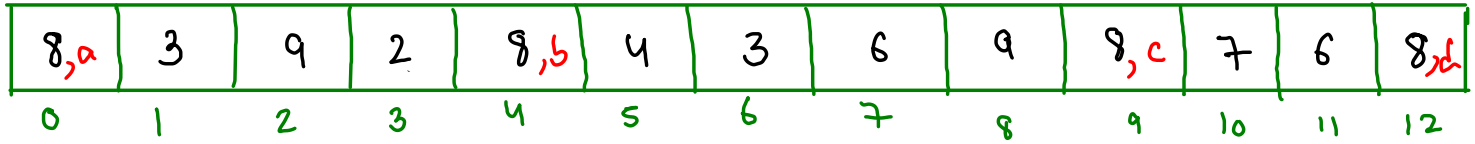
Count Sort

min = 2 max = 9

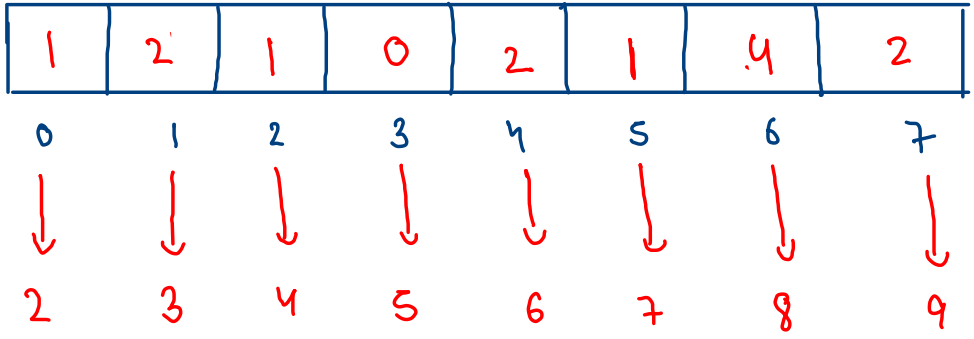
ele = 9 - 2 + 1 = 8



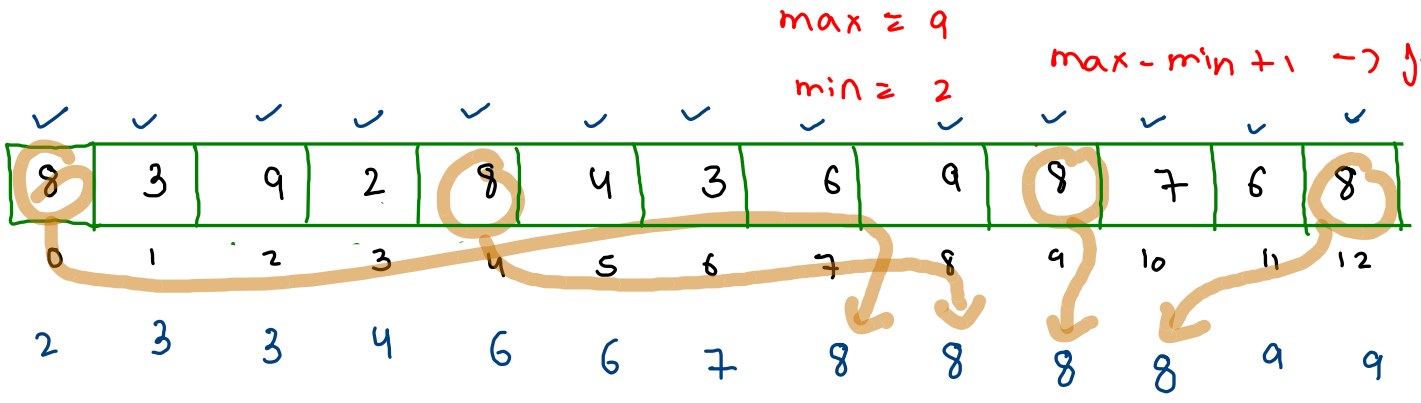
ans: 2 3 3 4 6 6 7 8 8 8 8 9 9

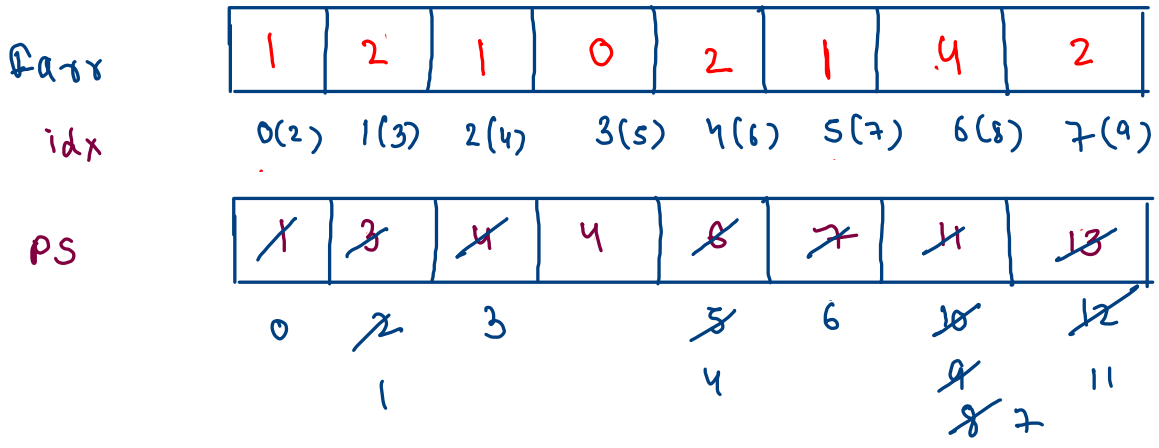
