```java
public static int[] mergeSort(int[] arr, int lo, int hi) {
    //write your code here
    if (lo == hi) {
        int[] ba = {
            arr[lo]
        };
        return ba;
    }

    int mid = (lo + hi) / 2;

    int[] left = mergeSort(arr, lo, mid);
    int[] right = mergeSort(arr, mid + 1, hi);

    int[] merged = mergeTwoSortedArrays(left, right);

    return merged;
}
```

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

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \text{— (1)}$$

$n \to \frac{n}{2}$ in (1)

$$2T\left(\frac{n}{2}\right) = 4T\left(\frac{n}{4}\right) + n \quad \text{— (2)} \times 2$$

$n \to \frac{n}{4}$ in (1)

$$4T\left(\frac{n}{4}\right) = 8T\left(\frac{n}{8}\right) + n \quad \text{— (3)} \times 4$$

$$\vdots \qquad \vdots$$

$$T(1) = c$$

$\times$

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

GP, $a = n$, $r = \frac{1}{2}$

last term $= ar^{x-1}$

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

$$2^{x-1} = n$$

$$x - 1 = \log_2 n$$

$$\boxed{x = (\log_2 n + 1)}$$

$$T(n) = c + xn$$

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

$$\boxed{T(n) = c + n\log_2 n + n}$$

$$T(n) \propto n\log_2 n$$

```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 -> present
    printSS(ros,ans + ch);

    //ch -> absent
    printSS(ros,ans);
}
```

n -> str. length abc (n=3)

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

$T(n) = n + 2T(n-1)$ ——— ①

$2T(n-1) = 2(n-1) + 4T(n-2)$ ——— ② (n -> n-1 in ①) × 2

$4T(n-2) = 4(n-2) + 8T(n-3)$ ——— ③ (n -> n-2 in ①) × 4

$T(0) = c$

n, n-1, n-2, n-3, ... 0

x = n

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

$T(n) = n + 2n - 2 + 4n - 8 + 8(n-3) + 16(n-4) + \dots c$

$T(n) = n + 2n + 4n + 8n + 16n \dots + (-2 - 8 - 24 - 48 \dots) + c$

$T(n) = n(1 + 2 + 4 + 8 + 16 \dots 2^{n-1}) + c_1$

GP, $a = 1$, $r = 2$, last $= ar^{x-1}$, last $= 2^{n-1}$

$= n(2^n - 1)$

$T(n) = n2^n - n$

$T(n) \propto n2^n$

$a = 1$

$r = 2$

$x = n$

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

$= 1(2^n - 1)$

$= 2^n - 1$

$n = 100$

$8(n-3) + 16(n-4) + 32(n-5) + 64(n-6) \dots 2^n(n-n)$

$= -2 - 8 - 24 - 64 \dots -n2^n$

$= -2[1 + 4 + 12 + 32 + \dots n2^{n-1}]$

Binary Search

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

c → mid

$T\left(\frac{n}{2}\right)$ → left or right

$n, \frac{n}{2}, \frac{n}{4}, \ldots\ldots 1$

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

$x = \log_2 n + 1$

$\times$

$T(n) = c + T\left(\frac{n}{2}\right)$ — ①

$n \to \frac{n}{2}$ in ①

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

$n \to \frac{n}{4}$ in ①

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

$\vdots$

$T(1) = c$

$+$

$T(n) = cx$

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

$T(n) = c \log_2 n + c$

$T(n) \propto \log_2 n$

```java
public static void printTargetSumSubsets(int[] arr, int idx, String set, int sos, int tar) {
    if(idx == arr.length) {
        if(sos == tar) {
            System.out.println(set + ".");
        }
        return;
    }

    if(sos > tar) {
        return;
    }

    //arr[idx] -> present
    printTargetSumSubsets(arr,idx+1,set + arr[idx] + ", ",sos + arr[idx],tar);

    //arr[idx] -> absent
    printTargetSumSubsets(arr,idx+1,set,sos,tar);
}
```

assumption, we have used SB or RL.

$x = n$

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

$T(n) = c + 2T(n-1)$ — ①

$2T(n-1) = 2c + 4T(n-2)$ — ② × 2

$4T(n-2) = 4c + 8T(n-3)$ — ③ × 4

⋮

$T(1) = c$

(x)

$T(n) = c + 2c + 4c + 8c + \ldots$

$= c(1 + 2 + 4 + \ldots + 2^{n-1})$

$= c(2^n - 1)$

$T(n) \propto 2^n$

for ( ) {

for ( ) { → O(n)

}

}

for(int i = 1 ; i <= n ;){

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

i++;

}

}

O(N)

N = 10

i = 1, j = 1 to 10

2

3

4

5

6

7 8 9 10

for ( ) → O(n²)

for (int i = 1, j = 1 ; i <= n ;) {

if (j < i) {

j++ ;

}

else {

i++;

j = 1;

}

}

N = 4

n-3 i = 1, j = 1

+

n-2 i = 2, j = 1, 2

+

n-1 i = 3, j = 1, 2, 3

+

n i = 4, j = 1, 2, 3, 4

i = 5, j = 1

$Sum = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$

$T(n) \propto n^2$

N = 5

```
for (int it = 1; it < n; it++) {
    for (int j = 0; j < n - it; j++) {
        if (isSmaller(arr, j + 1, j) == true) {
            //arr[j+1] is smaller than arr[j]
            swap(arr, j + 1, j);
        }
    }
}
```

it = 1 , comp = n-1

it = 2 , comp = n-2

it = 3 , comp = n-3

it = 4 , comp = n-4

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

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

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

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

Complexity	Recurrence	eg.
$\log n$	$T(n) = c + T\left(\frac{n}{2}\right)$	Binary search, power-log
$n \log n$	$T(n) = n + 2T\left(\frac{n}{2}\right)$	merge sort, quick sort
n^2	$T(n) = C(n-1) + T(n-1)$	bubble, selection
2^n	$T(n) = c + 2T(n-1)$	subseq
n	$T(n) = c + T(n-1)$	power-linear, max, linear-search.

```java
public static int quickSelect(int[] arr, int lo, int hi, int k) {
    //write your code here

    int pi = partition(arr, arr[hi], lo, hi);

    if (pi == k) {
        return arr[pi];
    } else if (pi < k) {
        return quickSelect(arr, pi + 1, hi, k);
    } else {
        return quickSelect(arr, lo, pi - 1, k);
    }
}
```

$T(n) = n + T\left(\frac{n}{2}\right)$ — ①

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

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

$\vdots$

$T(1) = c$

$T(n) = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \cdots \cdots$

$\log_2 n$

$a = n$

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

$x = \log_2 n$

$$S = \frac{a(1-r^x)}{1-r}$$

$$S = \frac{n\left(1 - \frac{1}{2^{\log_2 n}}\right)}{1 - \frac{1}{2}}$$

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

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

$T(n) \propto n$

space — complexity :

recursion

✗ O(1) ⟶ call stack

extra space

1. reverse an array ⟶ ✗ O(1)

2. inverse an array ⟶ O(n) ✓

3. rotate an array ⟶ ✗ O(1)