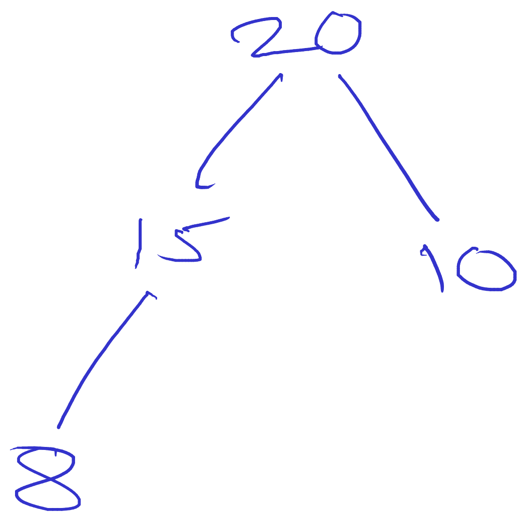
arr

10	8	20	15

size=0,1,2,3,4

arr

20	15	10	8



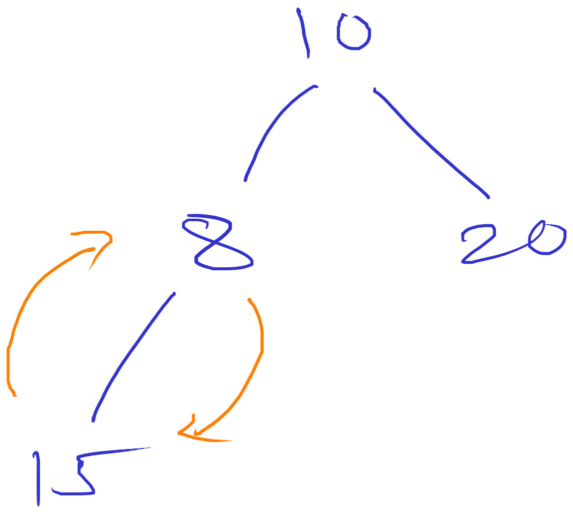
size=4,3,2,1,0

arr

8	10	15	20

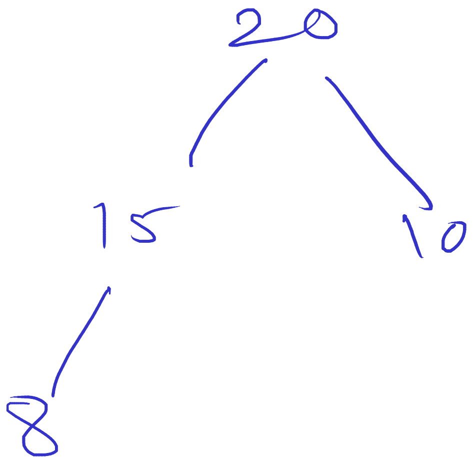
arr

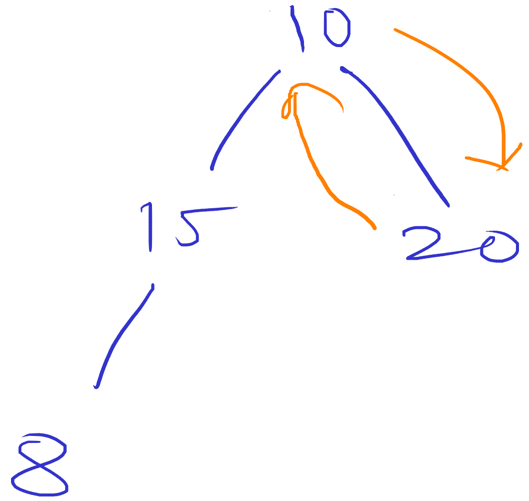
10	8	20	15

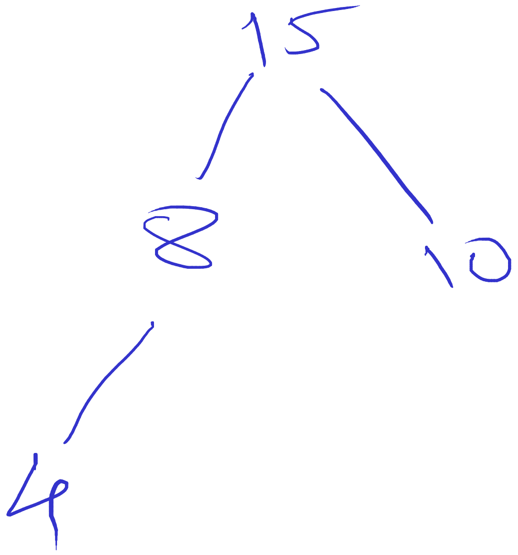


Heapify

$T(n) = O(n)$

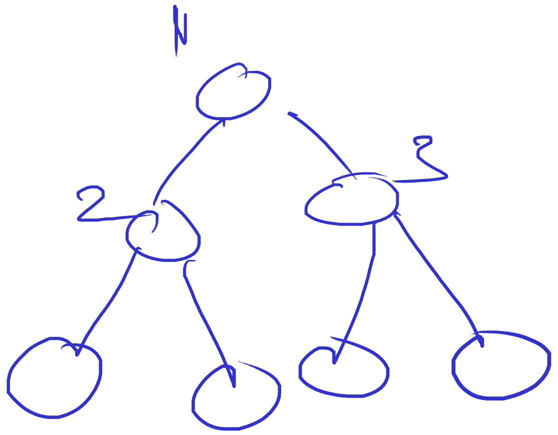






$n = 7$

$n/2 = 3$



Merge Sort

//1. divide array into two parts
//2. sort both partitions individually
//3. merge both sorted partitions into temp array in sorted order
//4. over write temp array into original array