a b c d

arr[i] - min



stable sort



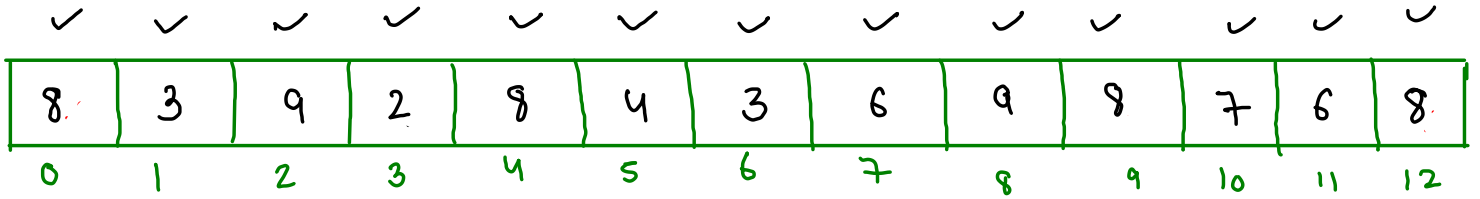


(i) freq. array

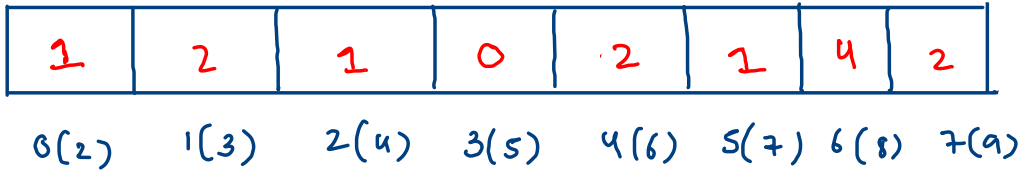
(ii) generate prefix sum array from freq arr.

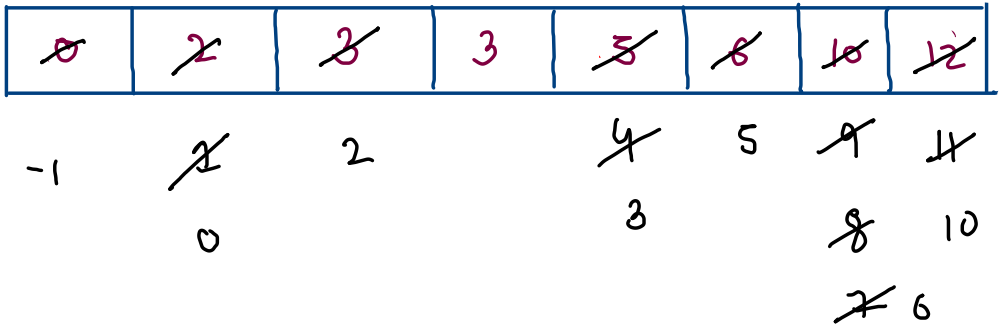
PS[i] = PS[i-1] + farr[i]

