Quick Sort

In this lesson, we'll learn how quick sort works and its implementation.

We'll cover the following

Divide and conquer

Quick sort is another divide and conquer algorithm, like merge sort. It picks an element and partitions the array around it, then recursively sorts the two partitions. This will become clearer with an example.

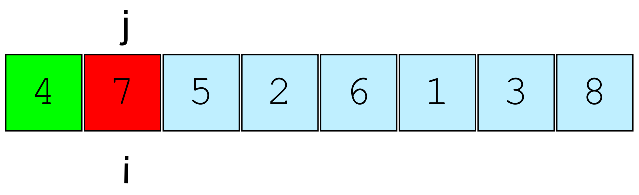
Pivot

To partition, we pick one element as the pivot. There are several ways to do this:

- Take the first element
- Take the last element
- Pick a random element as pivot
- Pick the median as the pivot

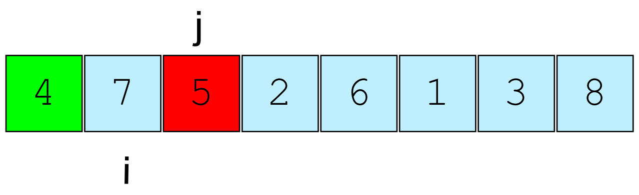
After picking the pivot, we need to partition the array into two halves with a pivot in the middle such that the left part only has elements less than the pivot and the right part has elements only greater than the pivot. Then recursively do this for the left and right partition.

The illustrations below pick the first element as the pivot.



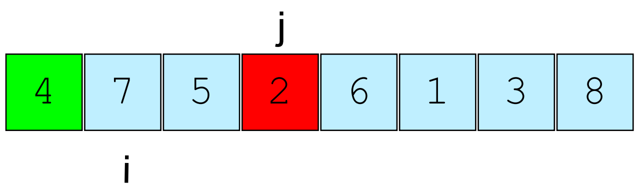
A[j] > pivot, do nothing

1 of 13



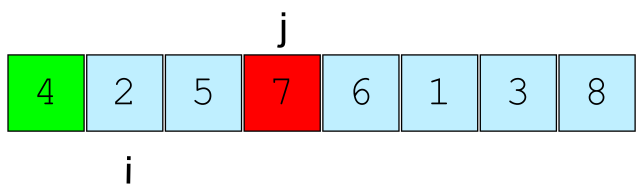
A[j] > pivot, do nothing

2 of 13

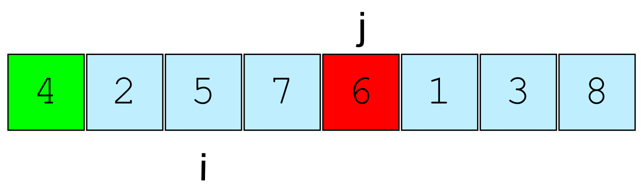


A[j] < pivot, swap (A[i], A[j])

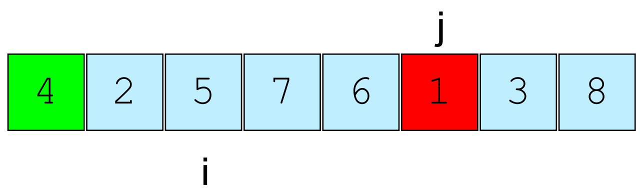
3 of 13



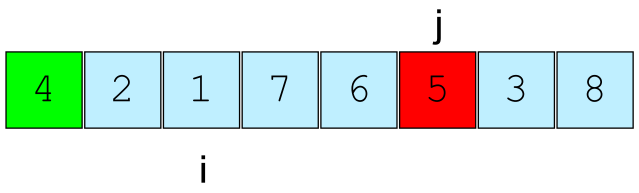
i ++



A[j] > pivot, do nothing

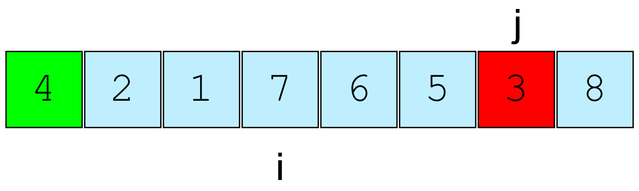


A[j] < pivot, swap(A[i], A[j])