Time Complexity

Array size = n

Levels of division = log n

Per level comparisions = n

Total comparisions = n log n

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

Auxillary Space Complexity

- temp array will be needed to merge sorted partitions of array.

size of (temp array) = n

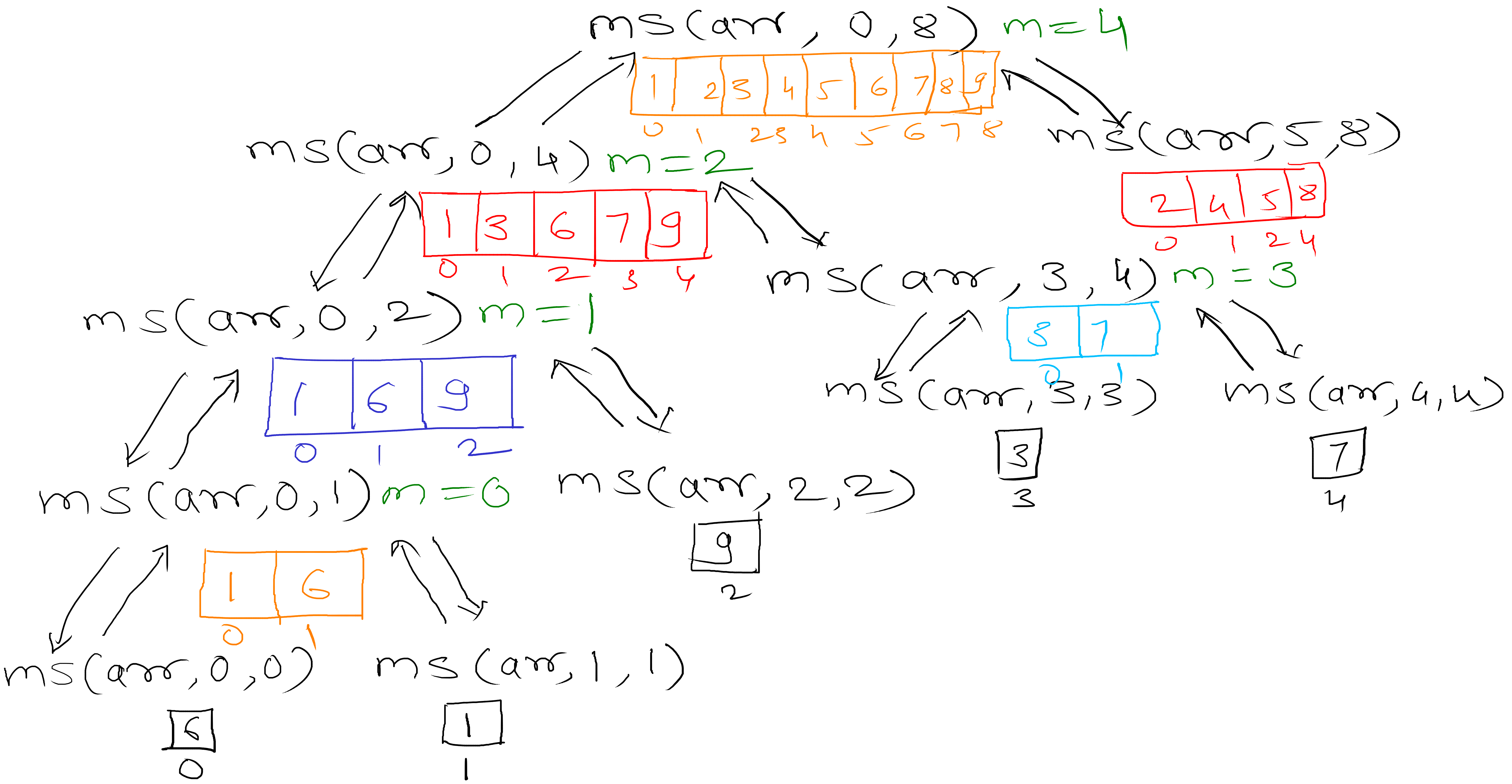
$S(n) = O(n)$

1 3 6 7 9