(iii) create ans (Stable sorted)



ans: 2 3 3 4 6 6 7 8 8 8 8 9 9






```
int[]ans = new int[arr.length];

for(int i=arr.length-1;i >= 0;i--) {
    int pos = ps[arr[i]-min];
    ans[pos] = arr[i];
    ps[arr[i]-min]--;
}

//copy ans array to original array
for(int i=0; i < arr.length;i++) {
    arr[i]=ans[i];
}
```

min = 2

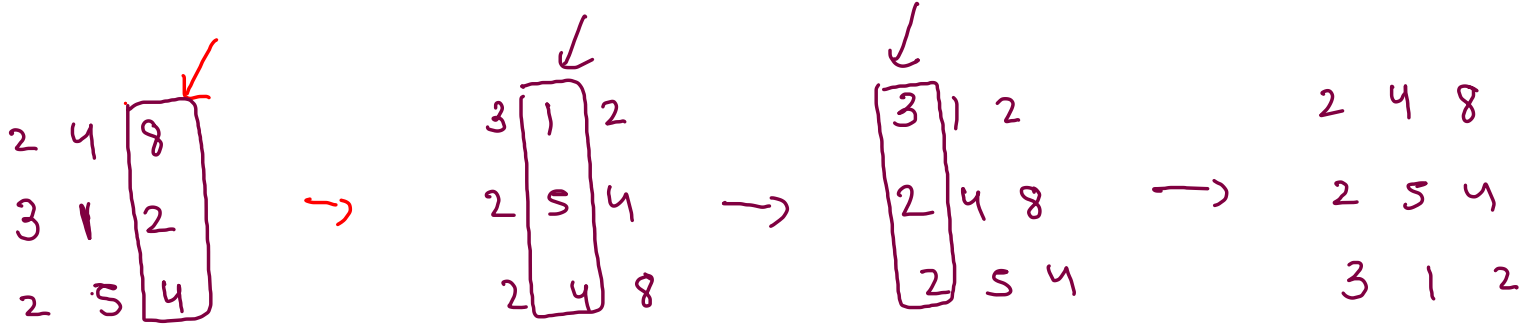
```
int len = max - min + 1;
int[]farr = new int[len]; //freq array

for(int i=0; i < arr.length;i++) {
    farr[arr[i]-min]++;
}
```

```
//generate prefix sum array of freq array
int[]ps = new int[len];
ps[0] = farr[0];

for(int i=1; i < farr.length;i++) {
    ps[i] = ps[i-1] + farr[i];
}

//pos in this ps array are value based, not idx based
for(int i=0; i < farr.length;i++) {
    ps[i] -= 1;
}
```

