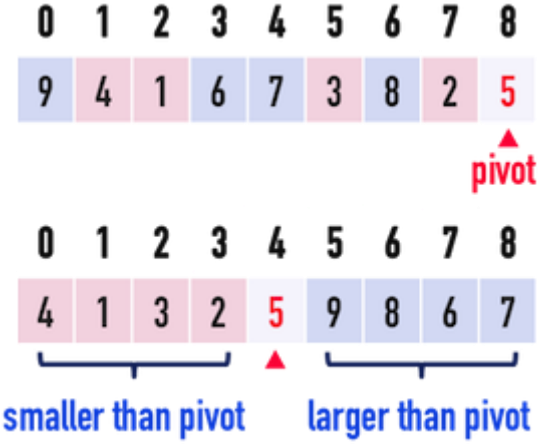
QuickSort Algorithm

Date：[2019.11.28]

一、算法原理

快速排序的基本思想是分为治之即'Divide and Conquer'，将一个大的问题缩减，我们随机挑选一个数作为枢 pivot，在两个游标 i,j 的作用下，不断的交换位置，使得数组左边的值均小于枢，而右边的值小于枢，从而将数组分为两个小的数组，如此递归下去直到分不出新的数列为止。



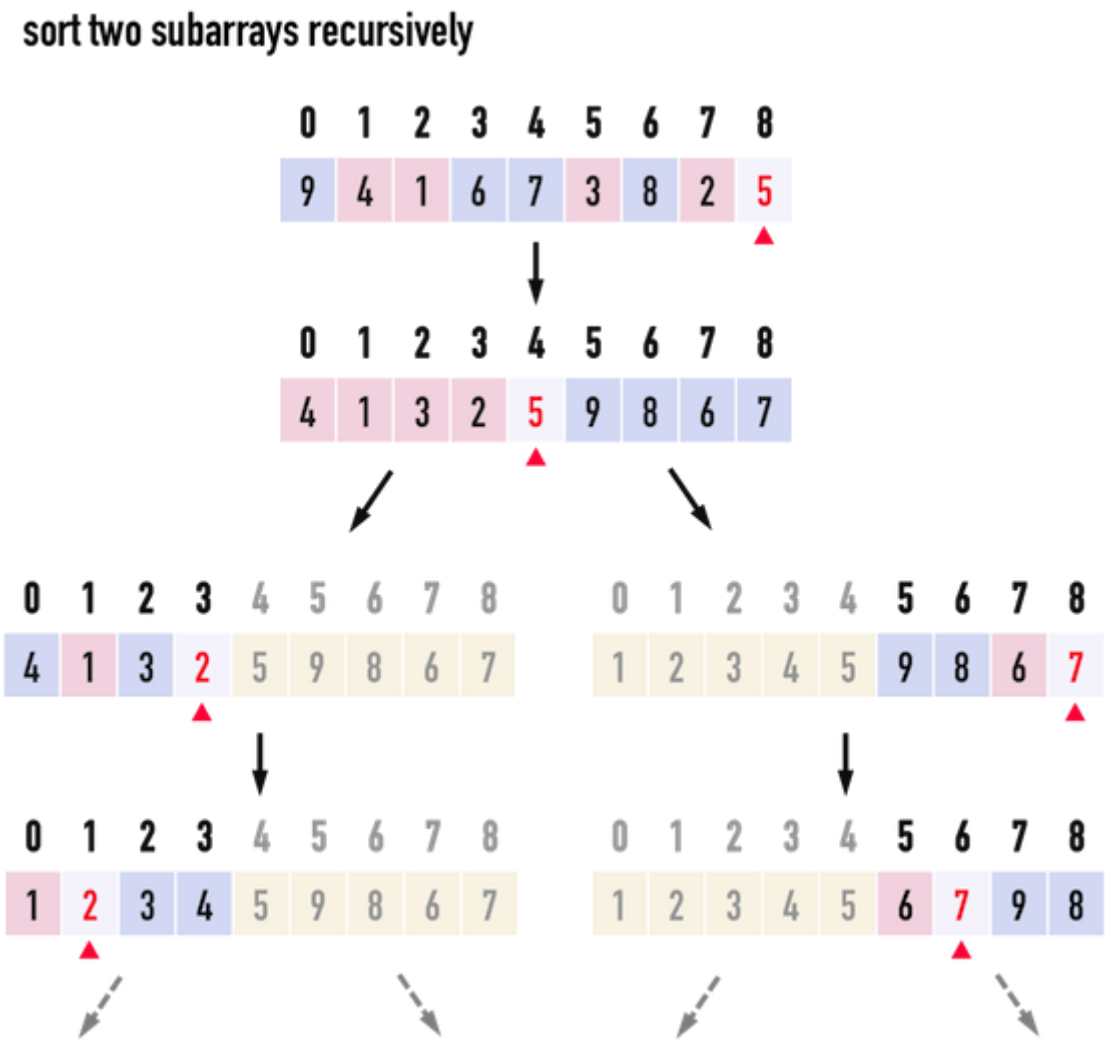
下面将要阐述其实现的过程，首先初始化我们选择 end 值指向的数为 pivot，当然我们可以随机的设定其他数组的元素，但是这样设置可以便于我们对 j 进行扫描。初始时我们设定 i=front-1,j=front，j 扫描 front 至 end-1 的所有数。