~~1~~ ~~6~~ ~~9~~ ~~3~~ ~~7~~

~~1~~ ~~6~~

~~6~~	~~1~~	~~9~~	~~3~~	~~7~~	2	8	4	5
0	1	2	3	4	5	6	7	8



Quick Sort

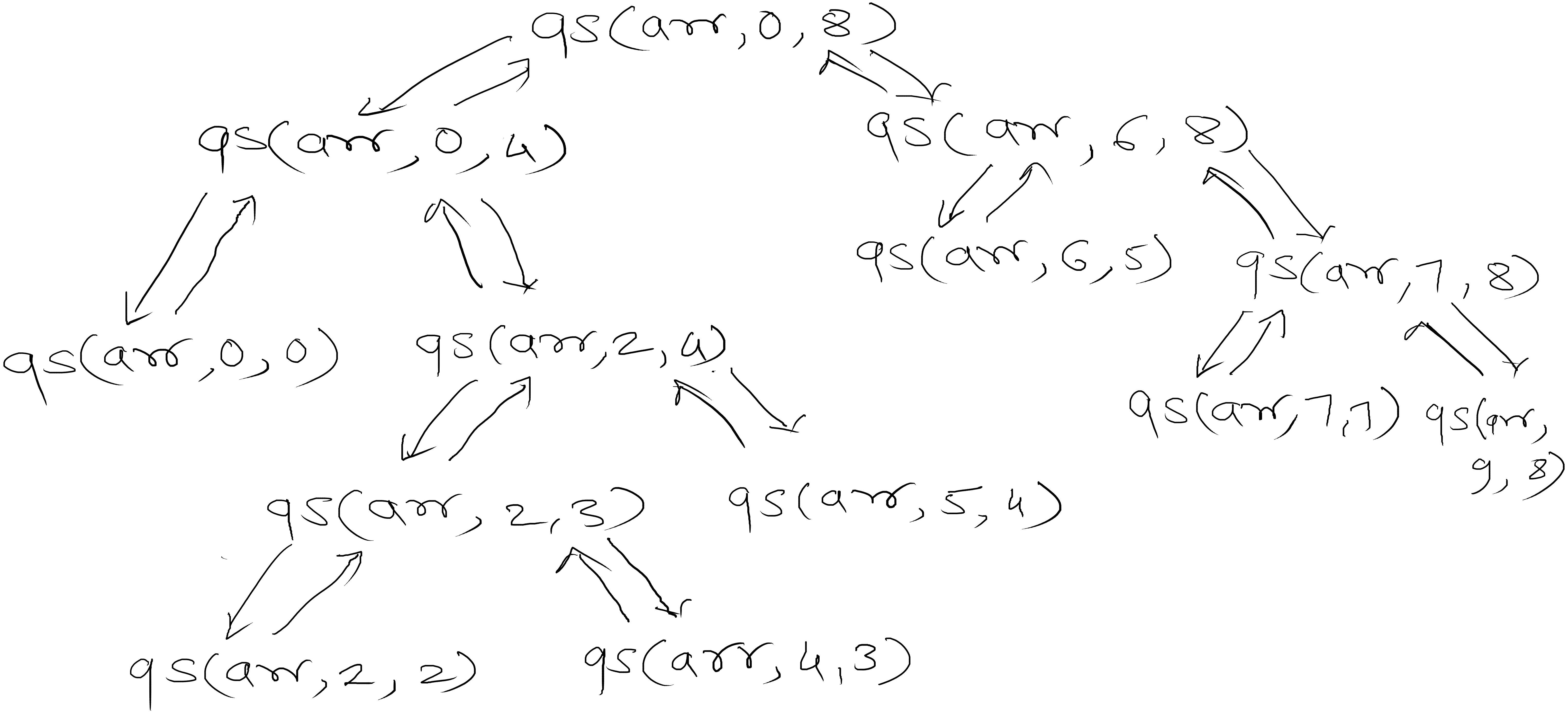
//1. select one referance/axis/pivot element from array

// leftmost/rightmost/middle

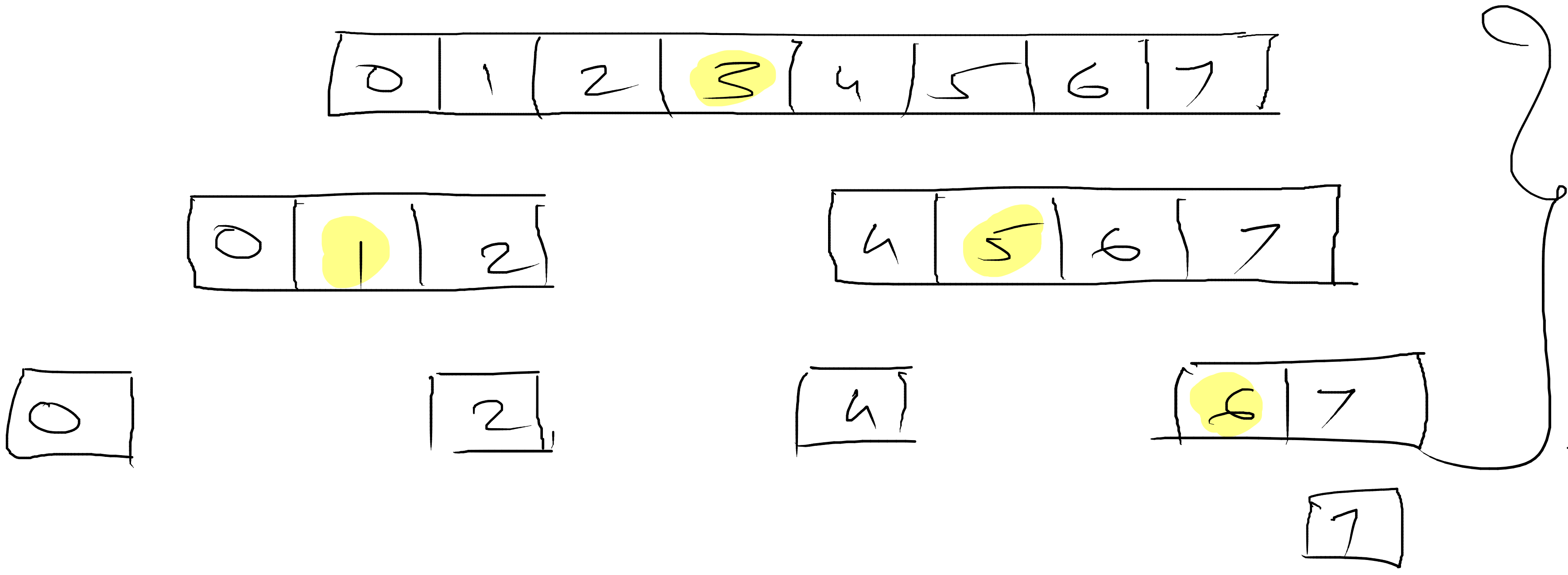
//2. arrange all smaller elements than pivot on left side of pivot

