# 排序算法总结

# Sort algorithm summary:

## 快速排序quicksort

Quicksort is a divide-and-conquer algorithm, it is also a comparison sort.

Its central idea is to select a pivot and divide the whole array into two sub-arrays. The left sub-array’s elements should be less than this pivot, and the right sub-arrays elements should be greater than this pivot. For every array, we find the first smaller element from the right side, select the first greater element from the left side, exchange their position, and continue search until the current array become sorted.

Average Time complexity O(nlogn), in the worst case O(n^2)

Public void quicksort(int[] nums, int l, in r){

If(l>r) return;

//一般 l=0, r = nums.length()-1

int i = recur(nums, l, r);

//Tail call优化

//由于普通快速排序每轮选取「子数组最左元素」作为「基准数」，因此在输入数组

//完全倒序，partition() 的递归深度会达到N ，即 最差空间复杂度 为)O(N) 。

//每轮递归时，仅对 较短的子数组 执行哨兵划分 partition() ，就可将最差的递归深度//控制在O(logN)

**if( i-l < r-i){**

**recur(nums, l, i-1);**

**l = i+1;**

**}**

**else{**

**recur(nums,i+1,r);**

**r=i-1;**

**}**

}

Public int recur(int[] nums, int l, int r){

int i = l, j = r;

//确定此时的主元为l， the pivot here is ‘l’

while(i<j){

while(i<j && nums[l]>nums[j]) j--;

while(i<j && nums[l]< nums[i]) i++;

swap(nums[i], nums[j]);

}

swap(nums[l], nums[i]);

return i;

}

Public void swap(int[] nums, int i, int j) {

// 交换 nums[i] 和 nums[j]

int tmp = nums[i];

nums[i] = nums[j];

nums[j] = tmp;

}

另外一种写法,把quicksort和recur写在一起,以题目为例子：

找到并且返回一个数组中前最小K个数字组成的子数组。

相比于用java内置函数Arrays.sort()，快速排序时间复杂度更低.

public int[] getLeastNumbersByQuickSort(int[] arr, int k) {

if(arr.length==0 || k<=0 ) return new int[0];

if(k>= arr.length) return arr;

return quickSort(arr,k,0,arr.length-1);

}

public int[] quickSort(int[] arr, int k, int l, int r){

int i=l, j=r;

while(i<j){

while(i<j && arr[j]>=arr[l]) j--;

while(i<j && arr[i]<=arr[l]) i++;

swap(arr,i,j);

}

swap(arr,i,l);

**if(i>k) return quickSort(arr,k,l,i-1);**

**if(i<k) return quickSort(arr,k,i+1,r);**

**return Arrays.*copyOf*(arr,k);**

**//上面这个就是采用tail call的优化**

}

public void swap(int[] arr, int a, int b){

int temp = arr[a];

arr[a] = arr[b];

arr[b] = temp;

}

## 采用大/小顶堆的数据结构

In java, PriorityQueue will also be used to store sotted numbers. In general, PriorityQueue is a heap that starts from smallest number to the greatest number. We can look it like a full search binary tree, the root of this tree is the smallest. When offering a element, PriorityQueue will automatically adjust the order of its elements. When building a PriorityQueue, you can change its order by using comparator.

PriorityQueue<Integer> q = new PriorityQueue<>((v1,v2)->v2-v1);

在java中，PriorityQueue<>()默认是小顶堆,即数组是从大到小排列，可以在创建时把它变成小顶堆。由于优先队列本质是一个完全二叉树结构，在插入数据时交互🌲的节点是的其变为有序。所以不需要自己给他排序，直接使用。注意，采用PriorityQueue的时候，每次插入删除的时间复杂度为O(logn). 而普通的队列，数组等插入的复杂度为O(1)，

## 归并排序Merge Sort