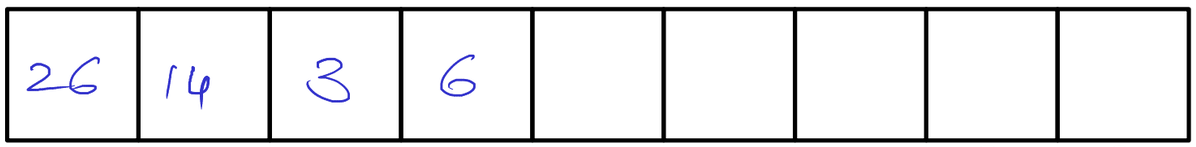
Heap Add and Delete

```
SIZE++;
arr[SIZE] = ele;
int pi = SIZE / 2;
while(pi >= 1) {
        int ci = pi * 2;
        if((ci + 1) <= SIZE && arr[ci + 1] > arr[ci])
                ci = ci + 1;
        if(arr[pi] > arr[ci])
                break;
        int temp = arr[pi];
        arr[pi] = arr[ci];
        arr[ci] = temp;
        pi = pi / 2;
}
```

SIZE
0
1 pi=0
2 pi=1,0 ci=2
3 pi=1 ci=2
4 pi=2,1,0 ci=4,2

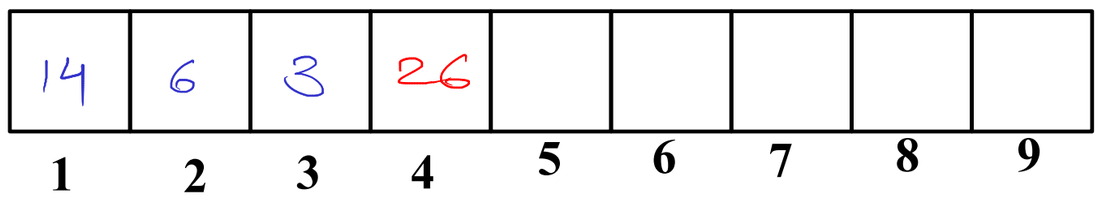
6 14 3 26 8 18 21 9 5



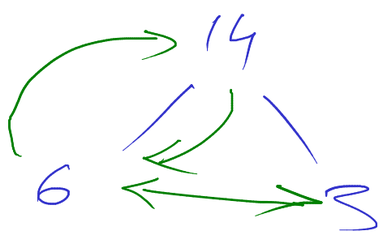
pi=1
ci=1*2=2

ci+1

3>14
X
arr[pi] > arr[ci]



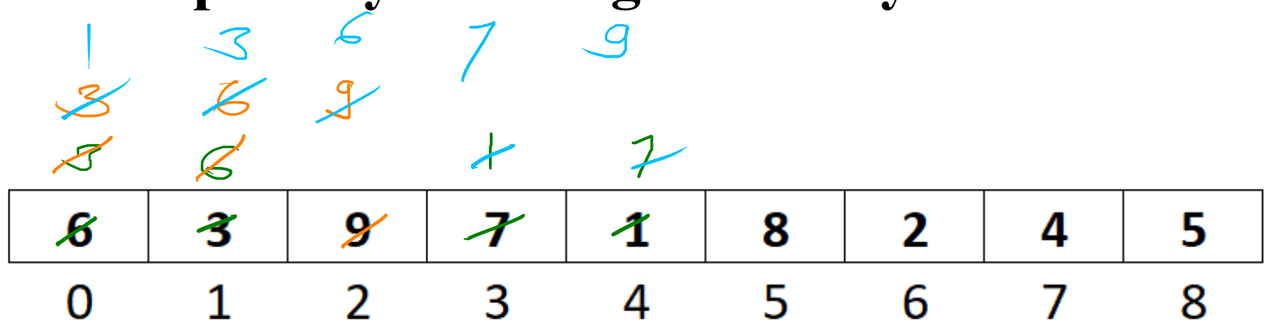
max=26



Merge Sort

Divide and Conquor

//1. Divide collection (array) into two parts
//2. Sort each part individually
//3. merge two sorted partitions in such a way that merged array is sorted
//4. overwrite temp array into original array