//3. arrange all greater elements than pivot on right side of pivot

//4. sort left and right partition of pivot individually



Quick Sort

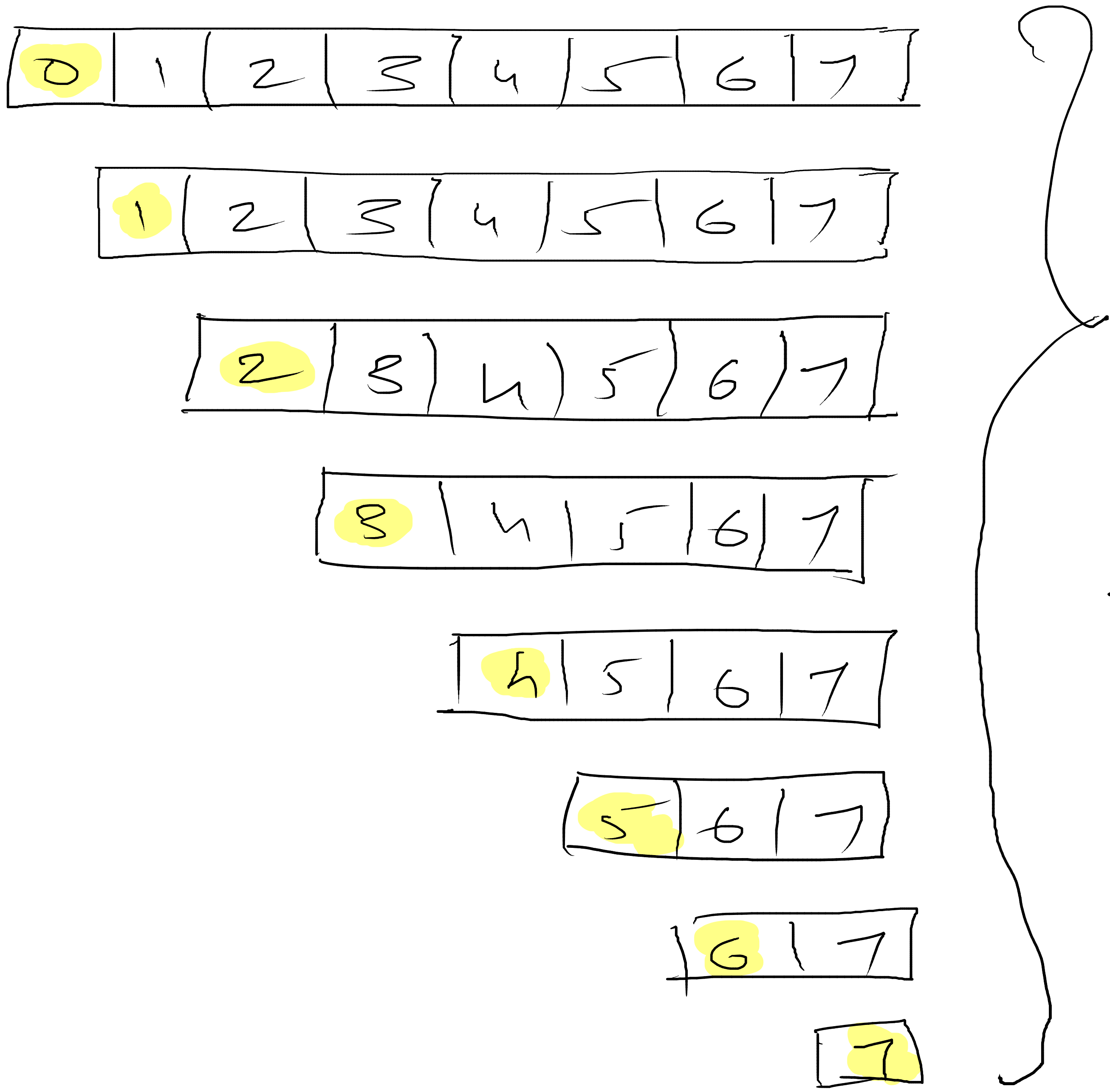


levels = log n

comps ∝ n per level

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

Time complexity of quick sort also depends on selection of pivot.