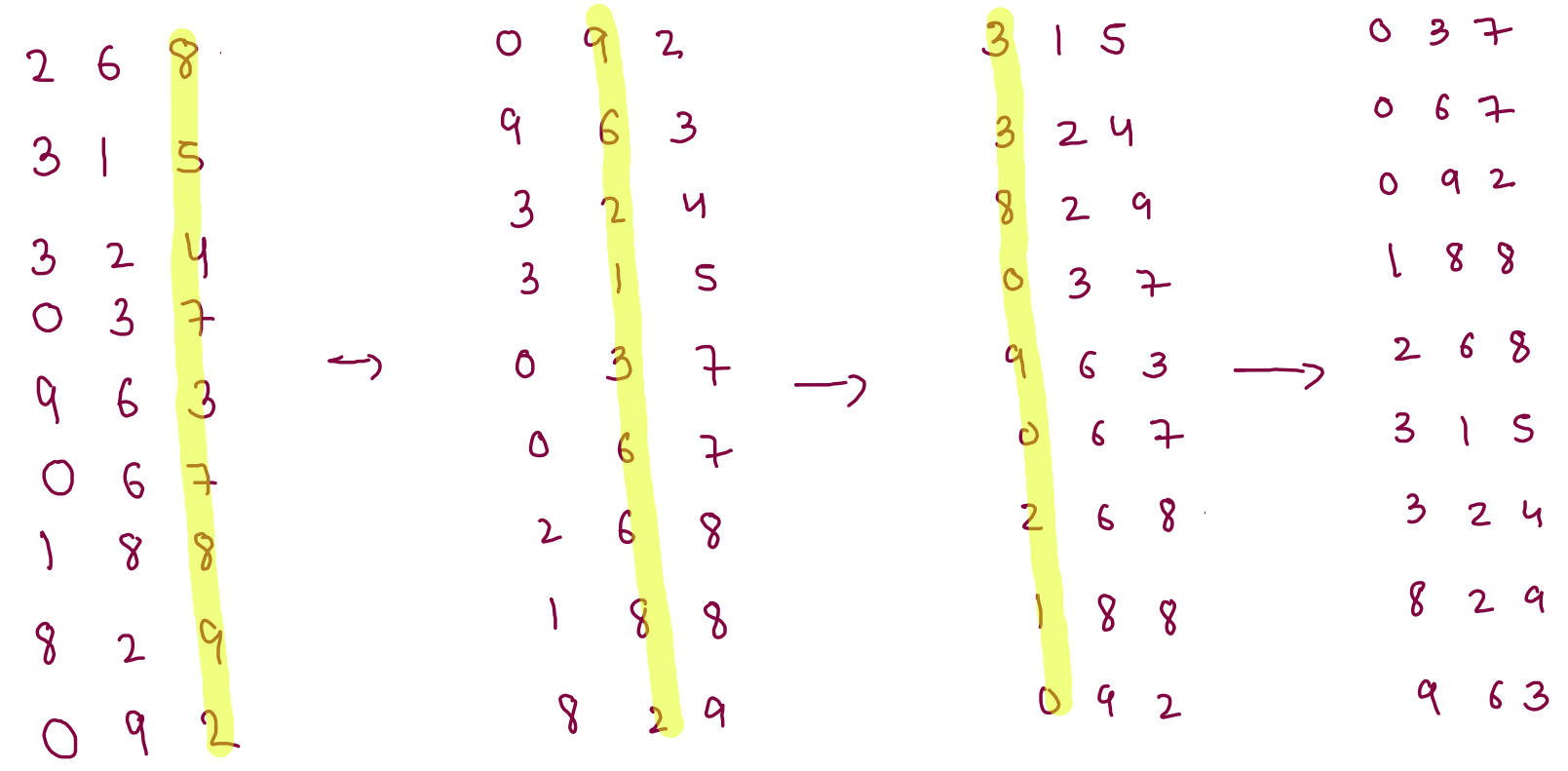
```
int[]ans = new int[arr.length];

for(int i=arr.length-1;i >= 0;i--) {
    int pos = ps[arr[i]-min];
    ans[pos] = arr[i];
    ps[arr[i]-min]--;
}

//copy ans array to original array
for(int i=0; i < arr.length;i++) {
    arr[i]=ans[i];
}
```

Radix Sort

O → t → H



Radix sort


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

9 7 8 4 3

exp = 1 $\quad 97843 \cdot / \cdot 10 \rightarrow 3$

exp = 10 $\quad \left(\frac{97843}{10}\right) \cdot / \cdot 10 \rightarrow 4$

exp = 100 $\quad \left(\frac{97843}{100}\right) \cdot / \cdot 10 \rightarrow 8$

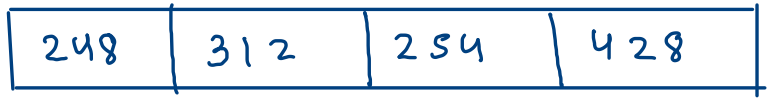
exp = 1000 $\quad \left(\frac{97843}{1000}\right) \cdot / \cdot 10 \rightarrow 7$

exp = 10000 $\quad \left(\frac{97843}{10000}\right) \cdot / \cdot 10 \rightarrow 9$