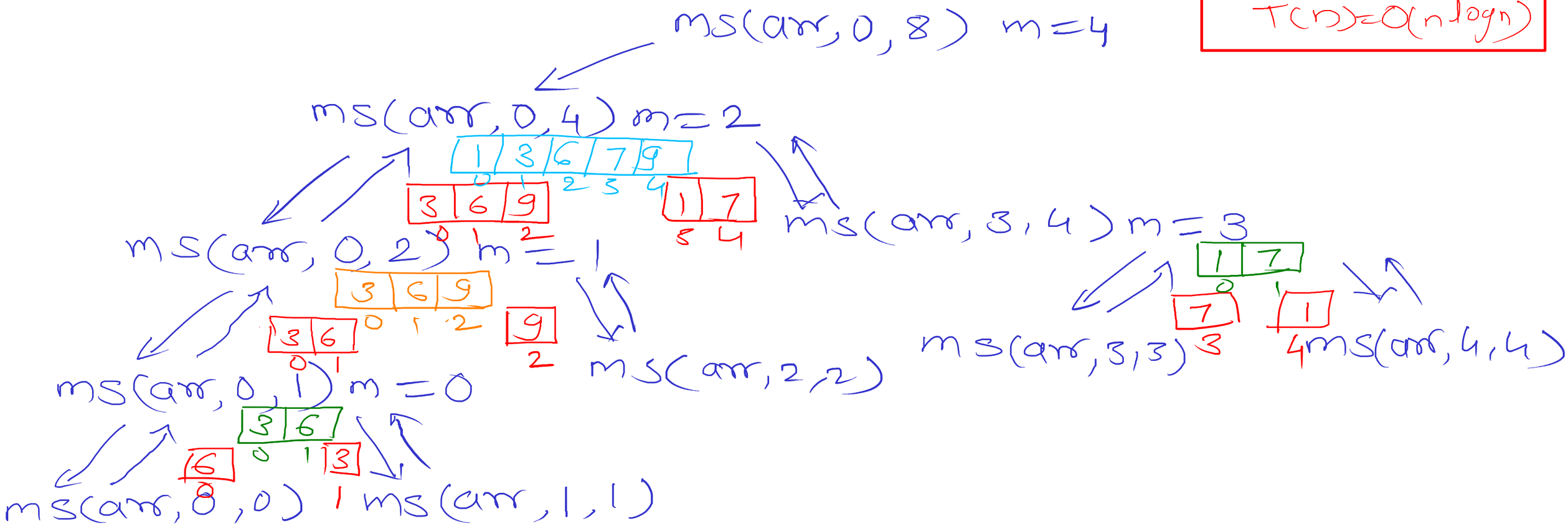


levels of division = log n

comparisions = n per level

total comp = n log n

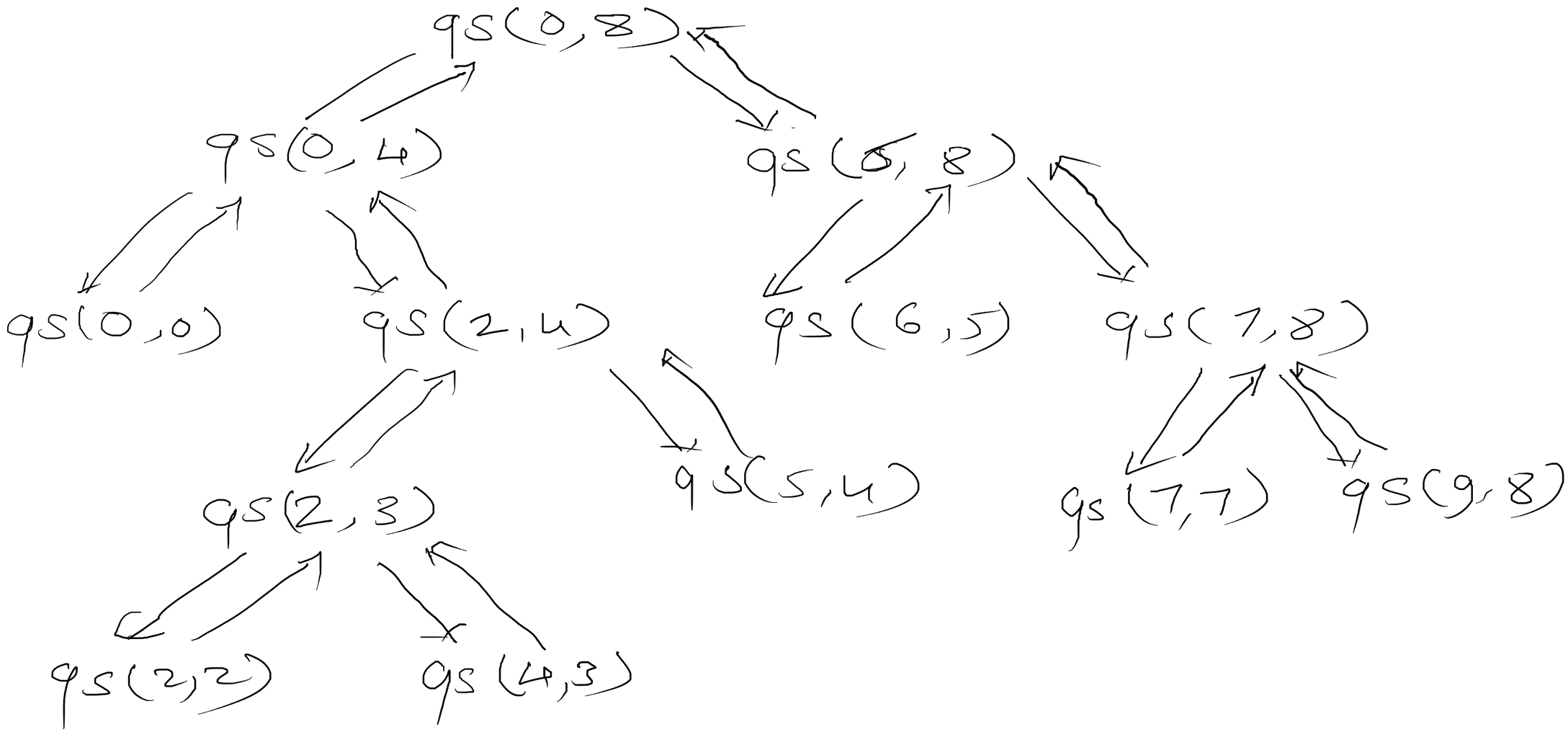
T(n) = O(n log n)