levels = n

comp per leve ∝ n

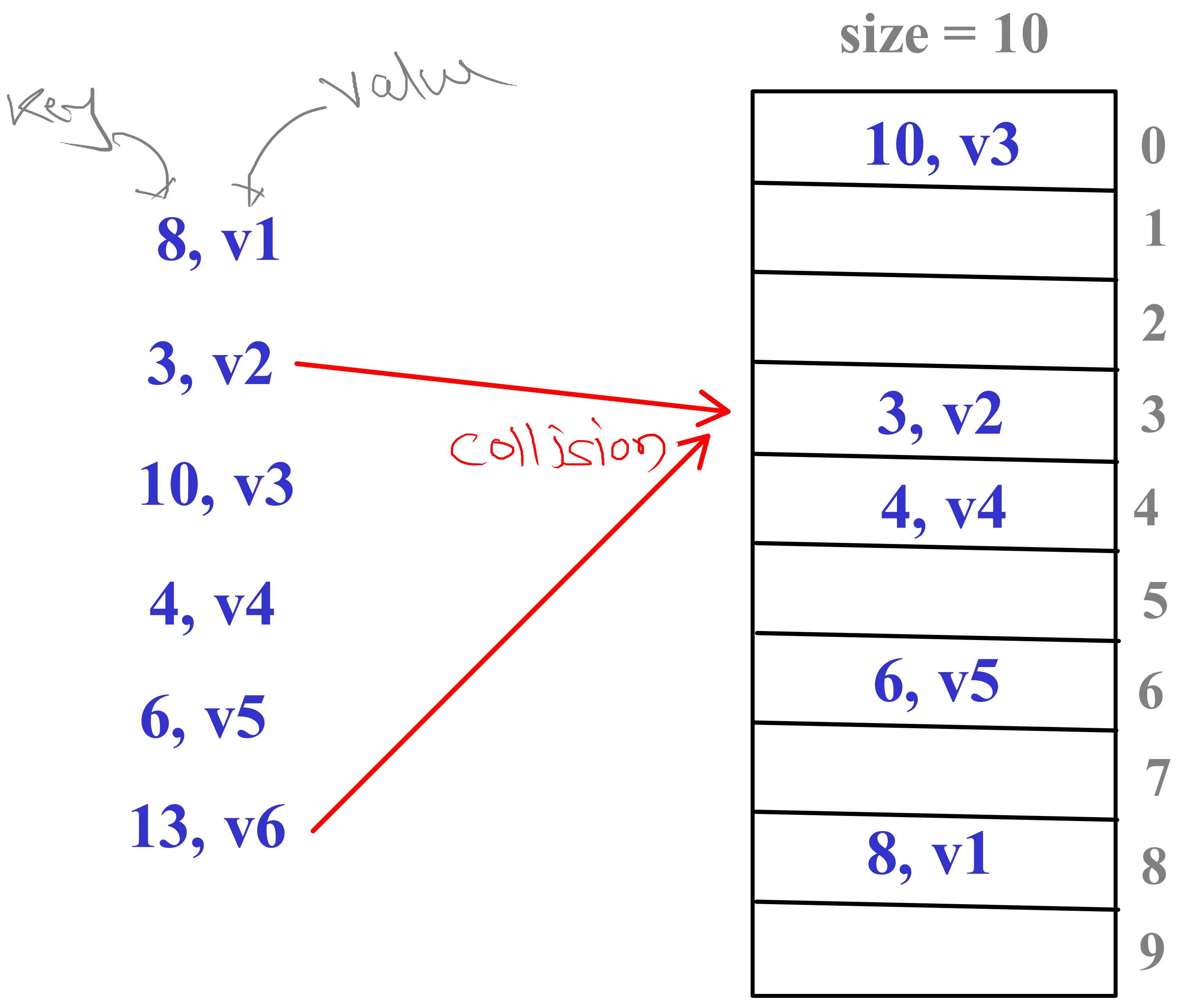
$T(n) = O(n^2)$

Pivot selection —

to improve performance

- dual pivot quick sort
- median of 3

Hashing



Hash Table

h(k) = k % size

$h(8) = 8 \% 10 = 8$

$h(3) = 3 \% 10 = 3$

$h(10) = 10 \% 10 = 0$

$h(4) = 4 \% 10 = 4$

$h(6) = 6 \% 10 = 6$

$h(13) = 13 \% 10 = 3$

Add/insert: O(1)
- slot = k % size
- arr[slot] = value

find/search: O(1)
- slot = k % size
- return arr[slot].value;

Delete/remove; O(1)
- slot = k % size
- arr[slot] = 0/null;

Collision :
- multiple keys yeild same slot.