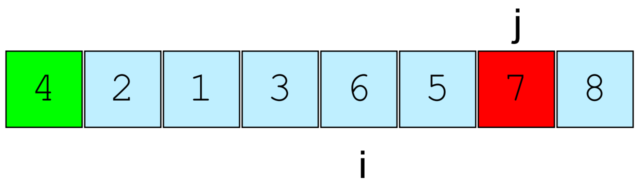
i++

7 of 13



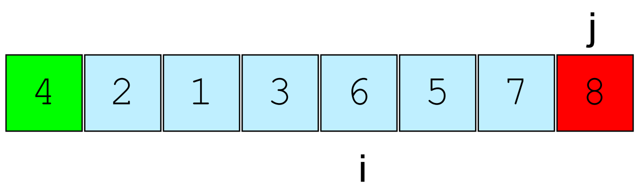
A[j] < pivot, swap (A[i], A[j])

8 of 13



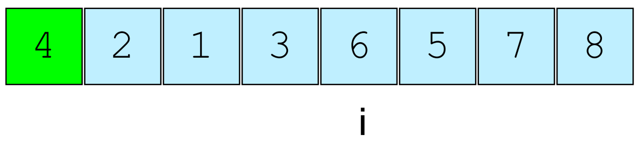
i++

9 of 13



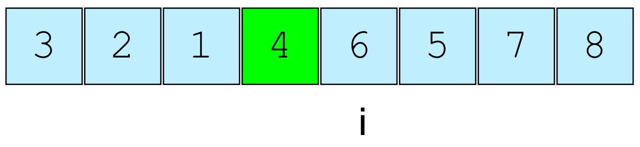
A[j] > pivot, do nothing

10 of 13



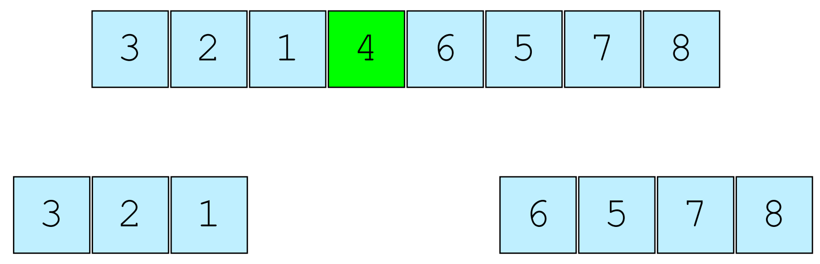
Pivot moves to (i-1)

11 of 13



Pivot moves to (i-1), swap(pivot, A[i-1])

12 of 13



recursively do for left and right part

13 of 13

```
#include <iostream>
using namespace std;

int partition(int arr[], int s, int e) {
  int i = s + 1,j = s + 1;
  int pivot = s;

  while (j <= e) {
    if (arr[j] < arr[pivot]) {
      swap(arr[j], arr[i]);
      i ++;
    }
    j ++;
  }
  swap(arr[pivot], arr[i-1]);
  return i - 1;
}

void quick_sort(int arr[], int s, int e) {
  if (s >= e)
    return;
  int p = partition(arr, s, e);

  quick_sort(arr, s, p - 1);
  quick_sort(arr, p + 1, e);
}

int main() {
  int N = 8;
  int arr[N] = {4, 7, 5, 2, 6, 1, 3, 8};

  quick_sort(arr, 0, N-1);

  for (int i = 0; i < N; i++)
    cout << arr[i] << " ";

  return 0;
}
```