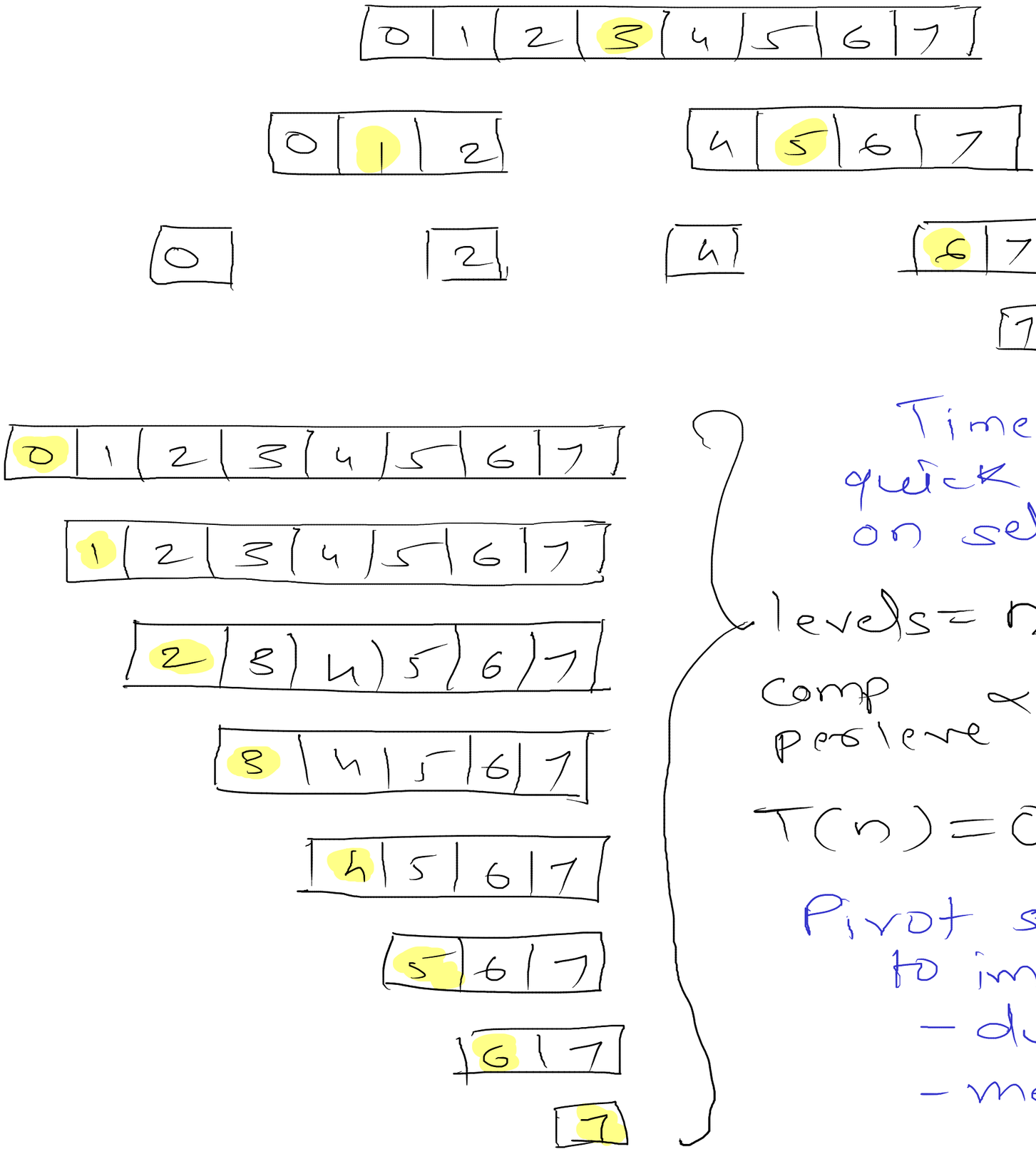
Quick Sort

Divide and Conquor

//1. select pivot (axis/referance) element from array
//2. arrange all smaller elements than pivot to the left side of pivot
//3. arrange all greater elements than pivot to the right side of pivot
//4. sort individual partitions(left and right of pivot) by applying same method