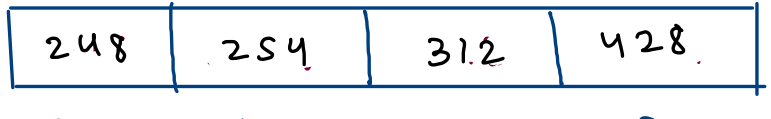
$$d = \left(\frac{n}{exp}\right) \cdot / \cdot 10$$

```java
public static void countSort(int[] arr, int exp) {
  // write code here

int[]farr = new int[10]; //freq array 0 to 9

for(int i=0; i < arr.length;i++) {
    int d = (arr[i] / exp) % 10;
    farr[d]++;
}

//generate prefix sum array of freq array
int[]ps = new int[10];
ps[0] = farr[0];

for(int i=1; i < farr.length;i++) {
    ps[i] = ps[i-1] + farr[i];
}

//pos in this ps array are value based, not idx based
for(int i=0; i < farr.length;i++) {
    ps[i] -= 1;
}

int[]ans = new int[arr.length];

for(int i=arr.length-1;i >= 0;i--) {
    int d = (arr[i] / exp) % 10;
    int pos = ps[d];
    ans[pos] = arr[i];
    ps[d]--;
}

//copy ans array to original array
for(int i=0; i < arr.length;i++) {
    arr[i]=ans[i];
}
```

```java
int exp = 1;
while(d > 0) {
    countSort(arr,exp);
    exp = exp*10;

    d--;
}
```



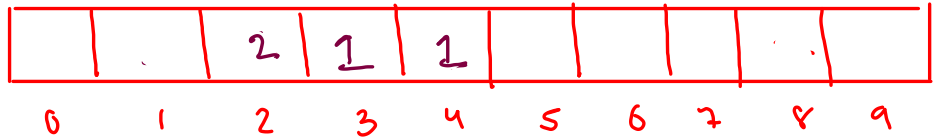
arr:



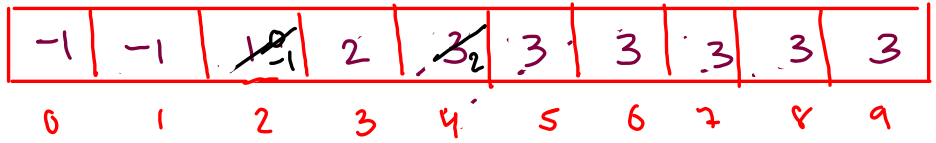
ans :

exp = 100

farr



ps



merge sort; $T(n) = c + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$

c → mid; $T\left(\frac{n}{2}\right)$ → left; $T\left(\frac{n}{2}\right)$ → right; n → merging

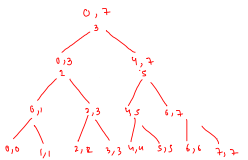
$T(n) = c + 2T\left(\frac{n}{2}\right) + n$ ~~+n~~

$2T\left(\frac{n}{2}\right) = 2c + 4T\left(\frac{n}{4}\right)$ ~~+n~~ ~~+n~~ $\quad n \to \frac{n}{2} \quad \times 2$

$4T\left(\frac{n}{4}\right) = 4c + 8T\left(\frac{n}{8}\right)$ ~~+4n~~ ~~+4n~~ $\quad n \to \frac{n}{2} \quad \times 4$

$\vdots$

$T(1) = c$



$n \to \frac{n}{2} \to \frac{n}{4} \to \frac{n}{8} \dots\dots 1$

GP; $a = n$ $\quad t$ (total terms)

$r = \frac{1}{2}$

$t_t = a r^{t-1}$

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

$2^{t-1} = n$

$\log_2 2^{t-1} = \log_2 n$

$t - 1 = \log_2 n$

$\boxed{t = \log_2 n + 1}$

$T(n) = (c + 2c + 4c + \dots\dots) + (n + n + n \dots\dots)$

$= c\left(1 + 2 + 4 + \dots\dots + 2^{\log_2 n}\right) + n\underbrace{(1 + 1 + \dots\dots)}_{t \text{ times}}$

