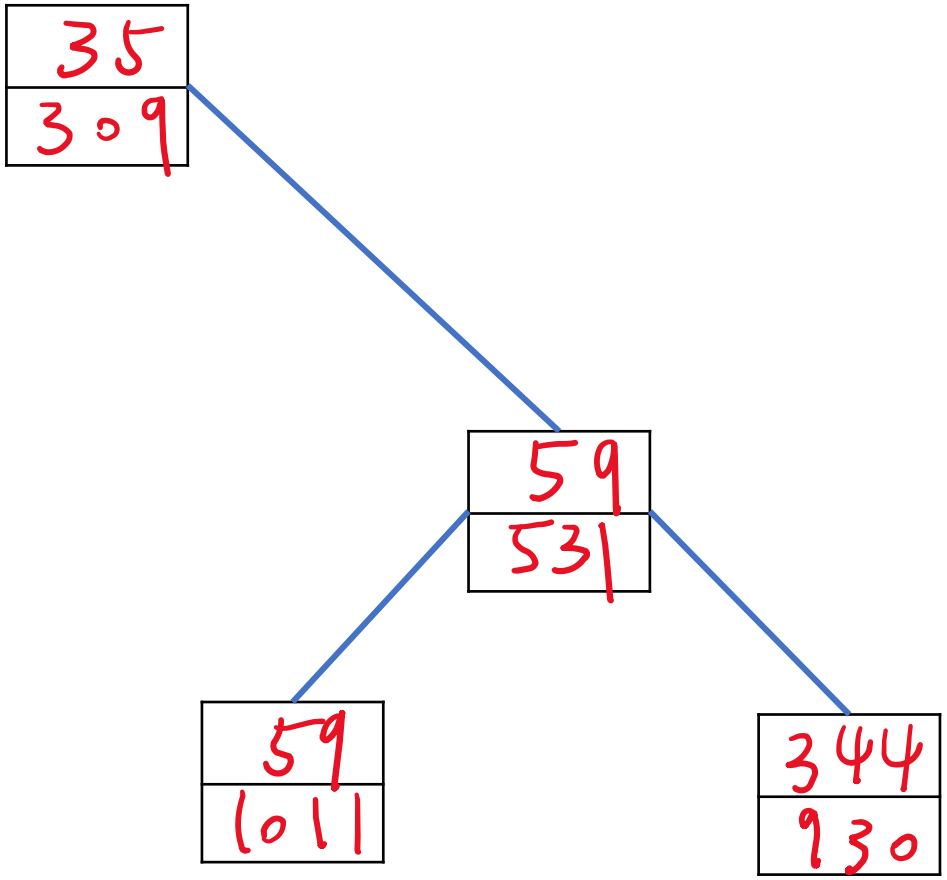

a
35
309
b
344
930
c
59
1011

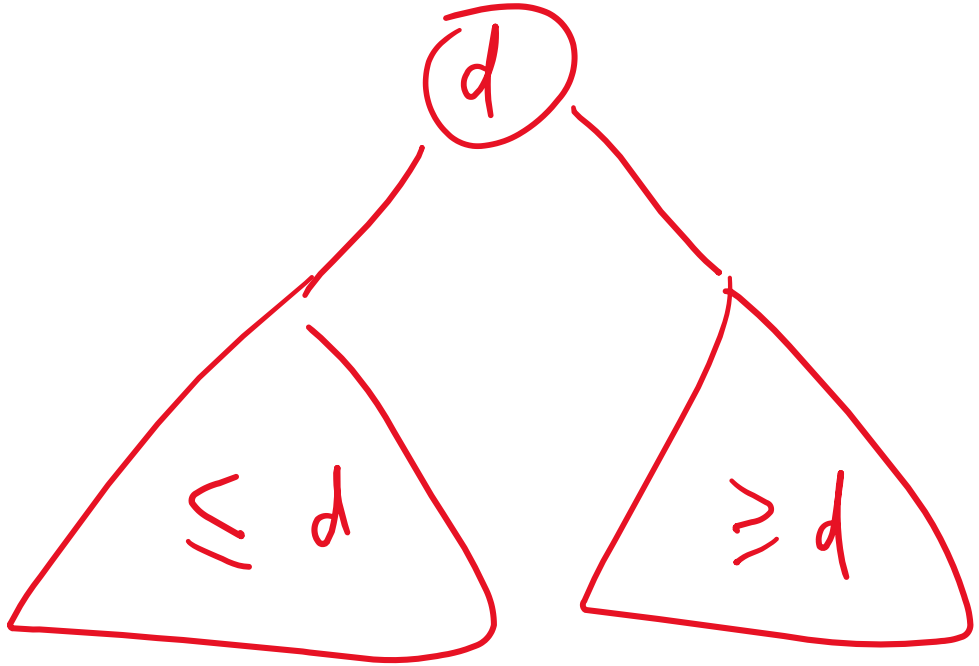

data
priority
random


b
a
c
a
b
c

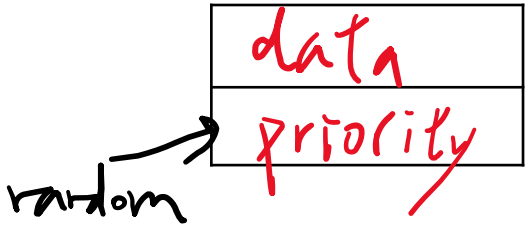
59, 531

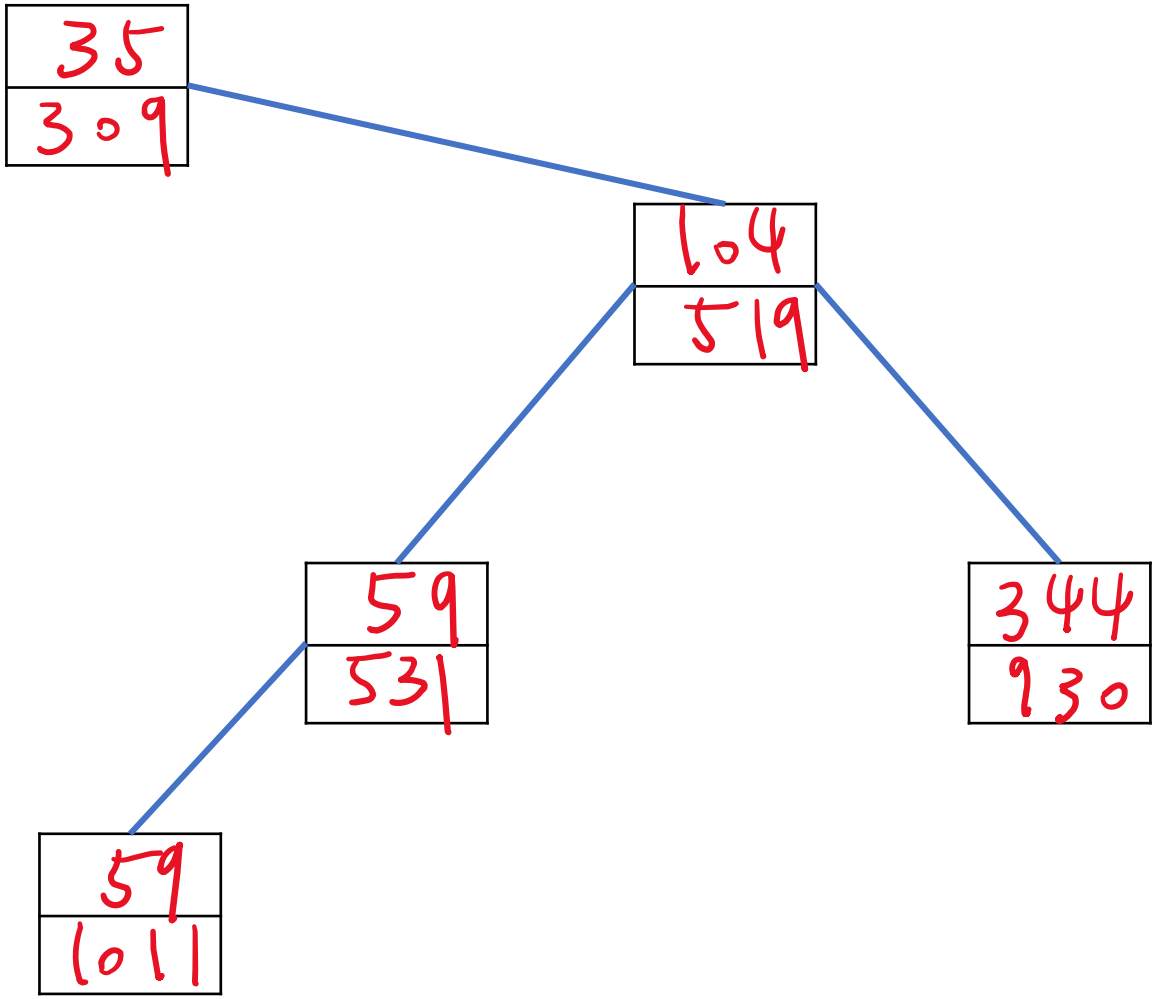






104, 519





Claim: For any node, the expected number of ancestors $= O(\log n)$

(# ancestors = depth of the node in the Tree)

$\Rightarrow$ time to search the node $= O(\log n)$

expected

Proof:

e_i : the i^{th} smallest data

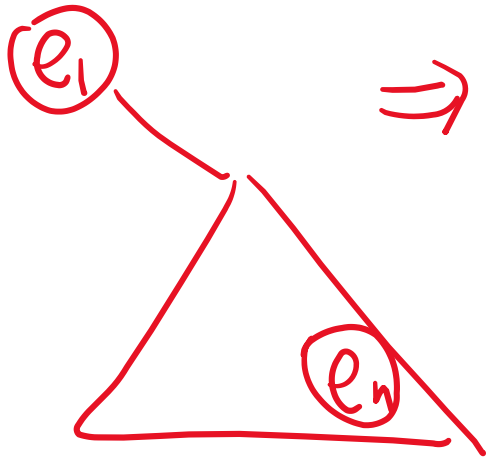
$X_{i,j} = 1$ if e_i is e_j's ancestor

$X_{i,j} = 0$ otherwise

goal : $Pr[X_{i,j} = 1]$ is small

$Pr[X_{1,n} = 1]$

$X_{1,n} = 1 \Rightarrow$



$\Rightarrow e_1$ is root

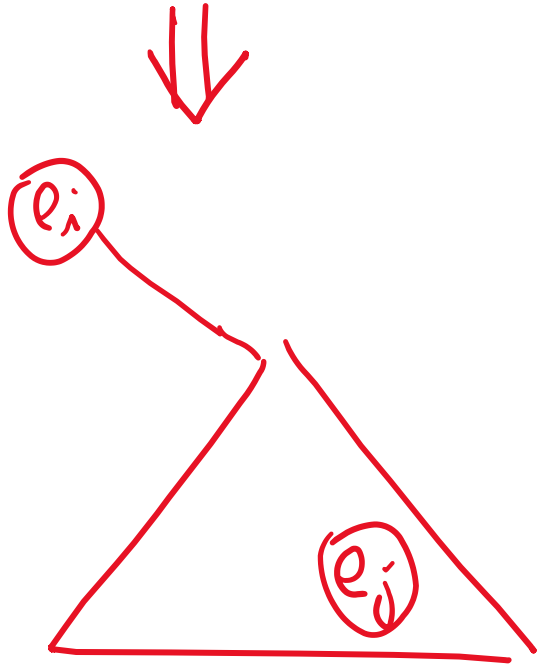
$$Pr[X_{i,j} = 1] \leq \frac{1}{|j-i|+1}$$