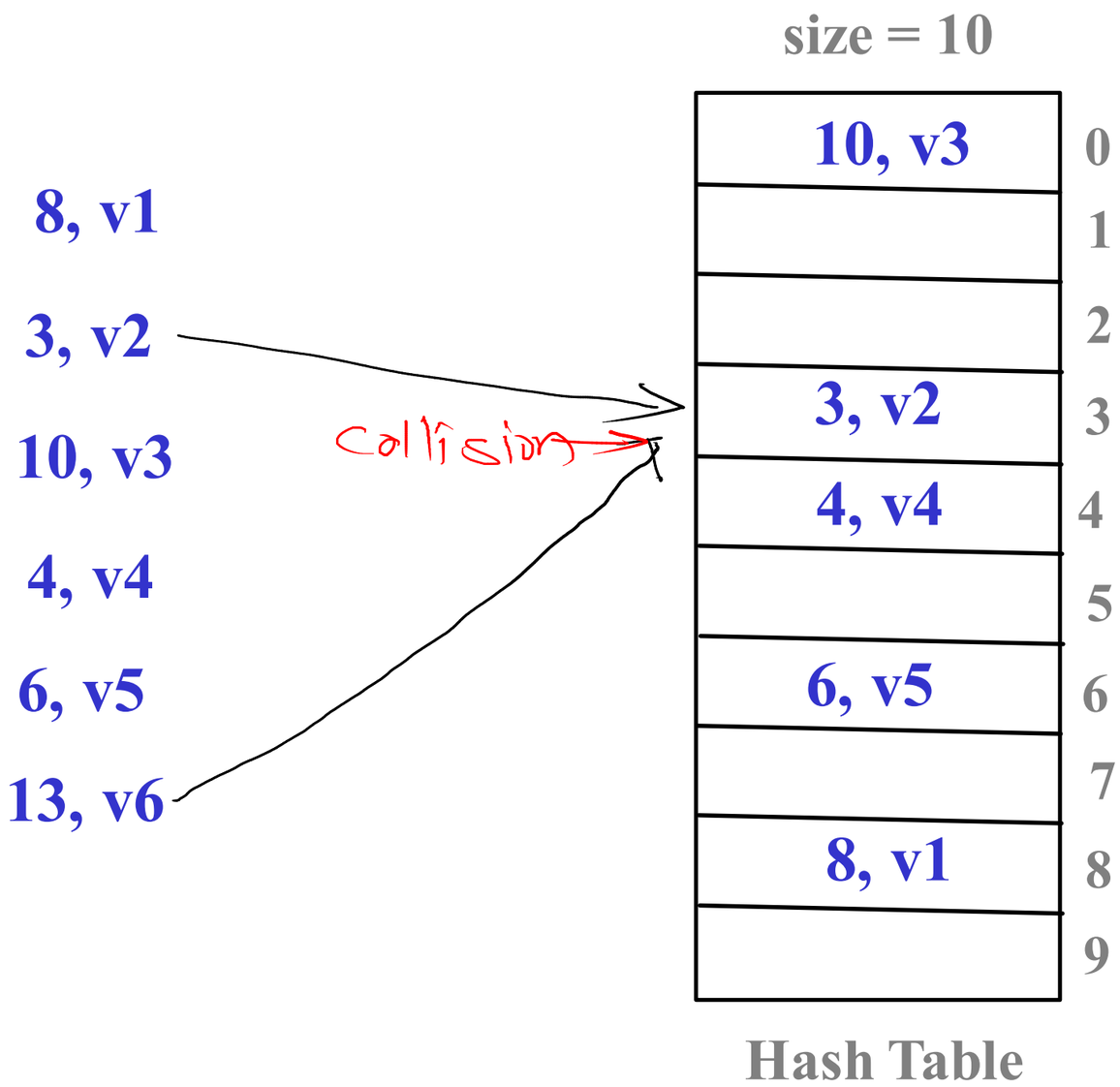
Quick Sort



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

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) = 8 % 10 = 3

h(10) = 10 % 10 = 0

h(4) = 4 % 10 = 4

h(6) = 6 % 10 = 6