$= c\left(1 + 2 + 4 + \dots\dots + n\right) + n\left(\log_2 n + 1\right)$

$= c\left(2^{\log_2 n + 1} - 1\right) + n\left(\log_2 n + 1\right)$

$= c\left(2^{\log_2 n} \cdot 2 - 1\right) + n\left(\log_2 n + 1\right)$

$= c\left(2n - 1\right) + n\log_2 n + n$

$= 2nc - c + n\log_2 n + n$

$T(n) = n(2c + 1) + n\log_2 n - c$

$\boxed{T(n) \propto n\log_2 n}$

```java
public static void printSS(String str, String ans) {
    if(str.length() == 0) {
        System.out.println(ans);
        return;
    }

    char ch = str.charAt(0);
    String ros = str.substring(1);

    //ch -> yes choice
    printSS(ros, ans + ch);

    //ch -> no choice
    printSS(ros, ans);
}
```

O(1) [VI]

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

ch left right

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

$n \to n-1$

$2T(n-1) = 2c + 4T(n-2)$ $\times 2$

$n \to n-1$

$4T(n-2) = 4c + 8T(n-3)$ $\times 4$

$T(0) = c$

$T(n) = (c + 2c + 4c + \cdots)$

$n \to n-1 \to n-2 \to n-3 \ldots 0$

n steps

$t = n+1$

$T(n) = c(1 + 2 + 4 + \cdots + 2^n)$

$= c(2^{n+1} - 1)$

$T(n) = 2^{n+1} \cdot c - c$

$T(n) \propto 2^n$

$a = 1$

$r = 2$

$t = n+1$

$\frac{a(r^t - 1)}{r-1} = \frac{1(2^{n+1} - 1)}{1}$

Binary search

```java
public static int binarySearch(int[]arr,int lo,int hi,int data) {
    if(lo > hi) {
        return -1;
    }

    int mid = (lo + hi) / 2;

    if(arr[mid] == data) {
        return mid;
    }
    else if(arr[mid] < data) {
        int ans = binarySearch(arr,mid+1,hi,data);
        return ans;
    }
    else {
        int ans = binarySearch(arr,lo,mid-1,data);
        return ans;
    }
}
```

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

$n \to \frac{n}{2}$

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

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

$\vdots$

$T(0) = c$

$n \to \frac{n}{2} \to \frac{n}{4} \to \cdots \to 1$

$t = \log_2 n + 1$

$T(n) = c + c + c + \cdots\cdots$

$= c(\log_2 n + 1)$

$\boxed{T(n) \propto \log_2 n}$

```java
public static void bubbleSort(int[] arr) {
  //write your code here

  int n = arr.length;
  for(int itr = 1; itr < n;itr++) {
      for(int j=0; j < n - itr;j++) {
          if(isSmaller(arr,j+1,j) == true) {
              //swapping is required
              swap(arr,j+1,j);
          }
      }
  }

}
```

itr	comparison
1	$n-1$
2	$n-2$
3	$n-3$
4	$n-4$

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

$$T(n) = n-1 + n-2 + n-3 + n-4$$

$$= \text{sum of } (n-1) \text{ natural no.}$$

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

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

$$T(n) \propto n^2$$

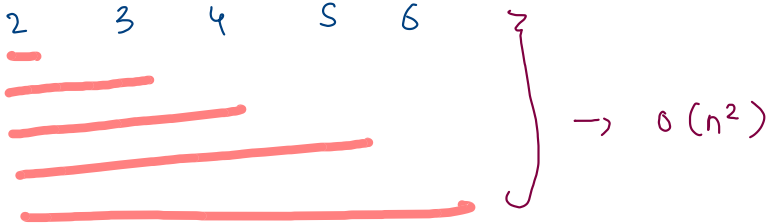
```
int j = 1 ;
for (int i = 1 ; i <= n ; i++) {
    while (j <= n) {
        i++ ;
        j++ ;
    }
}
```

n = 4

~~2 3 4 5 6~~

i = ~~1~~ , j = ~~1 2 3 4~~ 5

O(n)



→ $O(n^2)$

→ O(n)

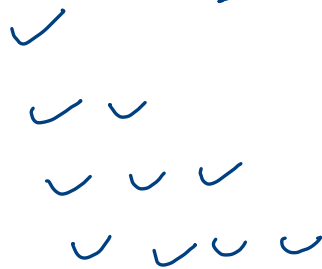
```
for (int i = 1 ; i <= n ; ) {
        if (j == i) {
            i++;
            j = 1;
        }
        else {
            j++;
        }
}
```