$X_{i,n=1}$ $A \Rightarrow B$ e_i is root

$$Pr(A) \leq Pr(B)$$

$$Pr[X_{i,n}=1] \leq Pr[e_i \text{ is root}] = \frac{1}{n}$$

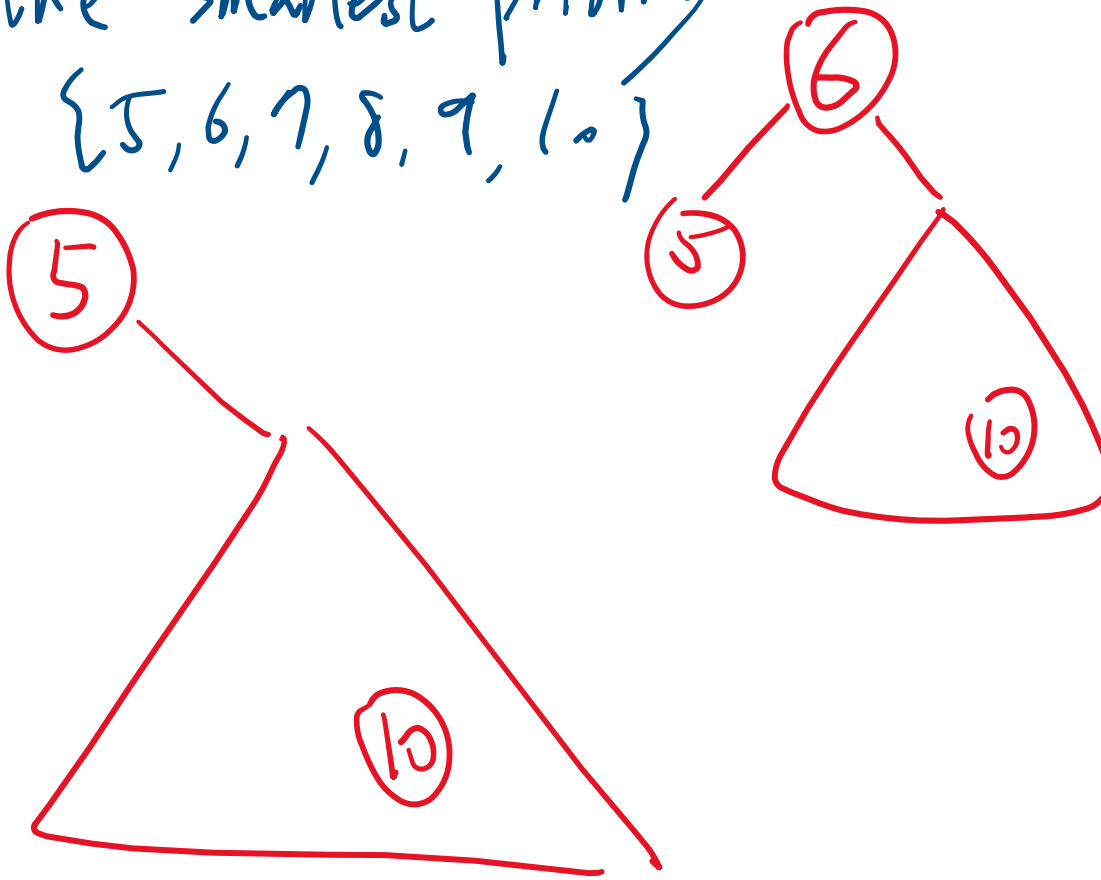
$X_{i,j} = 1 \quad (i < j)$



e_i has the smallest priority among $\{e_i, e_{i+1}, e_{i+2}, \cdots, e_j\}$

$X_{5,10} = 1 \quad (\text{data} = 1, 2, 3, \cdots, n)$

5 has the smallest priority among $\{5, 6, 7, 8, 9, 10\}$



if 6's priority < 5's priority,
⑤ cannot be ⑩'s ancestor

$X_{i,j} = 1$ 时 c_i 与 c_j

$A \Rightarrow B$ e_i has the smallest priority among $[e_i, e_{i+1}, e_{i+2}, \cdots, e_j]$

$$Pr(A) \leq Pr(B)$$

$$Pr[X_{i,j} = 1] \leq Pr[e_i \text{ has the smallest priority among } [e_i, e_{i+1}, e_{i+2}, \cdots, e_j]]$$

$$\frac{1}{j-i+1} = \frac{1}{|U-i+1|}$$

$i > j$ $X_{i,j} = 1$ $X_{i,j} = 1$

$A \Rightarrow B$ e_i has the smallest priority among $[e_j, e_{j+1}, e_{j+2}, \cdots, e_i)$

$$Pr(A) \leq Pr(B)$$

$Pr[X_{i,j} = 1] \leq Pr[e_i$ has the smallest priority among $[e_j, e_{j+1}, e_{j+1}, \cdots, e_i)]$

$$\frac{1}{i-j+1} \leq \frac{1}{|j-i|+1}$$

Expected # ancestors of e_i

$$= E[X_{1,i} + X_{2,i} + \cdots + X_{i-1,i} + X_{i+1,i} + X_{i+2,i} + \cdots + X_{n,i}]$$

$$= E[X_{1,i}] + E[X_{2,i}] + \cdots + E[X_{i-1,i}] + E[X_{i+1,i}] + E[X_{i+2,i}] + \cdots + E[X_{n,i}]$$

$$= Pr[X_{1,i}=1] + Pr[X_{2,i}=1] + \cdots + Pr[X_{i-1,i}=1] + Pr[X_{i+1,i}=1] + Pr[X_{i+2,i}=1] + \cdots + Pr[X_{n,i}=1]$$

$$\leq \frac{1}{n} \cdots + \frac{1}{4} + \frac{1}{3} + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = O(\log n)$$

Duplicated Data

① counter

② use multiple nodes to store duplicated data.

Key - value pair (key is unique)

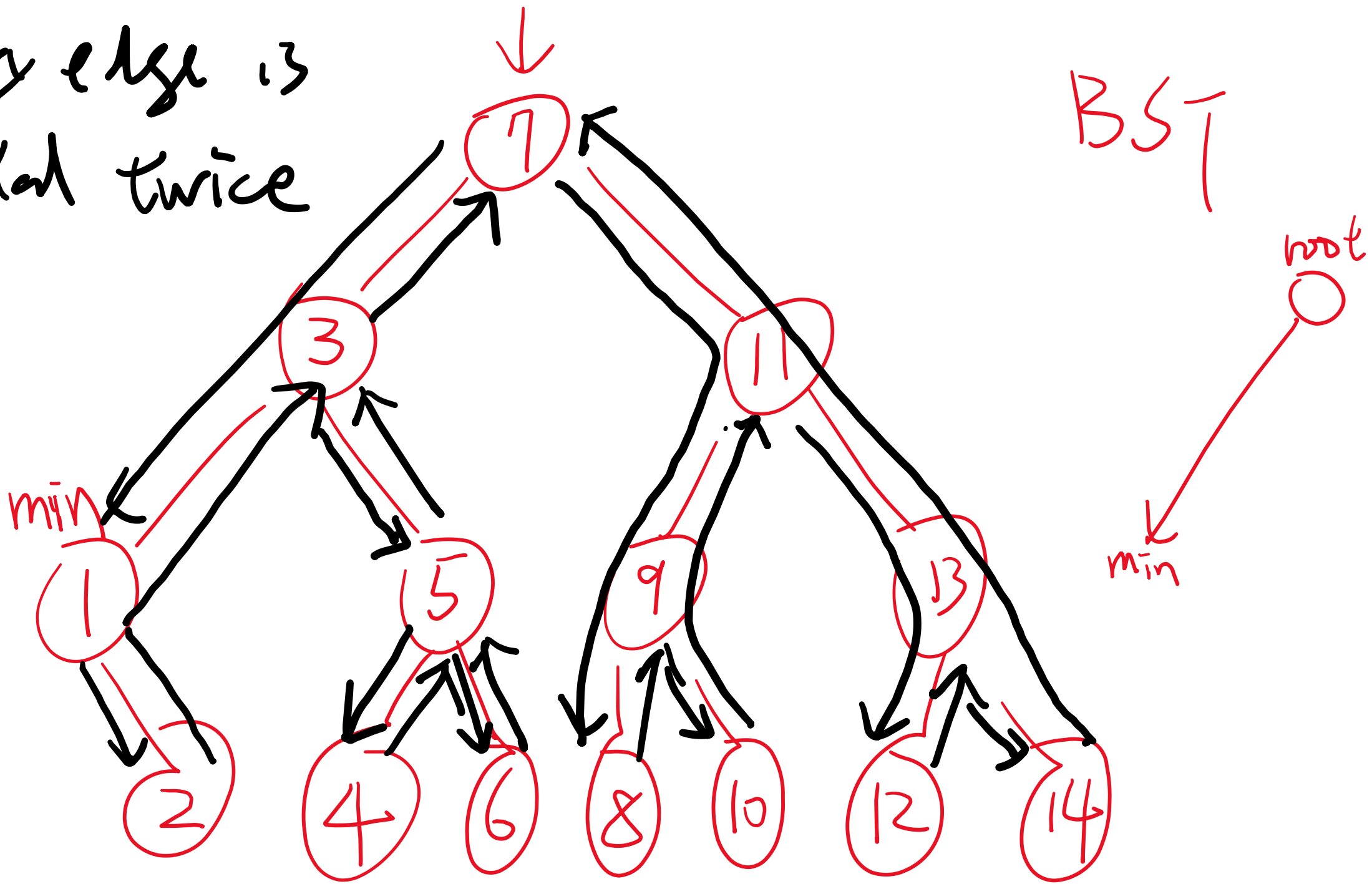
e.g. key = student-id insert/search base on keys