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

Closed Addressing/ Seperate Chaining / Chaining

8, v1

3, v2

10, v3

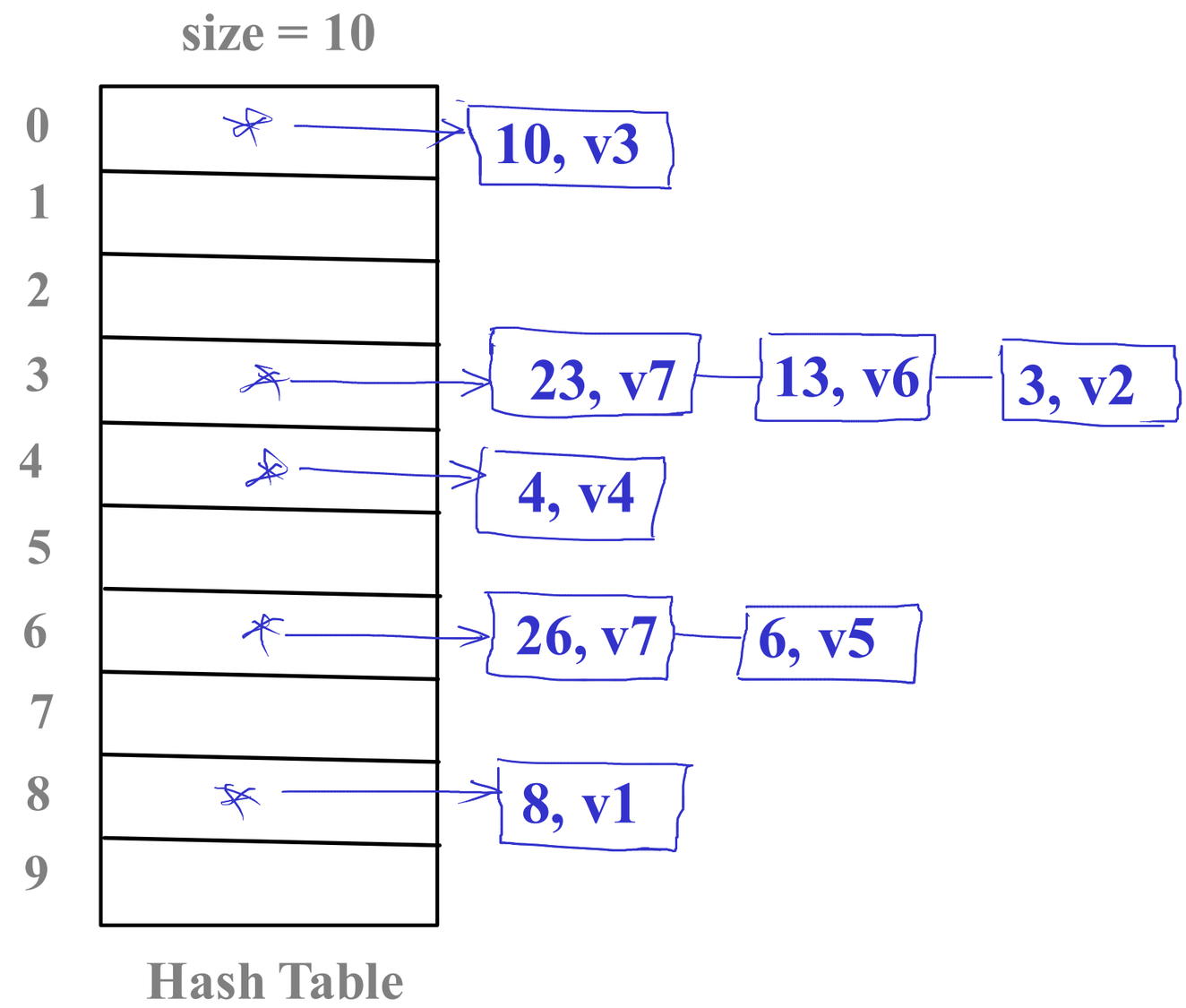
4, v4

6, v5

13, v6

23, v7

26, v7



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