$O(n^2)$

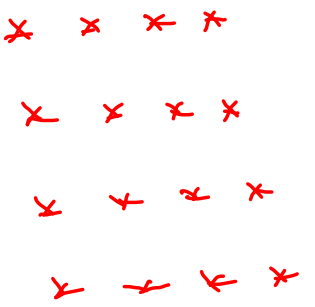
$n = 4$

i = 1 2 3 4 5

j = 1 2 1 2 3 1 2 3 4 1



```
for (int i = 1 ; i <= 4; i++ ) {

    for (int j = 1 ; j <= 4; j++) {

    }

}
```

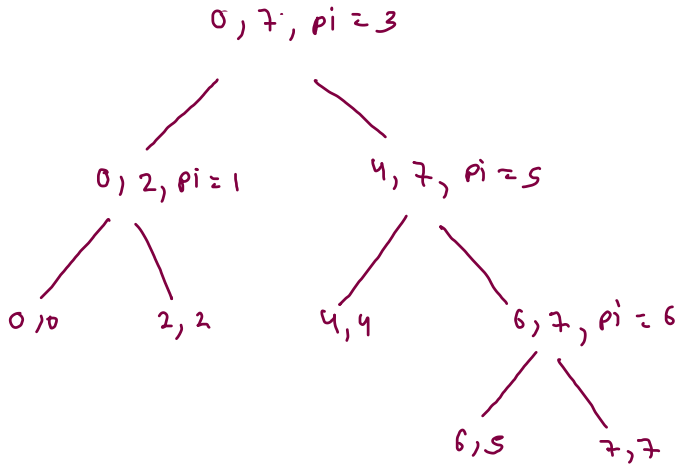


$$T(n) = n + T(n-1)$$

$$T(n) \propto n^2$$

t c	eq^n	eg.
$n \log n$	$T(n) = c + n + 2T\left(\frac{n}{2}\right)$	merge sort
2^n	$T(n) = c + 2T(n-1)$	subseq
n^2	$T(n) = n + T(n-1)$	bubble, selection, insertion
$\log n$	$T(n) = c + T\left(\frac{n}{2}\right)$	binary search
n	$T(n) = c + T(n-1)$	max, linear search

quick sort (best case)



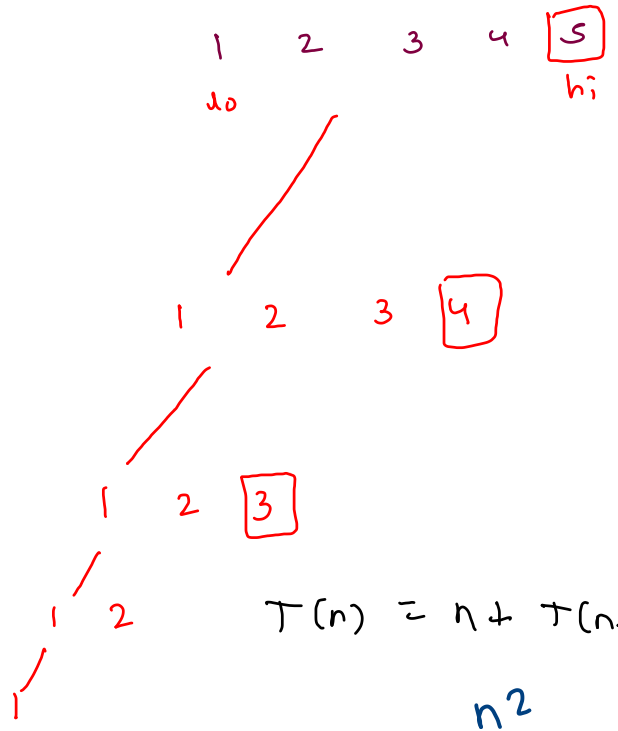
$T(n) = n + 2T\left(\frac{n}{2}\right)$

2 5 1 4 3

nlogn

2 1 3 5 4

quick sort (worst case)



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

n^2

nlogn to n^2