value = score

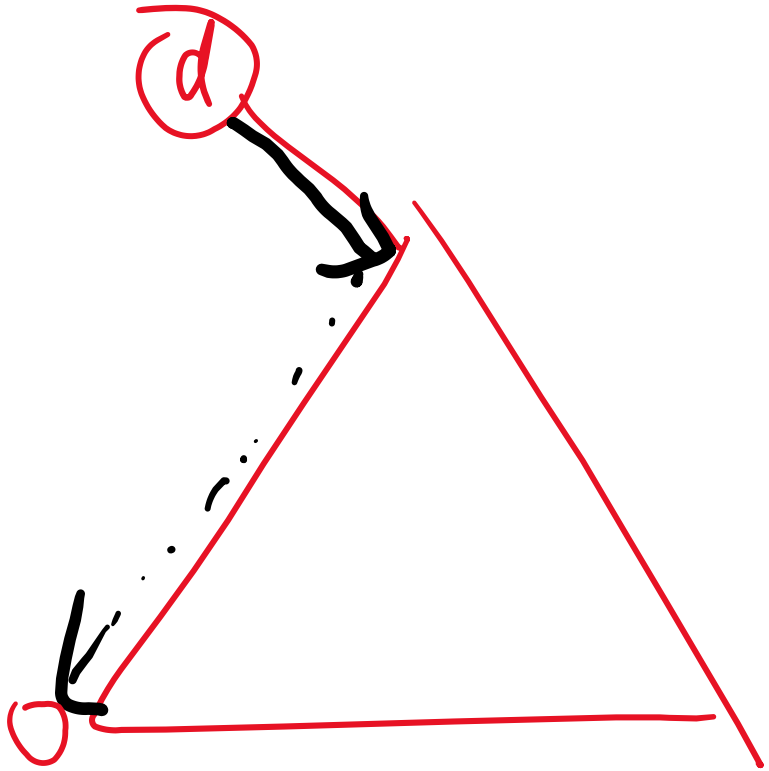
C++ STL
map

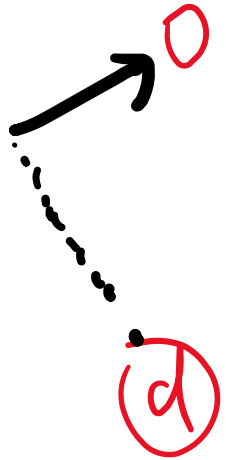
```
struct node {
    int key
    int value
    ;  // pointers
}
```

every edge is visited twice



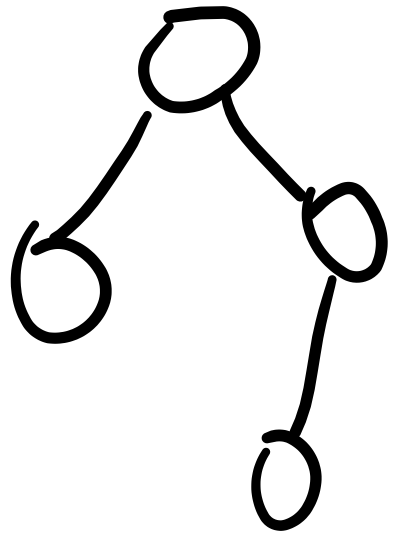
BST





Time complexity to find the next data

$= O(\text{tree height})$



n nodes

⇓

n-1 edges in BST

Sorting $\Rightarrow O(\#\text{edges} \times 2)$

$= O(n)$

① ~~midterm - no hash-table~~

(11/4)

③ 11/24 midterm : hash-table.

(11/17 review)

Hash Table

(Assumption: n is known in advance)

Goal: $O(n)$ space

insert in $O(1)$ time

search in $O(1)$ avg. time.

General Steps:

1. Create an array of size M
2. Create a (random) hash function h
3. insert/search x at arr[h(x)]

(arr[i] is a list that stores all insert data x with $h(x) = i$)

Time Complexity: (assume $h(x)$ can be computed in $O(1)$ time for all x)

1° insert = $O(1)$

(∵ $h(x)$ takes $O(1)$ time

insert x to $arr[h(x)] \Rightarrow O(1)$ time)

2° search(x) = $O($length of $arr[h(x)])$

$= O(n)$

Example 1

$n = 10$, $h(x) = x$

$d_i = i \times 10000$

M : array size $= 10000 \times n$

drawback : huge space

Example 2

$n = 10$ $\quad h(x) = x \% n$

$d_i = i \times 10$

10

20

30

i

100

drawback: lots of collisions

Random hash function

- For any fixed hash function h,
  there is a collection of data that has the same output.

$\Rightarrow$ we should construct h randomly

Desired Property:

$\forall x, y$ if $x \neq y$, then

$$\Pr_h[h(x) = h(y)] \leq \frac{1}{M}$$

(the prob. that x and y collide under h is small)

Example 3

$h(x) = (x \cdot r) \% M$, r is uniformly randomly chosen from $\{0, 1, 2, 3, \cdots, M-1\}$

$M = 10$,

$d_i = i \times 10 + 1$

$h(x) = (r x) \% 10$

$r = 3 \Rightarrow h(x) = (3x) \% 10$	$r = 6 \Rightarrow h(x) = (6x) \% 10$
$h(11) = 3$	$h(11) = 6$
$h(21) = 3$	$h(21) = 6$
⋮	⋮
$h(101) = 3$	$h(101) = 6$

if $X \equiv y \pmod{M}$

$\Rightarrow r \cdot X \equiv r \cdot y \pmod{M}$

$h(x) = (r \cdot x) \% M$ does not have the desired property

consider $M = 10$, $x = 20$, $y = 30$

$$Pr[h(20) = h(30)] = 1$$

r is chosen uniformly randomly from $\{0, 1, 2, \ldots, 9\}$

$h(x) = (x + r) \% M$

if $x \equiv y \pmod M$

$x + r \equiv y + r \pmod M$

$h(x) = x \% r$ (r chosen from $1 \sim M$)

if $x \neq y$

$X = M!$ $y = 2M!$

$P_r[h(x) = h(y)] = 1$ ⟸

Assume X can be written as $(x_1\ x_2)_{10}$

$0 \le x_1, x_2 \le 9$

$$h(x) = (x_1 \cdot r + x_2) \% M$$

$r = 3 \quad h(21) = (2 \cdot 3 + 1) \% 10 = 7$

$$h(X) = (x_1 \cdot r_1 + x_2 \cdot r_2) \% M$$

r_1, r_2 chosen uniformly randomly from $0 \sim M-1$

Example

mpht $d_i = 5 + 10i$ $M = 10$

$r_1 = 3$, $r_2 = 7$

$h(15) = (1 \times 3 + 5 \times 7) \% 10 = 8$

$(5 \times 0) \% 10 = 0$

$(5 \times 1) \% 10 = 5$

$(5 \times 2) \% 10 = 0$

$(5 \times 3) \% 10 = 5$

$(5 \times 4) \% 10 = 0$

$\vdots \qquad \vdots$

$(5 \times 9) \% 10 = 5$

$X_1 = 5$

$r = 0 \sim 9$

$(2\times0)\%10=0$

$(2\times1)\%10=2$

$(2\times2)\%10=4$

$(2\times3)\%10=6$

$(2\times4)\%10=8$

$(2\times5)\%10=0$

$(2\times6)\%10=2$

$(2\times7)\%10=4$

$(2\times8)\%10=6$

$(2\times9)\%10=8$

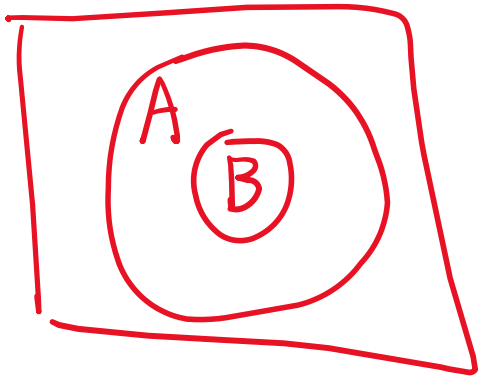
$X_1=2$

$r=0\sim9$

$X = 5 \qquad y = 55$

$A \Leftarrow B$

$$Pr[h(x) = h(y)] = \frac{1}{2}$$



$h(x) = (5 \times r_2) \% 10$

$Pr(A) \geq Pr(B)$

$h(y) = (5 \times r_1 + 5 \times r_2) \% 10$

$h(x) = h(y) \Leftrightarrow 5 \times r_1 \% 10 = 0$

($h(x) = h(y)$: A; $5 \times r_1 \% 10 = 0$: B)

$$Pr[5 \cdot r_1 \% 10 = 0] = \frac{1}{2}$$

$(2\times0)\%11=0$

$(2\times1)\%11=2$

$(2\times2)\%11=4$

$(2\times3)\%11=6$

$(2\times4)\%11=8$

$(2\times5)\%11=10$

$(2\times6)\%11=1$

$(2\times7)\%11=3$

$(2\times8)\%11=5$

$X_1=2$

$r_1=0\sim10$

$M=11$

$(2\times9)\%11=7$

$(2\times10)\%11=9$

$(3 \times 0) \% 11 = 0$

$(3 \times 1) \% 11 = 3$

$(3 \times 2) \% 11 = 6$

$(3 \times 3) \% 11 = 9$

$(3 \times 4) \% 11 = 1$

$(3 \times 5) \% 11 = 4$

$(3 \times 6) \% 11 = 7$

$(3 \times 7) \% 11 = 10$

$(3 \times 8) \% 11 = 2$

$X_1 = 3$

$r_1 = 0 \sim 10$

$M = 11$

$(3 \times 9) \% 11 = 5$

$(3 \times 10) \% 11 = 8$

Assume every data has 4 bytes

$\Rightarrow X \rightarrow (X_1, X_2, X_3, X_4)$

$0 \le X_i \le 255$

M: ① $M \ge n$

② M prime

③ $M > 255$

$\exists$ prime between $n \sim 2n$

$\Rightarrow M \le 2n$

$$h_M(X) = (X_1 r_1 + X_2 r_2 + X_3 r_3 + X_4 r_4) \% M$$

r_i chosen from $0 \sim M-1$

Claim: If $x \neq y$, $\Pr[h_M(x) = h_M(y)] \leq \frac{1}{M}$

$X \to x_1 x_2 x_3 x_4$

$y \to y_1 y_2 y_3 y_4$

without loss of generality

$$h_M(x) = h_M(y) \iff \sum_i x_i r_i \equiv \sum_i y_i r_i \pmod{M}$$

w.l.o.g., assume $x_1 \neq y_1$

$$(x_1 - y_1) r_1 \equiv \underbrace{\sum_{i=2}^{4} (y_i - x_i) r_i}_{A} \pmod{M}$$

$$\iff (x_1 - y_1) r_1 \equiv A \pmod{M}$$

$$Pr[h_M(x) = h_M(y)] = Pr[(x_1 - y_1) r_1 \equiv A \pmod{M}]$$

$x_1 - y_1 \neq 0$

$(x_1 - y_1)$: fixed (e.g., 3)

r_1: random

A: fixed (e.g., 2)