情形一：若扫描到的值大于 pivot，则不动，j++继续向下扫描；

情形二：若扫描到的值小于 pivot，则 i++，将 i 和 j 指向的值交换，确保小的值在左边，当然不要忘了 j 还要++，因为接下来还要继续扫描，如下图所示。

```
if array[ j ] < pivot
do i++
    swap( array[ i ], array[ j ] )
j++
```

经过以上的步骤，我们将数组分为比 pivot 小的数组，pivot，和比 pivot 大的数组，此后需要做的就是一个递归 recursive 的过程。



二、算法代码

```
#include<iostream>
void swap(int *a,int *b){
  int temp = *a;
  *a = *b;
  *b = temp;
}

int Partition(int *arr,int front,int end){
  int pivot = arr[end];
  int i = front-1;
  for (int j=front;j<end;j++){
    if(arr[j]<pivot){
      i++;
      swap(&arr[i],&arr[j]);
    }
  }
  i++;
  swap(&arr[i],&arr[end]);
  return i;
}

void QuickSort(int *arr,int front,int end){
  if(front<end){
    int pivot = Partition(arr,front,end);
```

```
25.       QuickSort(arr,front,pivot-1);
26.       QuickSort(arr,pivot+1,end);
27.     }
28.   }
29.
30.   void PrintArray(int *arr,int size){
31.     for(int i= 0;i<size;i++){
32.       std::cout<<arr[i]<<" ";
33.     }
34.     std::cout<<std::endl;
35.   }
36.
37.   int main(){
38.     int n = 9;
39.     int arr[] = {9,4,1,6,7,3,8,2,5};
40.     std::cout<<"original:\n";
41.     PrintArray(arr,n);
42.
43.     QuickSort(arr,0,n-1);
44.
45.     std::cout<<"sort:\n";
46.     PrintArray(arr,n);
47.     return 0;
48.   }
```

三、算法复杂度

	Quick Sort	Merge Sort	Heap Sort	Insertion Sort	Selection Sort
best case	$N \log N$	$N \log N$	$N \log N$	N	N^2
average case	$N \log N$	$N \log N$	$N \log N$	N^2	N^2
worst case	N^2	$N \log N$	$N \log N$	N^2	N^2

当划分产生的两个子问题分别包含 n-1 和 0 个元素时，最坏情况发生。划分操作的时间复杂度为 Θ (n)， T(0)=Θ(1)，这时算法运行时间的递归式为 T(n)=T(n−1)+T(0)+Θ(n)=T(n−1)+Θ(n)，解为 T(n)=Θ(n2)。

当划分产生的两个子问题分别包含⌊n/2⌋和⌈n/2⌉−1 个元素时，最好情况发生。算法运行时间递归式为 T(n)=2T(n/2)+Θ(n)，解为 T(n)=Θ(nlgn)。

四、参考文献

1.http://alrightchiu.github.io/SecondRound/comparison-sort-quick-sortkuai-su-pai-xu-fa.html
2. https://harttle.land/2015/09/27/quick-sort.html

Vector_In_Cpp

Date：[2019.11.28]

一、简介

向量（Vector）是一个封装了动态大小数组的顺序容器（Sequence Container）。跟任意其它类型容器一样，它能够存放各种类型的对象。可以简单的认为，向量是一个能够存放任意类型的动态数组。其特性是顺序序列、动态数组、能够感知内存分配器的 Allocator-aware。与 string 相同, vector 同属于 STL(Standard Template Library, 标准模板库)中的一种自定义的数据类型, 可以广义上认为是数组的增强版。

在使用它时, 需要包含头文件 vector, #include<vector>, vector 容器与数组相比其优点在于它能够根据需要随时自动调整自身的大小以便容下所要放入的元素。此外, vector 也提供了许多的方法来对自身进行操作。

二、基本操作

2.1 代码的声明及初始化

```
1.    vector<int> a ;                 //声明一个 int 型向量 a
2.    vector<int> a(10) ;             //声明一个初始大小为 10 的向量
3.    vector<int> a(10, 1) ;          //声明一个初始大小为 10 且初始值都为 1 的向量
```

2.2 简单的操作函数

```
1.    a.push_back(1);              //向量尾部增加一个元素 1
2.    a.insert(position,1);        //在 position 位置插入元素 1
3.    a.size()                     //获取向量中的元素个数
4.    a.empty()                    //判断向量是否为空
5.    a.clear()                    //清空向量中的元素
6.    a.erase(position) ;          //将 position 位置的元素删除
```

2.3 迭代器使用

```
1.    vector<int>::iterator t ;
2.    for(t=a.begin(); t!=a.end(); t++)
3.        cout<<*t<<" " ;
```

2.4 二维数组的定义

```
1.    vector<vector<int> > obj(N, vector<int>(M)); //定义二维动态数组 N 行 M 列
```

2.5 注意

Vector 作为函数的参数或者返回值时，需要注意它的写法：double Distance(vector<int>&a, vector<int>&b) 其中的“&”绝对不能少!!!

三、参考博客：

1. https://www.runoob.com/w3cnote/cpp-vector-container-analysis.html
2. https://www.cnblogs.com/mr-wid/archive/2013/01/22/2871105.html
3. https://blog.csdn.net/duan19920101/article/details/50617190