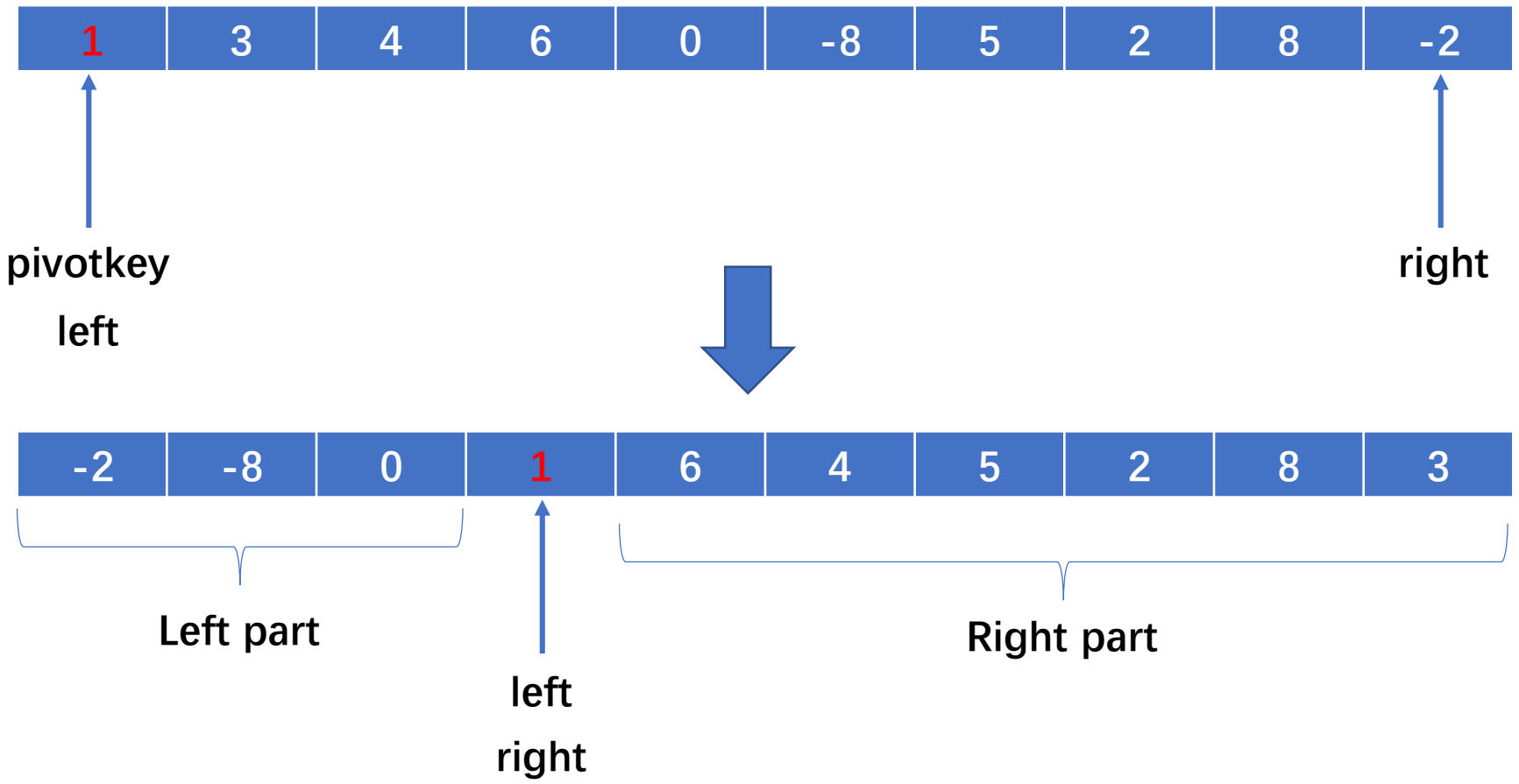
1
3
4
6
0
-8
5
2
8
-2
pivotkey
left
right
-2
-8
0
1
6
4
5
2
8
3
Left part
left
right
Right part

每次递归需要的东西就是left，right，array是全局都有的

```
// recursive
int quick_sort_recursive(int array[], int left, int right) {
    if (left < right) {
        int partial = partition(array, left, right);
        quick_sort_recursive(array, left, partial - 1);
        quick_sort_recursive(array, partial + 1, right);
    }
}
```

每次进入递归时，有的信息是array和left、right，说明模拟递归时需要pop出left和right，然后进行一次划分。

划分完之后进入递归，也就是用栈保存下一次需要用的信息，以便pop时能得到保存的信息

```
Rec *rec0 = (Rec *) malloc(sizeof(rec));
rec0->left = left;
rec0->right = right;
stack[++top] = rec0; // push into stack
```

```
// recursive
int quick_sort_recursive(int array[], int left, int right) {
    if (left < right) {
        int partial = partition(array, left, right);
        quick_sort_recursive(array, left, partial - 1);
        quick_sort_recursive(array, partial + 1, right);
    }
}
```

```
Rec *rec1 = stack[top--]; // pop
int partitial = partition(array, rec1->left, rec1->right);
```

```
// if left part can be recursive
if (rec1->left < partitial - 1) {
    Rec *rec2 = (Rec *) malloc(sizeof(rec));
    rec2->left = left;
    rec2->right = partitial - 1;
    stack[++top] = rec2; // push into stack
}

// if right part can be recursive
if (rec1->right > partitial + 1) {
    Rec *rec3 = (Rec *) malloc(sizeof(rec));
    rec3->left = partitial + 1;
    rec3->right = right;
    stack[++top] = rec3; // push into stack
}
```