M (e.g., 11)

$$Pr[3r \,\%\, 11 = 2] = \frac{1}{11}$$

r is chosen randomly from 0 to 10

uniformly

$$= \frac{1}{M}$$

($\because$ M is prime)

$M = 11$ $\quad x_1 = 34$, $y_1 = 23$

$$x_1 - y_1 = 11$$

$$11 \bmod M = 0$$

$0 \le x_1, y_1 \le 255$ $\quad x_1 - y_1 \neq 0$

if $M > 255$

$|x_1 - y_1|$ cannot be a multiple of M

$(x_1 - y_1) \bmod M \neq 0$

$(3 \times 0) \% 11 = 0$

$(3 \times 1) \% 11 = 3$

$(3 \times 2) \% 11 = 6$

$(3 \times 3) \% 11 = 9$

$(3 \times 4) \% 11 = 1$

$(3 \times 5) \% 11 = 4$

$(3 \times 6) \% 11 = 7$

$(3 \times 7) \% 11 = 10$

$(3 \times 8) \% 11 = 2$

$X_1 \cdot y_1 = 3$

$r_1 = 0 \sim 10$

$M = 11$

$(3 \times 9) \% 11 = 5$

$(3 \times 10) \% 11 = 8$

$(0 \times 0) \% 11 = 0$

$(0 \times 1) \% 11 = 0$

$(0 \times 2) \% 11 = 0$

$(0 \times 3) \% 11 = 0$

$(0 \times 4) \% 11 = 0$

$(0 \times 5) \% 11 = 0$

$(0 \times 6) \% 11 = 0$

$(0 \times 7) \% 11 = 0$

$(0 \times 8) \% 11 = 0$

$X_1 \cdot y_1 = 0$

$r_1 = 0 \wedge 10$

$M = 11$

$(0 \times 9) \% 11 = 0$

$(0 \times 10) \% 11 = 0$

if $(x_1 - y_1) \bmod M = 0$,

then $(x_1 - y_1) r_1 \equiv 0 \pmod M$

$$X = 12 \times 2^0 + 23 \times 2^8 + 34 \times 2^{16} + 45 \times 2^{24}$$

$$Y = 56 \times 2^0 + 67 \times 2^8 + 78 \times 2^{16} + 89 \times 2^{24}$$

$M = 11$

$$h_{11}(x) = \left(\underbrace{12}_{x_1} \times r_1 + \underbrace{23}_{x_2} \times r_2 + \underbrace{34}_{x_3} \times r_3 + \underbrace{45}_{x_4} \times r_4\right) \bmod 11$$

$(r_1 + r_2 + r_3 + r_4) \bmod 11$

$$h_{11}(y) = \left(\underbrace{56}_{y_1} \times r_1 + \underbrace{67}_{y_2} \times r_2 + \underbrace{78}_{y_3} \times r_3 + \underbrace{89}_{y_4} \times r_4\right) \pmod{11}$$

Claim: Under h_m, avg. search time $= O(1)$

Proof: fixed x. It suffices to show

length of $arr[h_m(x)] = O(1)$

Assume data in the hash table are $d_1, d_2, \dots, d_n$ (assume $x \neq d_i$)

$$\begin{cases} X_i = 1 \text{ if } d_i \text{ is in } arr[h_m(x)] \\ X_i = 0 \text{ otherwise} \end{cases}$$

length of $arr[h_m(x)] = X_1 + X_2 + \dots + X_n$

$$E[X_1 + X_2 + \cdots + X_n] = n \times \frac{1}{M} = \frac{n}{M} \leq 1 = O(1)$$

$$\underbrace{Pr[X_i = 1]}_{\| \; E[X_i]} = Pr[h_M(d_i) = h_M(x)] = \frac{1}{M}$$

$$E[X_i] = 1 \times Pr[X_i = 1] + 0 \times Pr[X_i = 0]$$

$$= Pr[X_i = 1]$$

Next Week:

1. What if n is unknown?

2. Perfect hashing

3. Review

What if n is unknown?

① initially capacity = 1 ⇒ M: ① M ≥ capacity
② M prime
③ M ≥ 256
⇓ randomly generate h_M

② When $\frac{\text{size}}{\text{capacity}} > 2$

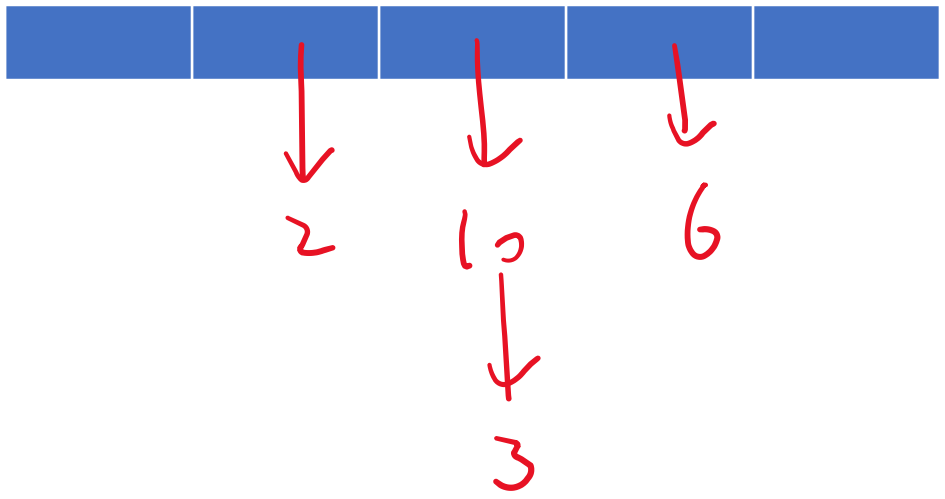
create a new hash table:

① capacity *= 2;

② insert the data in the old hash table to the new hash table

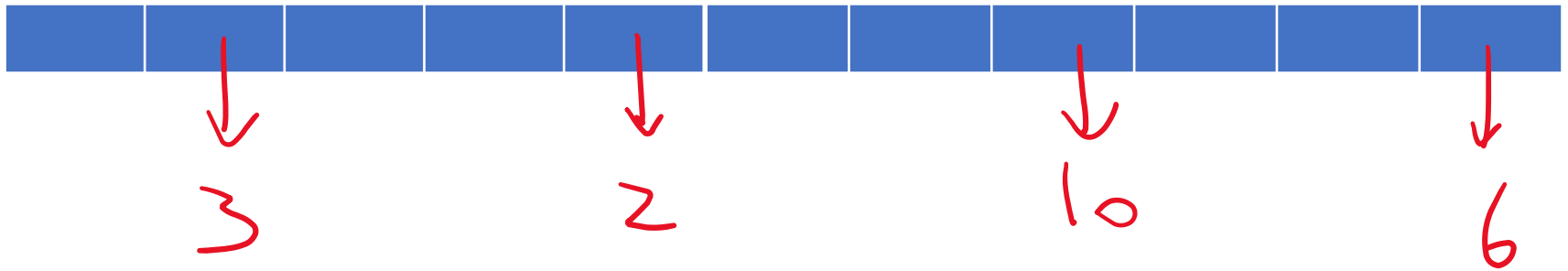
time
= O(capacity)
(rehash)

capacity = 4 size = 4 $(M = 5 \Rightarrow h_5)$



rehash

capacity = 8, size = 4 , $(M = 11 \Rightarrow h_{11})$



Perfect Hashing

input: $d_1, d_2, \cdots, d_n$

output: a hash table without collisions.

avg. time complexity $= O(n)$

Method 1: An $O(n^2)$-space hash table. $O(n)$ time

1° $M \geq n^2$, M prime, insert $d_1 \sim d_n$ according to h_M

2° if no collision, then we're done Prob. $> \frac{1}{2}$

otherwise, go to 1° and create

a new hash table

success = no collision $\quad \Pr[\text{success}] \geq \frac{1}{2}$

fail = otherwise

$X_{i,j} = 1 \Rightarrow h(d_i) = h(d_j)$

if $i \neq j \quad \Pr[h(d_i) = h(d_j)] \leq \frac{1}{M} \leq \frac{1}{n^2}$

$\Pr[\text{fail}] = \Pr[X_{1,2}=1 \text{ or } X_{1,3}=1 \text{ or } \cdots \text{ or } X_{1,n}=1$

$\text{or } X_{2,3}=1 \text{ or } X_{2,4}=1 \text{ or } \cdots \text{ or } X_{2,n}=1$

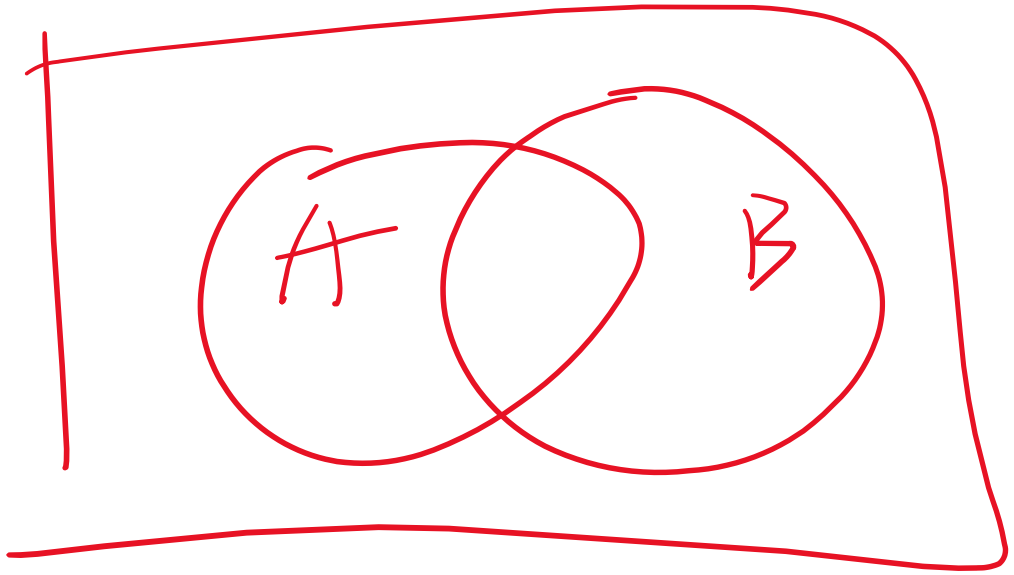
$\vdots$

$\text{or } X_{n-1,n}]$

$$\Pr[\text{fail}] = \Pr[X_{1,2}=1 \text{ or } X_{1,3}=1 \text{ or } \cdots \text{ or } X_{1,n}=1 \text{ or } X_{2,3}=1 \text{ or } X_{2,4}=1 \text{ or } \cdots \text{ or } X_{2,n}=1 \text{ or } \cdots \text{ or } X_{n-1,n}]$$

$$\leq \Pr[X_{1,2}=1] + \Pr[X_{1,3}=1] + \cdots + \Pr[X_{1,n}=1] + \Pr[X_{2,3}=1] + \cdots + \Pr[X_{2,n}=1] + \cdots + \Pr[X_{n-1,n}=1]$$

$$\leq \frac{n(n-1)}{2} \times \frac{1}{n^2} < \frac{1}{2}$$



$$Pr[A \cup B] \leq Pr[A] + Pr[B]$$

Method 2:

(last week)

A° $M \geq n$, M prime. insert $d_1 \sim d_n$ according to h_M. ← to arr

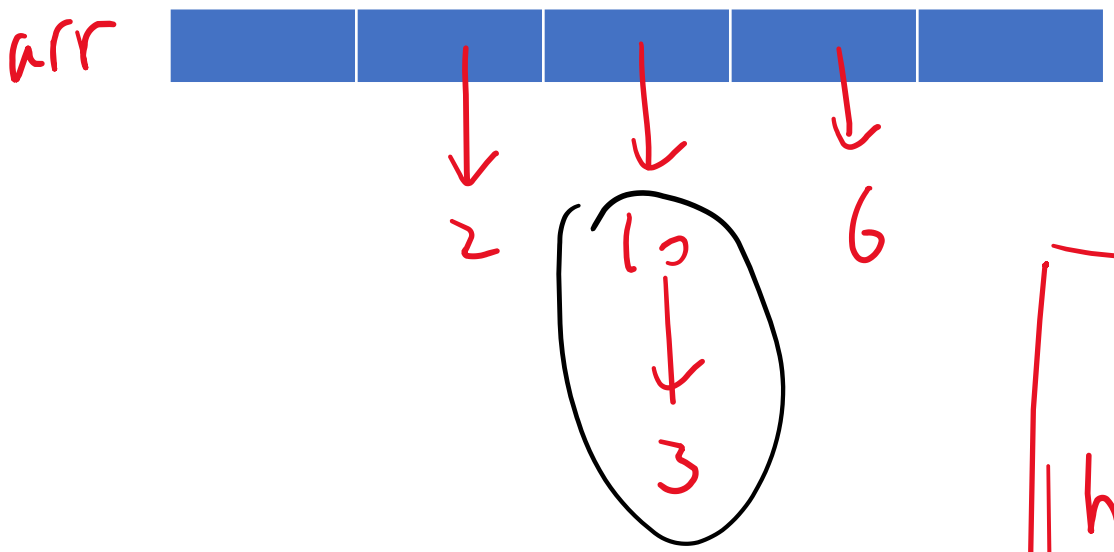
$O(n)$ time

B° For $i = 0$ to $M-1$, store data in arr[i] using Method 1.

$O(n)$ time (avg)

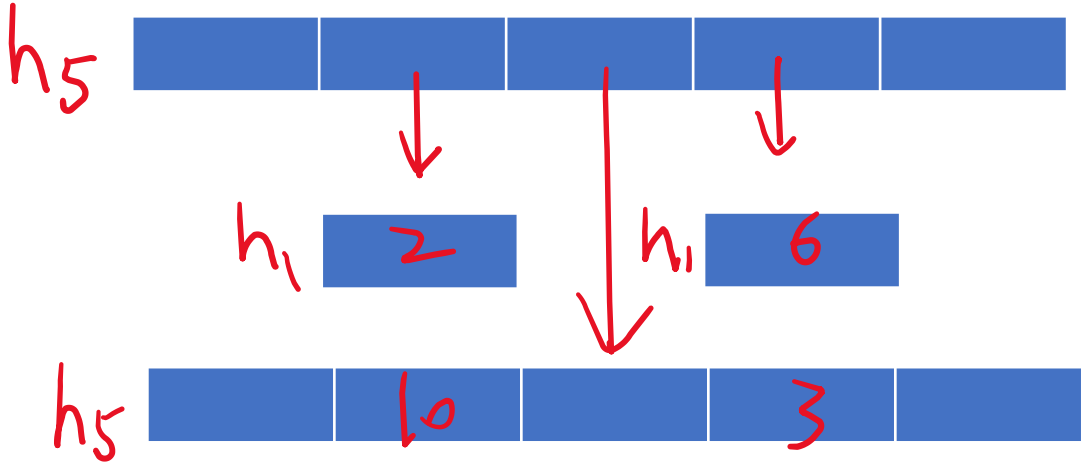
Let n_i be the # data in arr[i].

$$\text{space} = O(n) + \sum_{i=1}^{M-1} O(n_i^2)$$



Method 2

$n_0 = 0,\ n_1 = 1,\ n_2 = 2$



$n_3 = 1,$

$n_4 = 0$

$$\text{space} = 5 + 1^2 + 1^2 + O(2^2)$$

$\sum_{k=1}^{3} n_k^2 = 1^2 + 1^2 + 2^2 = 6$

$d_1 = 2,\ d_2 = 10$

$d_3 = 6,\ d_4 = 3$

$X_{i,j} = 1$ if $h(d_i) = h(d_j)$ or $i = j$

$X_{i,j} = 0$ otherwise

$X_{1,1} = 1$ $X_{1,2}$ $X_{1,3}$ $X_{1,4}$

$X_{2,1}$ $X_{2,2} = 1$ $X_{2,3}$ $X_{2,4} = 1$

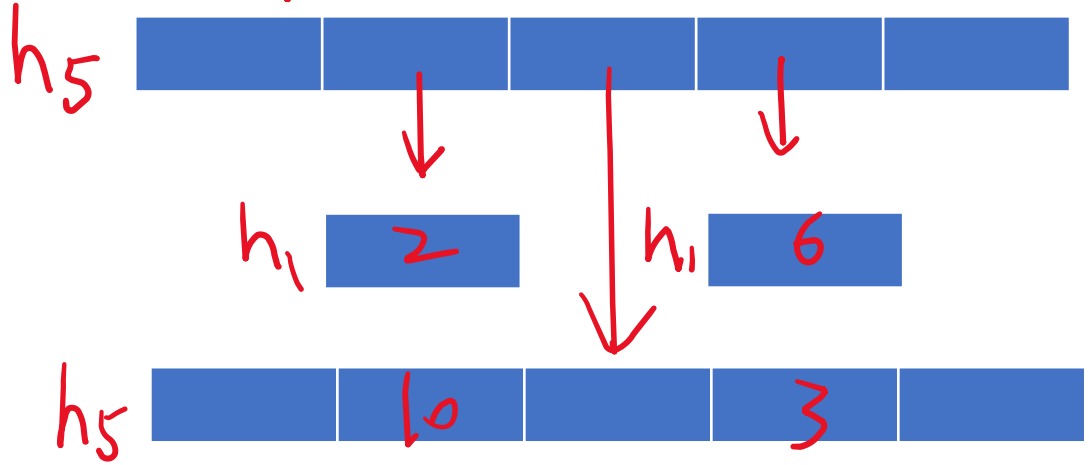
$X_{3,1}$ $X_{3,2}$ $X_{3,3} = 1$ $X_{3,4}$

$X_{4,1}$ $X_{4,2}$ $X_{4,3}$ $X_{4,4} = 1$

$= 1$

Step 2

$n_0 = 0,\ n_1 = 1,\ n_2 = 2$



$n_3 = 1,$

$n_4 = 0$

space $= 5 + 1^2 + 1^2 + O(2^2)$

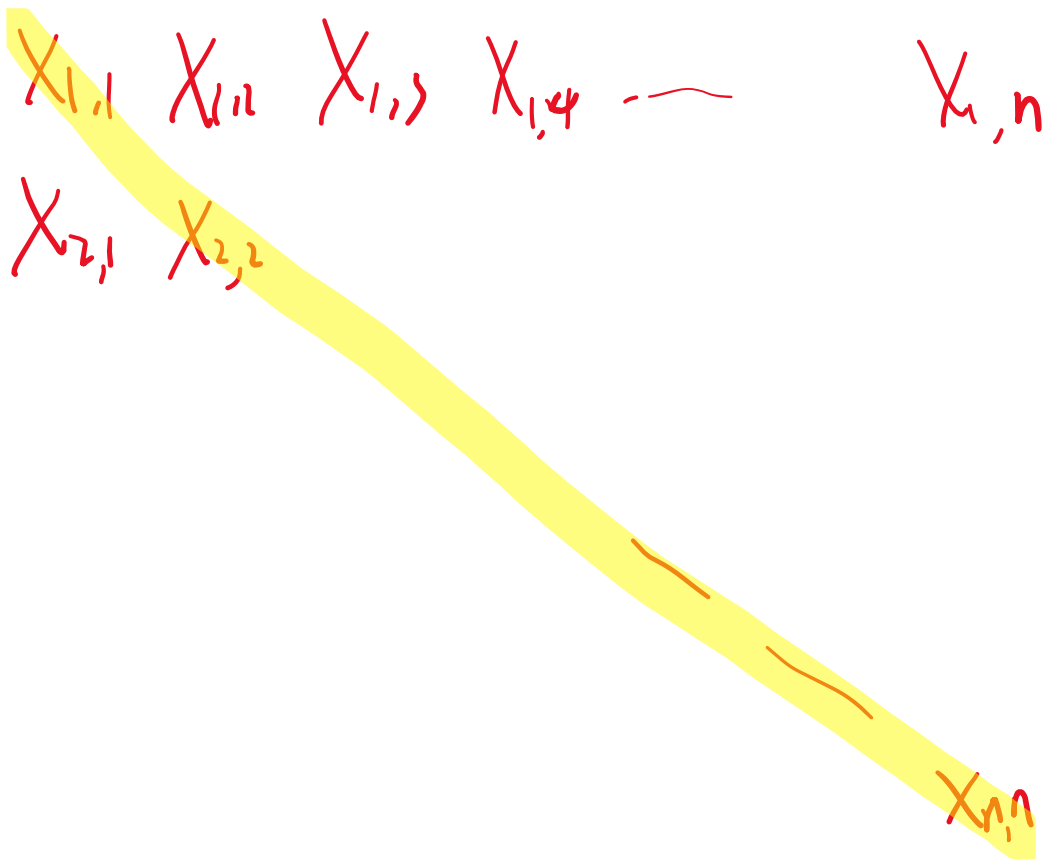
$$E[\text{space}] = E\left[O(n) + \sum_{i=0}^{M-1} O(n_i^2)\right]$$