Closed Addressing/ Seperate Chaining / Chaining

8, v1

3, v2

10, v3

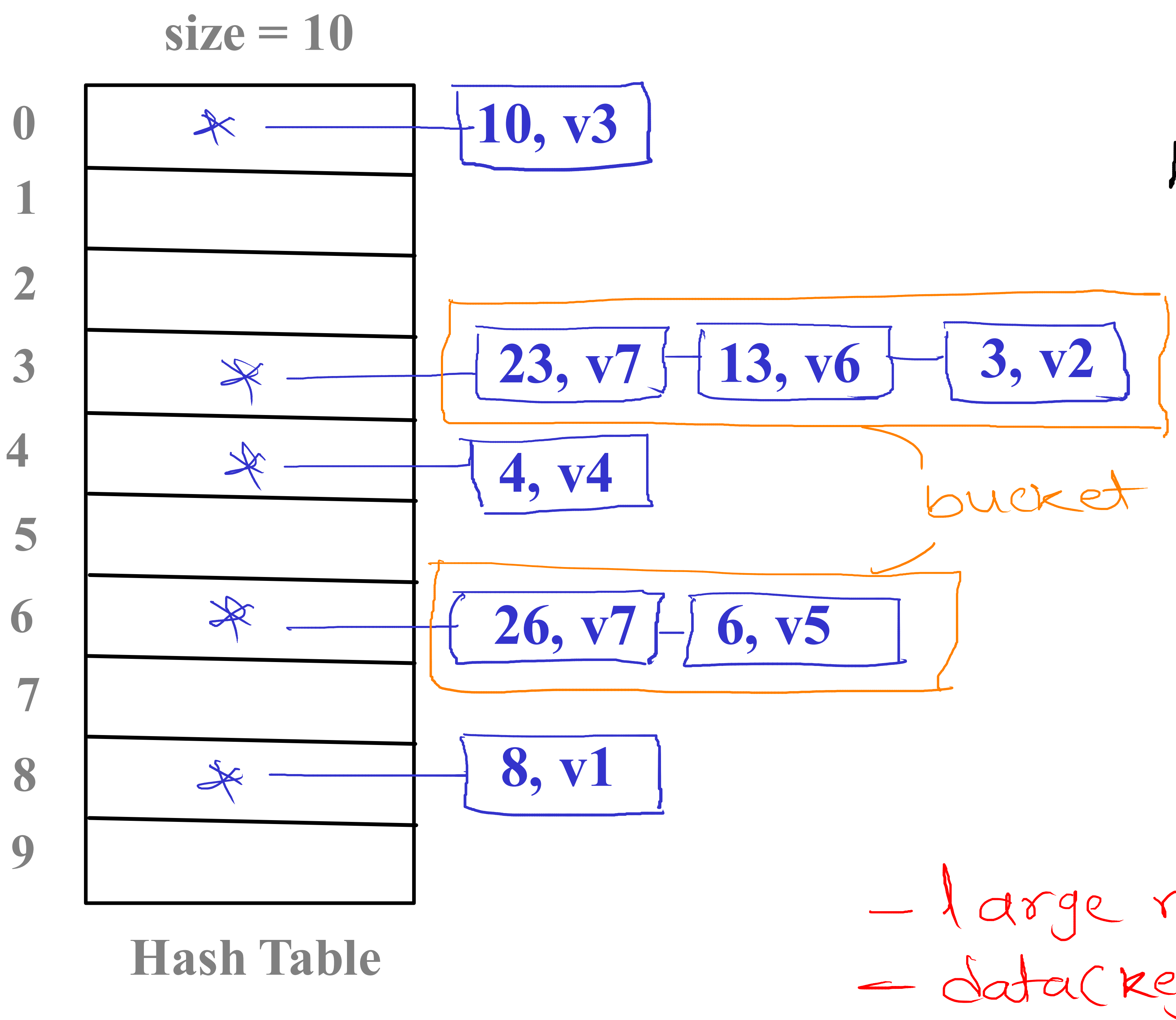
4, v4

6, v5

13, v6

23, v7

26, v7


size = 10

Hash Table

h(k) = k % size

$h(8) = 8 \% 10 = 8$

$h(3) = 3 \% 10 = 3$

$h(10) = 10 \% 10 = 0$

$h(4) = 4 \% 10 = 4$

$h(6) = 6 \% 10 = 6$

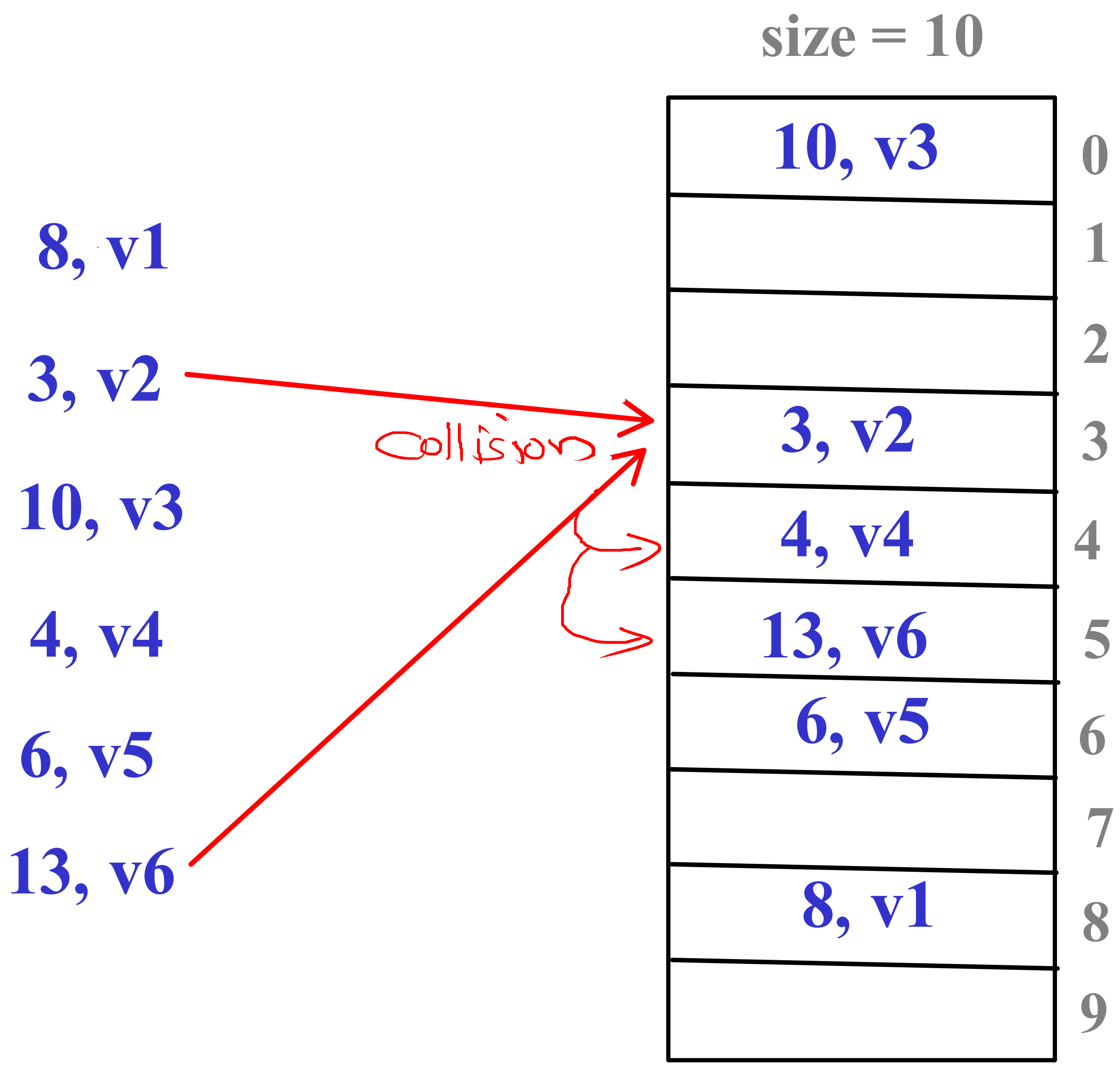
$h(13) = 13 \% 10 = 3$ ⓒ

$h(23) = 23 \% 10 = 3$ ⓒ

$h(26) = 26 \% 10 = 6$ ⓒ

- large memory requirement
- data(key, value) is stored outside the table
- worst case - O(n)
  ↳ if maximum keys yield same slot

Open Addressing - Linear Probing



0	10, v3
1	
2	
3	3, v2
4	4, v4
5	13, v6
6	6, v5
7	
8	8, v1
9	

Hash Table

$\underline{h(k)}$ = key % size

$h(k, i) = [\ \underline{h(k)} + \underline{f(i)}\ ]$ % size

$f(i) = \underline{i}$

where i = 1, 2, 3, ...