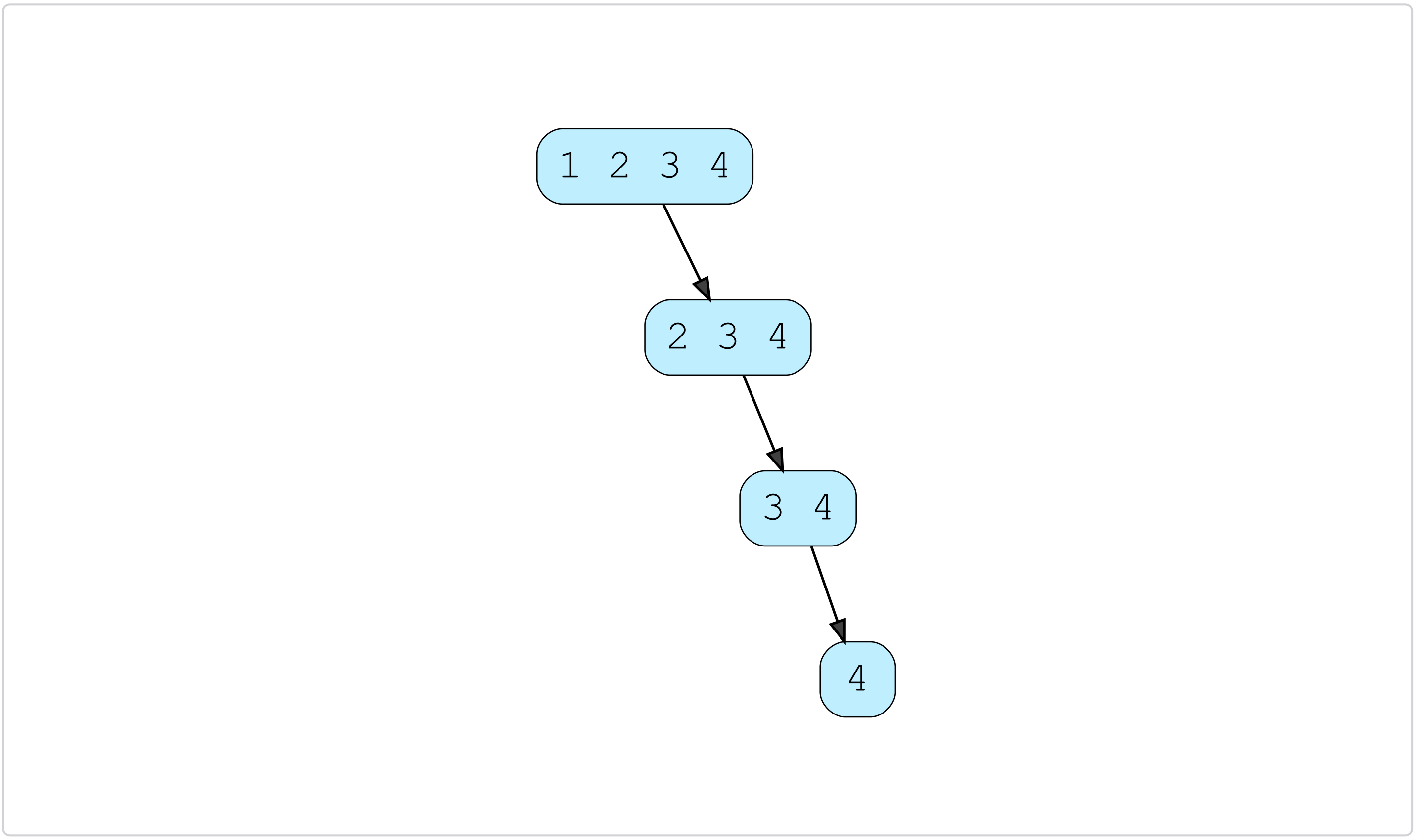
Explanation

Easy termination step -> when array size is 1 (`s >= e`). Otherwise, we take the first element as a pivot, partition as explained in the illustration.

After we partition at pivot `p`, we recursively call for two parts: `quick_sort(arr, s, p - 1)` and `quick_sort(arr, p + 1, e)`.

Time complexity

To analyze the worst case scenario when we pick the first element as the pivot, think about what happens when the array is already sorted. Similar to merge sort analysis, each level in the recursion tree has $O(N)$ elements. In the case of a sorted array, there would be N levels.



The worst-case complexity turns out to be $O(N^2)$.

No matter what pivot scheme you define, some input array will sort in $O(N^2)$.

Solution: Pick random pivots. The worst case is still $O(N^2)$ but the

Solution: Pick random pivots. The worst case is still $O(N^2)$, but the average case is $O(NlogN)$. Many of the algorithms and data structures, like Treap, are built on these probabilities that the chances for random pivots to pick the worst position is slim and good enough for use in competitions.

In the next lesson, we'll discuss the applications in competitions.