$$= O\left(E\left[n + \sum_{i=0}^{M-1} n_i^2\right]\right) = O(n)$$

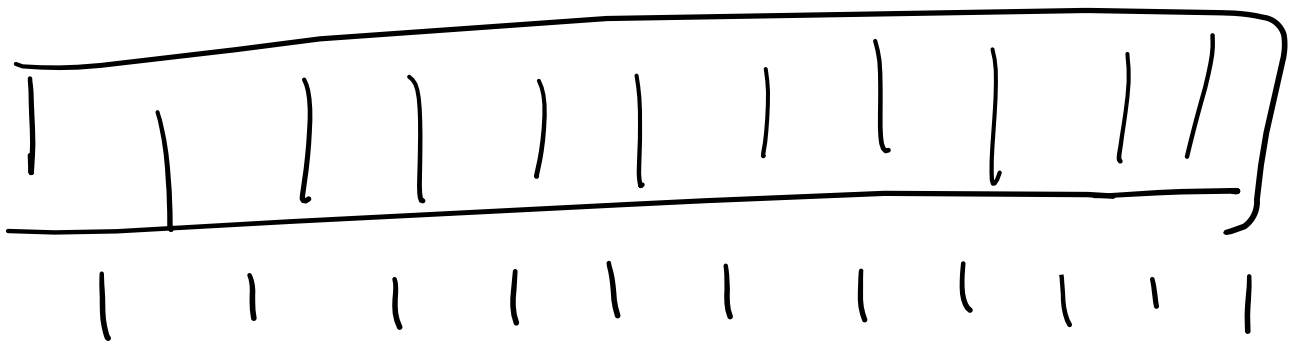
Claim: $E\left[\sum_{i=0}^{M-1} n_i^2\right] = O(n)$

Proof $E\left[\sum_{i=0}^{M-1} n_i^2\right] = E\left[\sum_{i=1}^{n}\sum_{j=1}^{n} X_{i,j}\right]$ $\qquad O(n)$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} Pr[X_{i,j}=1] = n \times 1 + (n^2-n) \times \frac{1}{M} \leq n + \frac{n^2-n}{n}$$

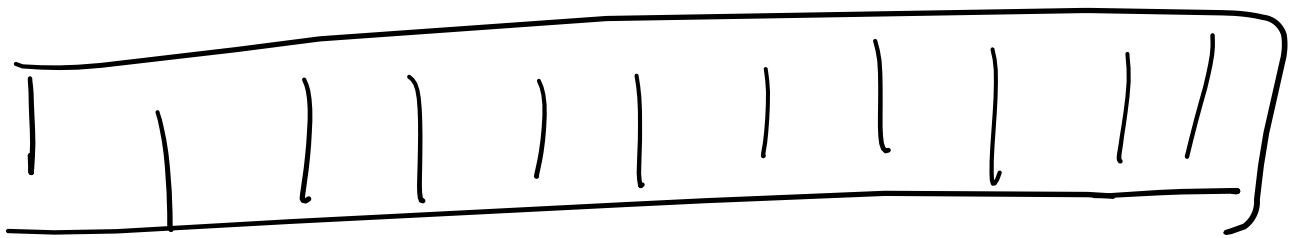

$X_{1,1}$ $X_{1,2}$ $X_{1,3}$ $X_{1,4}$ — $X_{1,n}$
$X_{2,1}$ $X_{2,2}$
$X_{n,n}$

n box

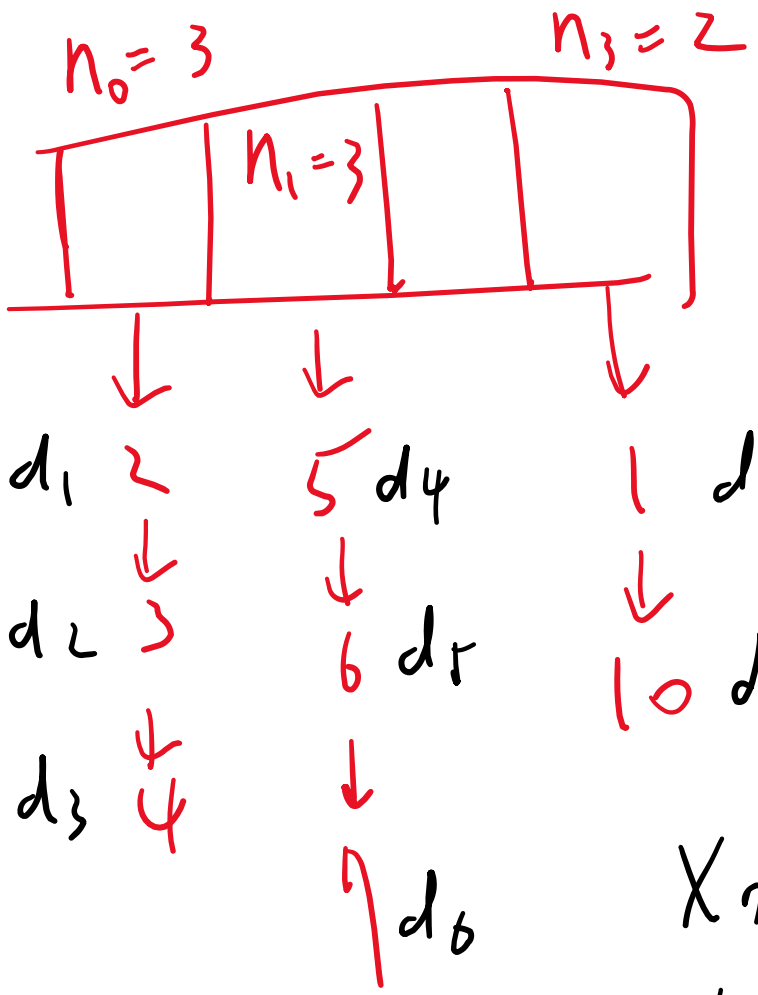


$n_i = 1 \qquad \sum_i n_i^2 = O(n)$

n box



$$n_i = 1 \qquad \sum_i n_i^2 = n^2$$



$3^2 + 3^2 + 2^2 = 22$

$8 + 6 + 6 + 2$

$= 22$

$X_{1,2}$ $X_{1,3}$

$X_{2,1}$, $X_{2,3}$

$X_{3,1}$, $X_{3,2}$

$X_{7,8}$

$X_{5,7}$

2

```
int f() {
    int arr[1500000]; // stack
    f()
}
```

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

    arr[i] --

}
for(i=0 ; i<n ; i++){

    *(arr+i)