↳ probe number

h(8) = 8 % 10 = 8

h(3) = 3 % 10 = 3

h(10) = 10 % 10 = 0

h(4) = 4 % 10 = 4

h(6) = 6 % 10 = 6

h(13) = 13 % 10 = 3 (collision)

h(13,1) = [3 + 1] % 10

= 4 (1st) (collision)

h(13,2) = [3 + 2] % 10

= 5 (2nd probe)

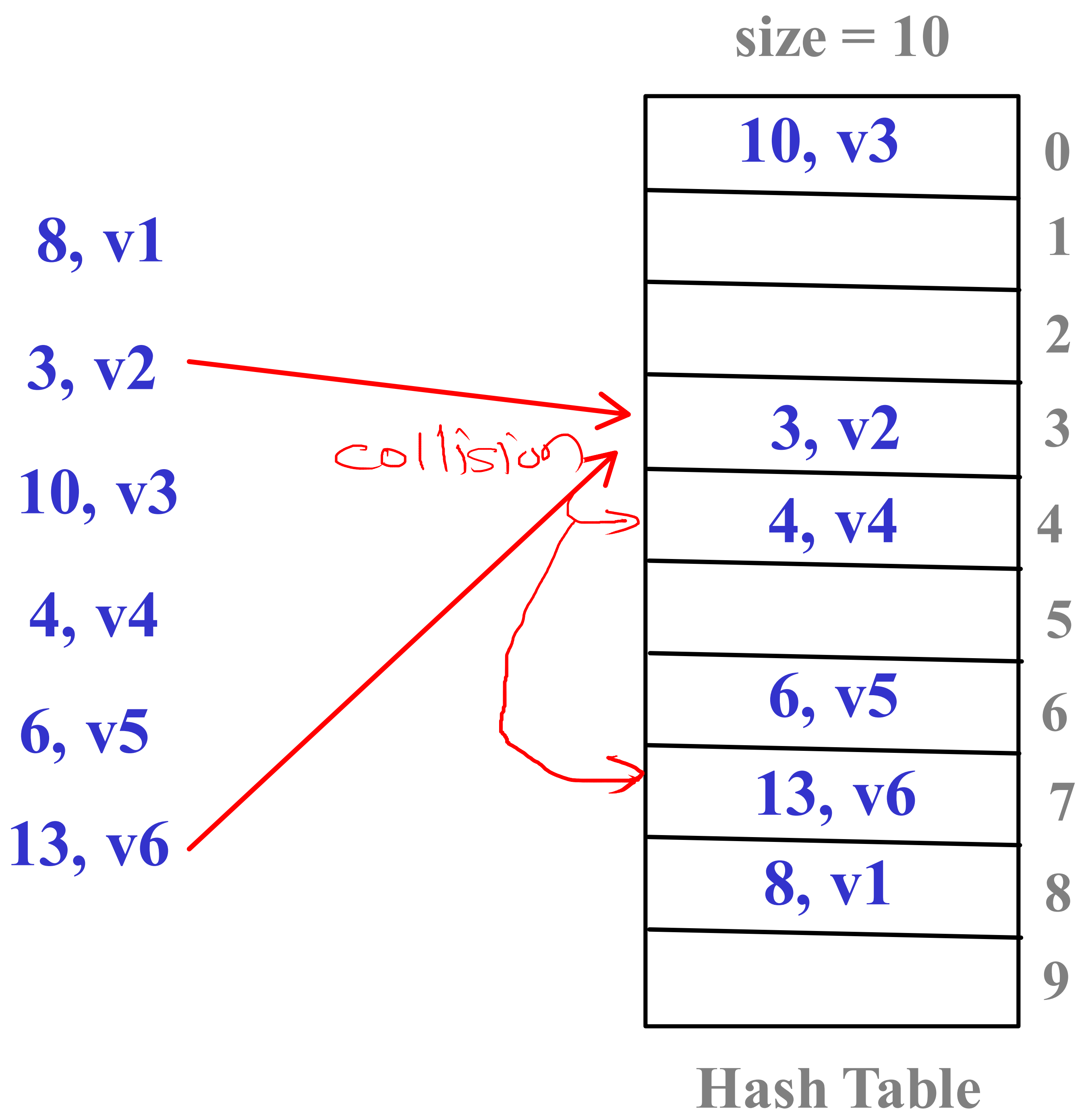
Probing -

finding new slot for key, if collision occured

Primary clustering -

- need long runs of filled slots "near" key position to find free slot

Open Addressing - Quadratic Probing



h(k) = key % size

h(k, i) = [h(k) + f(i)] % size

f(i) = i^2

where i = 1, 2, 3, ...

h(8) = 8 % 10 = 8

h(3) = 3 % 10 = 3

h(10) = 10 % 10 = 0

h(4) = 4 % 10 = 4

h(6) = 6 % 10 = 6

h(13) = 13 % 10 = 3 (collision)

h(13, 1) = [3 + 1] % 10

= 4 (1^{st}) (collision)

h(13, 2) = [3 + 4] % 10

= 7 (2^{nd})

Open Addressing - Quadratic Probing

23, v7

33, v8

size = 10

10, v3	0
	1
23, v7	2
3, v2	3
4, v4	4
	5
6, v5	6
13, v6	7
8, v1	8
33, v8	9

Hash Table

h(k) = key % size

h(k, i) = [h(k) + f(i)] % size

f(i) = i^2

where i = 1, 2, 3, ...