}
```

100 101 102 103

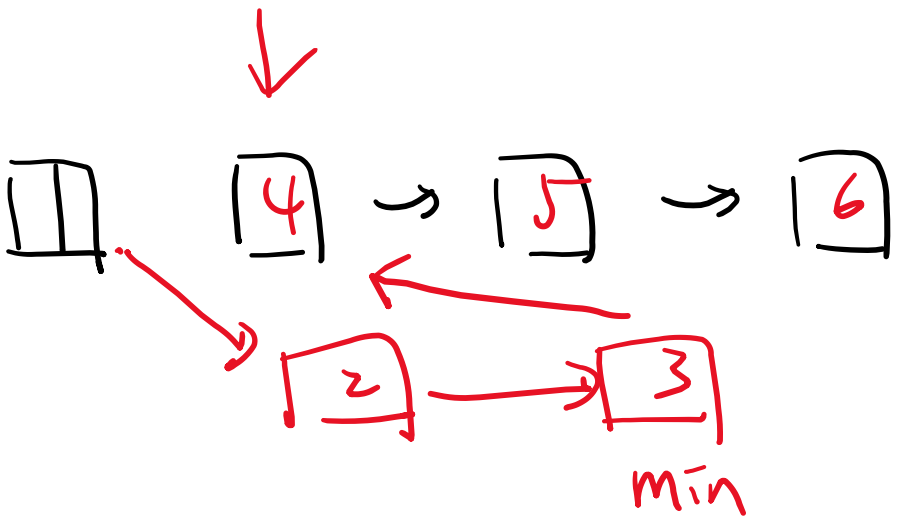
arr

*(arr++)

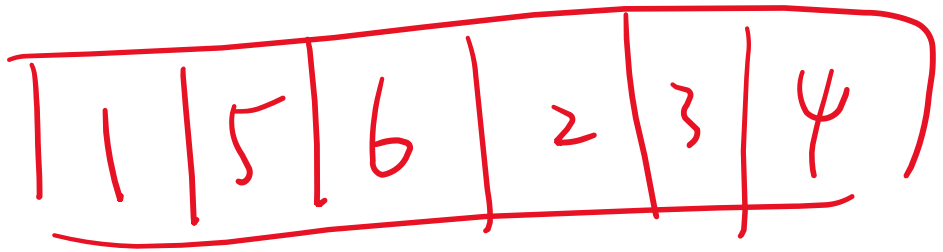
arr[3]

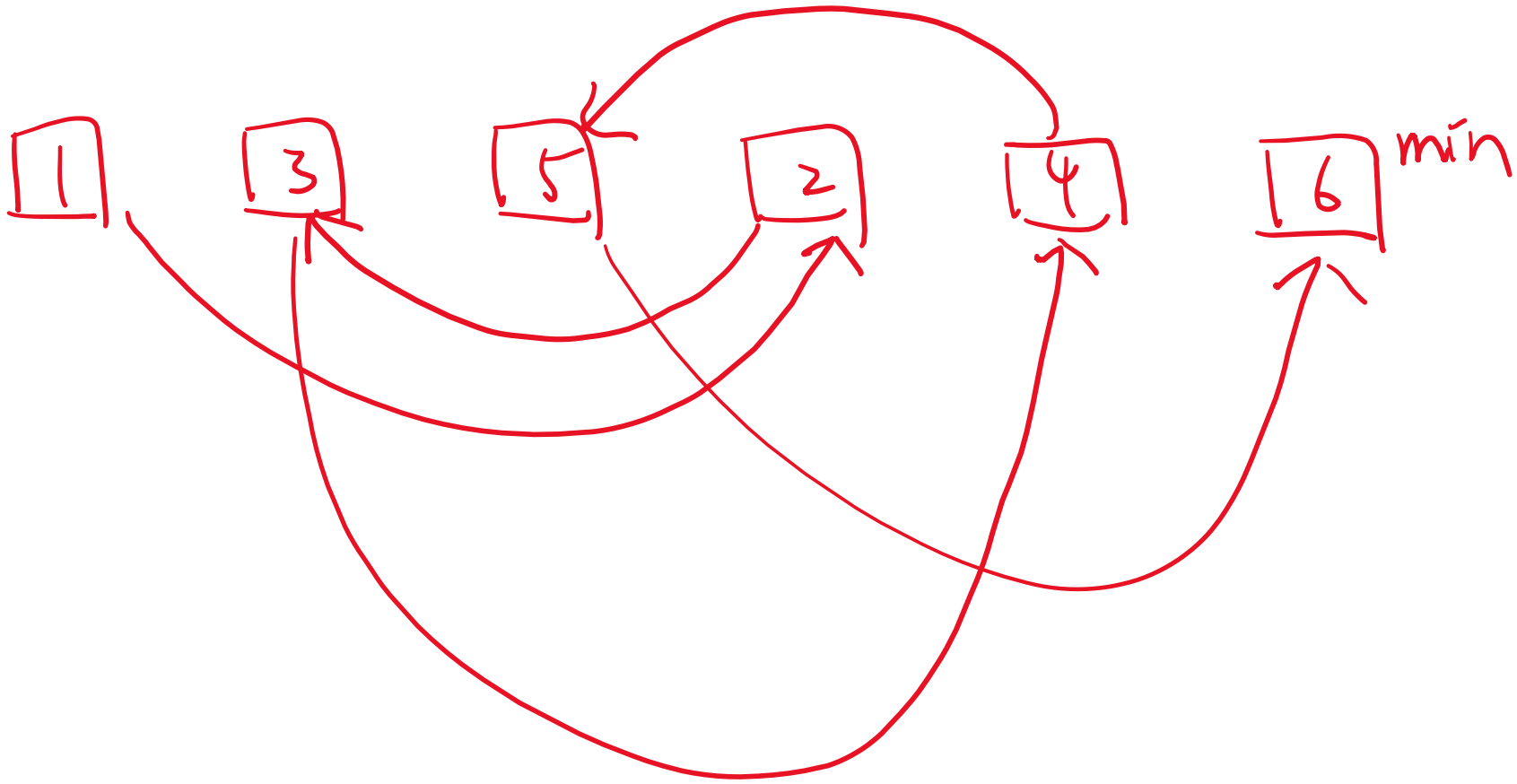
1° addr = 100 + 3

1² get the data in addr.

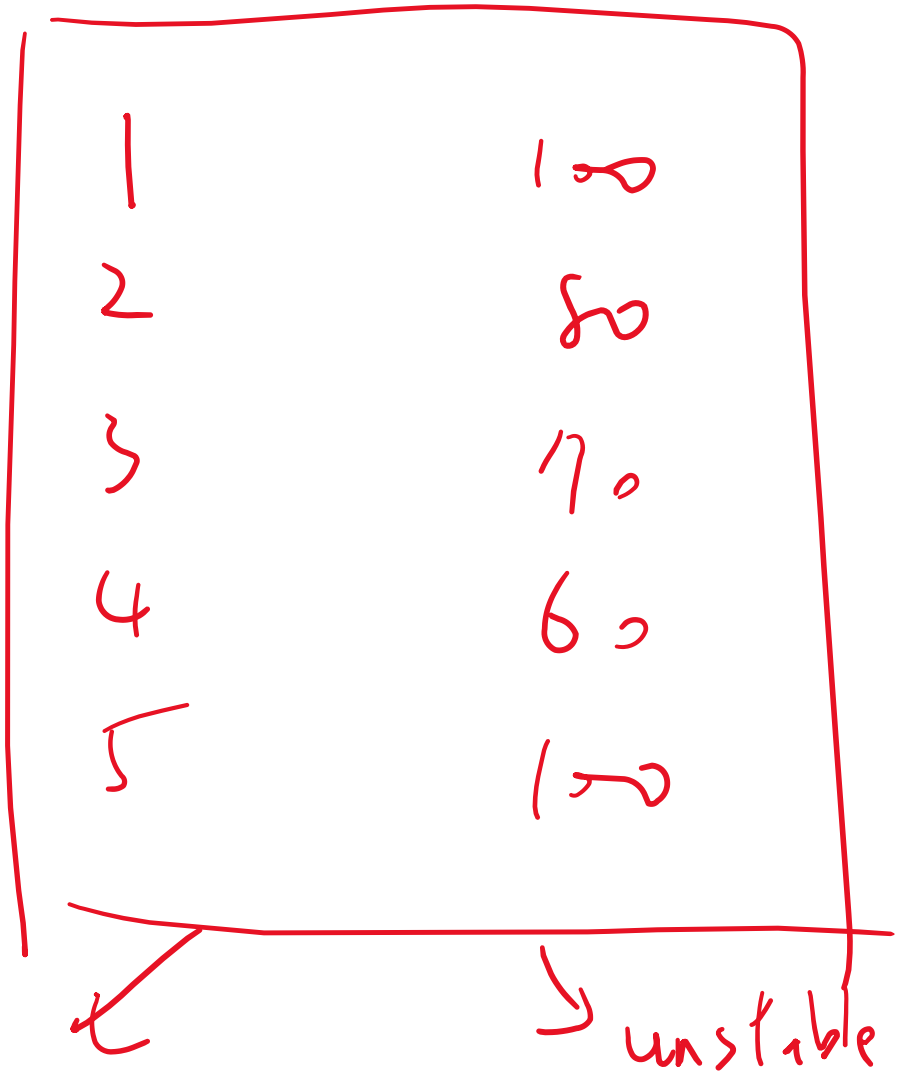


arr[i]




1
3
5
2
4
6
min

TA	Score
5	100
2	80
3	70
1	100
4	60



stable

1	100
5	100
	80
	70

unstable

5	100
1	100
	80
	60

$n \log n + n = O(\log n)$

$O(n(\log n + 1)) = O(n \log n)$

quick sort time complexity

$O(n^3)$ ok

$\Theta(n^2)$ X

merge sort

$\Theta(n \log n)$

insertion sort

$O(n^2)$ $\Theta(n^2)$

(U) the worst-case time complexity of insertion sort is $O(n^2)$

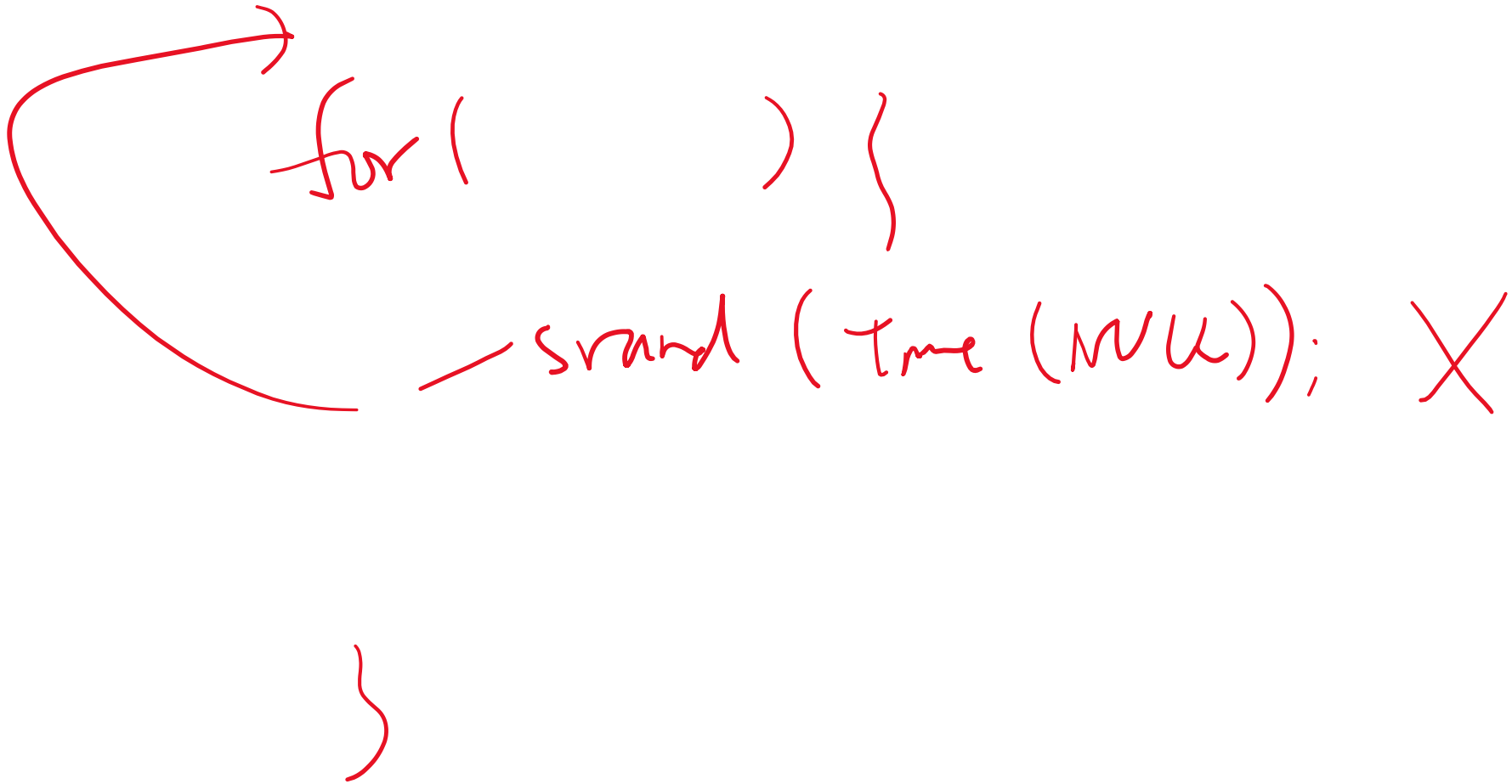
(U) the "best-case" time complexity of insertion sort is $O(n^2)$

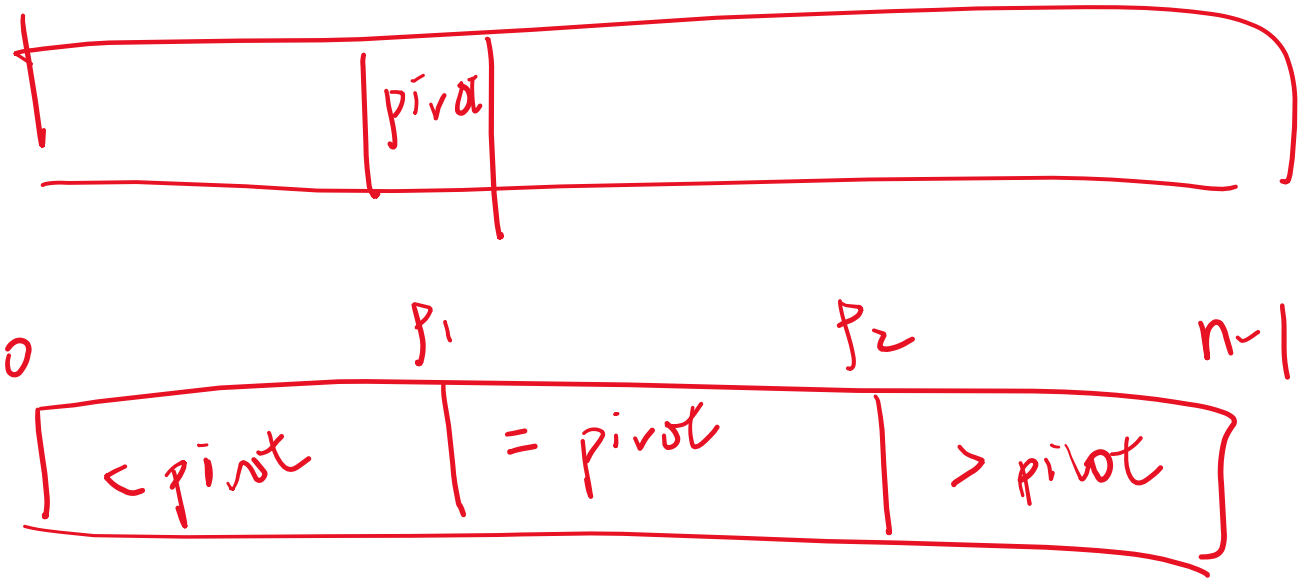
(U) the "best case" time complexity of insertion sort is $O(n)$

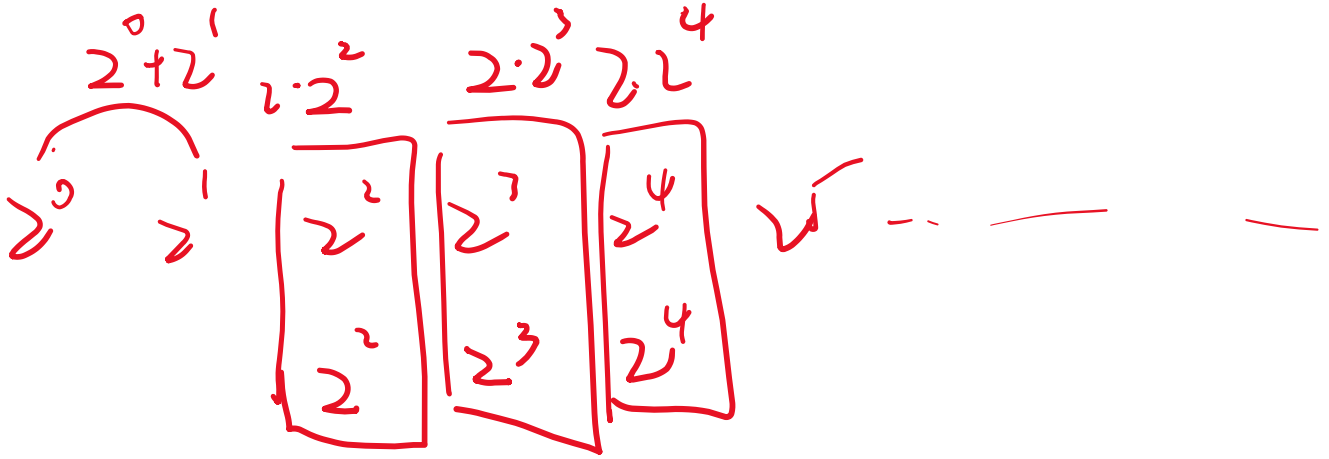
(✓) the worst-case time complexity of insertion sort is $\Theta(n^2)$

(✗) the "best-case" time complexity of insertion sort is $\Theta(n^2)$

(✓) the "best case" time complexity of insertion sort is $\Theta(n)$

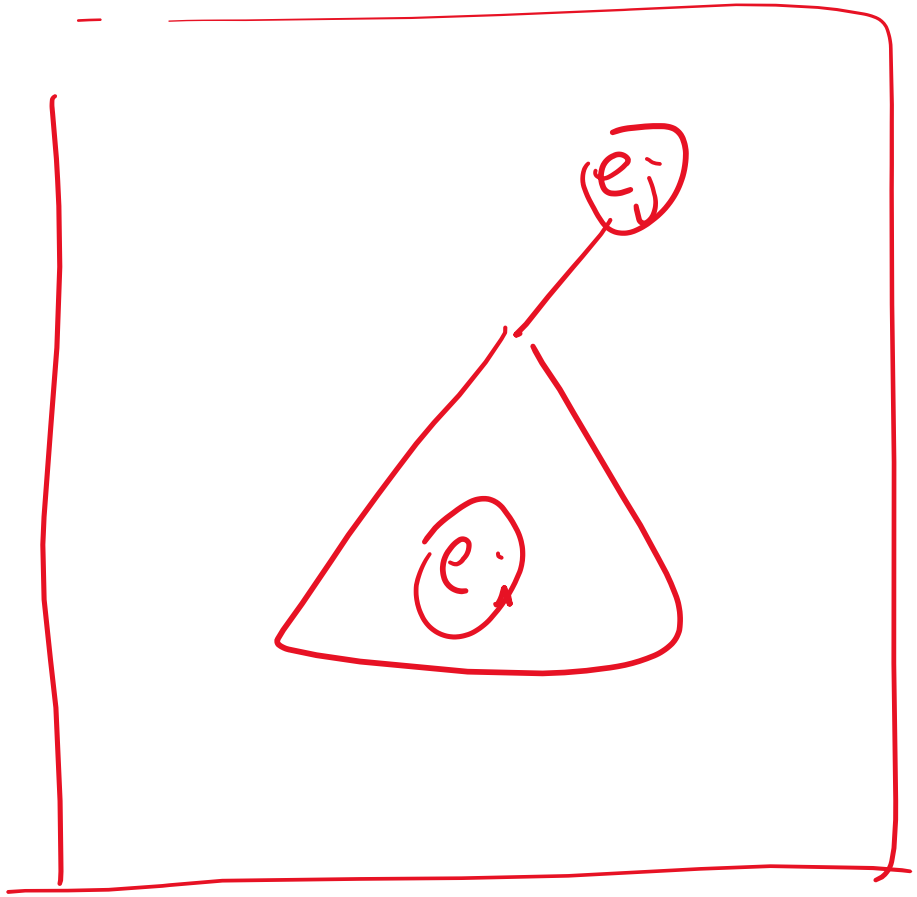

for( ){
srand(time(NULL));
}

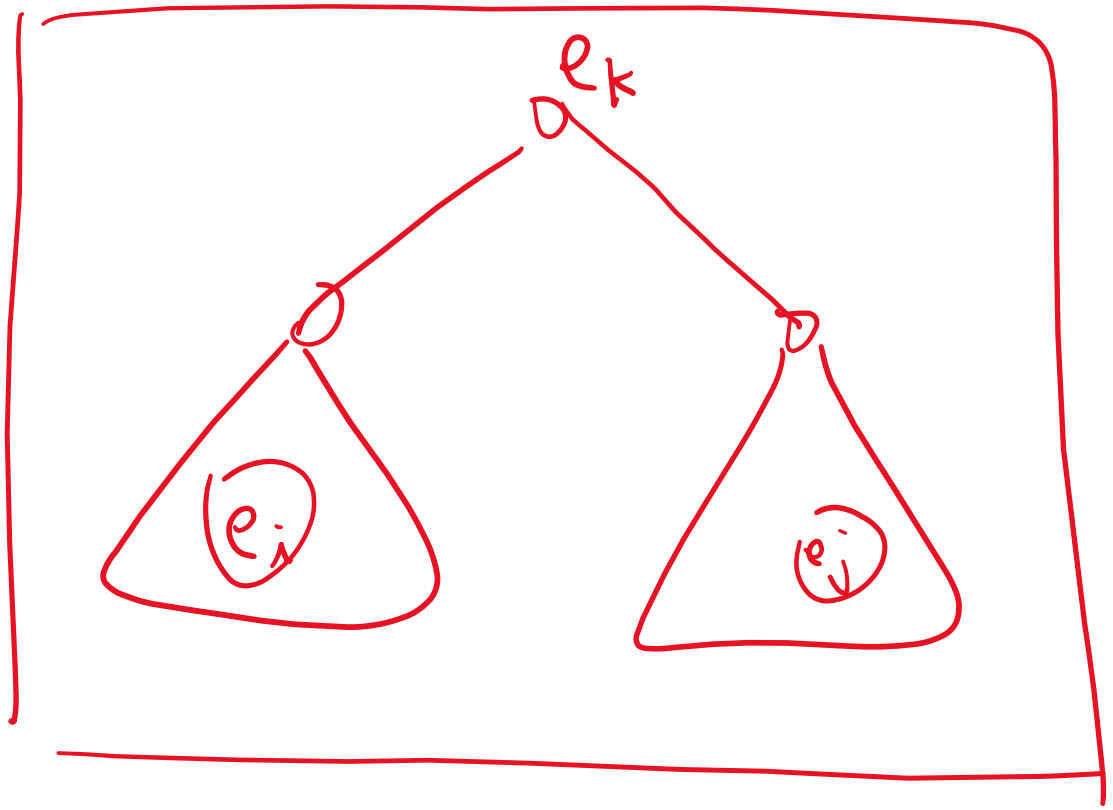

pivot
0
p1
p2
n-1
< pivot
= pivot
> pivot



$$2\left(2^1+2^2+2^3+\cdots+2^k\right)$$

$$= O(2^k)$$





$$e_i \leq e_k \leq e_j$$

$$M = 10$$

$$d_1 = 10$$

$$d_2 = 20$$

$$d_3 = 30$$

$$d_4 = 40$$

$$d_i^2 \bmod M = 0$$