h(23) = 23 % 10 = 3 (collision)

h(23,1) = [3+1] % 10 = 4 (1st) (collision)

h(23,2) = [3+4] % 10 = 7 (2nd) (collision)

h(23,3) = [3+9] % 10 = 2 (3rd)

h(33) = 33 % 10 = 3 (collision)

h(33,1) = [3+1] % 10 = 4

h(33,2) = [3+4] % 10 = 7

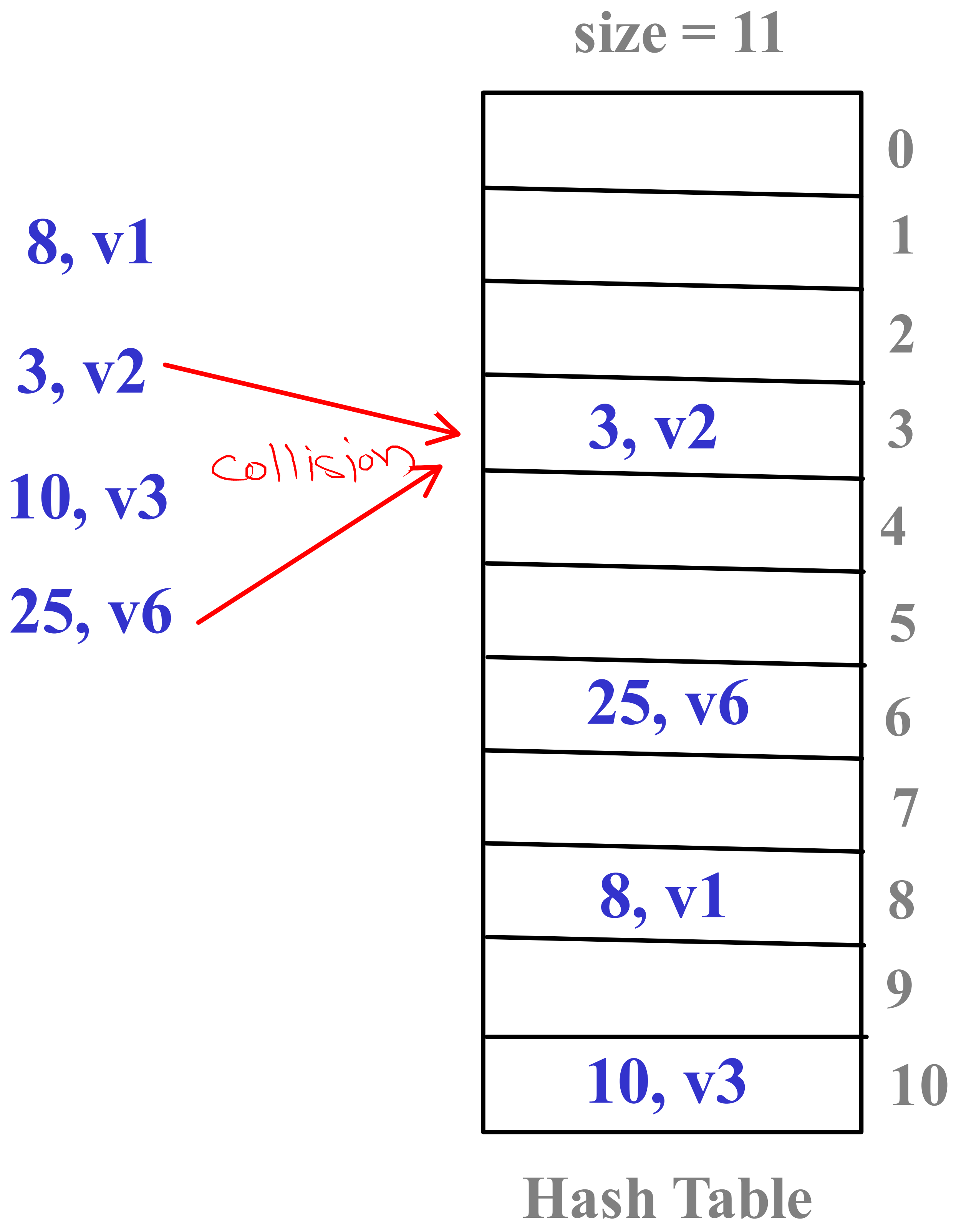
h(33,3) = [3+9] % 10 = 2

h(33,4) = [3+16] % 10 = 9

Secondary clustering -

- need long runs of filled slots "away" key position to find free slot

Hashing - Double Hashing



size = 11

Hash Table

$h1(k) = key \ \% \ size$

$h2(k) = 7 - (key \ \% \ 7)$

$h(k, i) = [h1(k) + i * h2(k)] \ \% \ size$

$h1(8) = 8 \% 11 = 8$

$h1(3) = 3 \% 11 = 3$

$h1(10) = 10 \% 11 = 10$

$h1(25) = 25 \% 11 = 3$

$h2(25) = 7 - (25 \% 7) = 3$

$h(25, 1) = [3 + 1 * 3] \% 11$

$= 6$ (1^{st} probe)

Rehashing

$\lambda = 0.75 \rightarrow 75\%$

$\lambda = 0.5 \rightarrow 50\%$

$$\text{Load Factor} = \frac{n}{N}$$

(λ)

n - Number of elements (key value pairs) in hash table
N - Number of slots in hash table

if n < N	Load factor < 1	- free slots are available
if n = N	Load factor = 1	- no free slots
if n > N	Load factor > 1	- can not insert at all

- Rehashing is make the hash table size twice of existing size if hash table is 70 or 75 % full

- In rehashing existing key value pairs are again mapped according to new hash table size