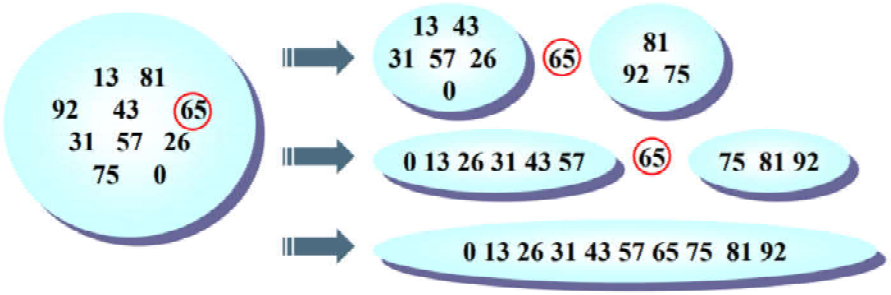
# 快速排序

2019年6月30日 20:22

- 分而治之




什么是快速排序算法的最好情况？

每次正好中分 ⟶ $T(N) = O(N\log N)$

- 分而治之

```
void Quicksort( ElementType A[], int N )
{
 ?pivot = 从A[]中选一个主元;
 ?将S = { A[] \ pivot } 分成2个独立子集:
      A1={ a∈S | a ≤ pivot } 和
      A2={ a∈S | a ≥ pivot };
   A[] = Quicksort(A1,N1)∪
                 {pivot}∪
         Quicksort(A2,N2);
}
```

选主元

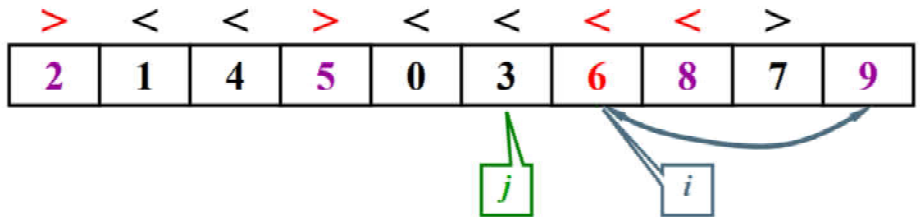
随机取 pivot？ rand()函数不便宜啊！

· 取头、中、尾的中位数 · 例如 8、12、3的中位数就是8 · 测试一下pivot不同的取法对运行速度有多大影响？

```
ElementType Median3( ElementType A[], int Left, int Right )
{   int  Center = ( Left + Right ) / 2;
    if ( A[ Left ] > A[ Center ] )
        Swap( &A[ Left ], &A[ Center ] );
    if ( A[ Left ] > A[ Right ] )
        Swap( &A[ Left ], &A[ Right ] );
    if ( A[ Center ] > A[ Right ] )
        Swap( &A[ Center ], &A[ Right ] );
    /* A[ Left ] <= A[ Center ] <= A[ Right ] */
    Swap( &A[ Center ], &A[ Right-1 ] ); /* 将pivot藏到右边 */
    /* 只需要考虑 A[ Left+1 ] … A[ Right-2 ] */
    return  A[ Right-1 ];  /* 返回 pivot */
}
```

子集划分

· 如果有元素正好等于pivot怎么办？ · 停下来交换




i<pivot ,j>pivot

快速排序的问题

· 用递归……

· 对小规模的数据（例如 N不到100）可能还不如插 入排序快 ·

解决方案

· 当递归的数据规模充分小，则停止递归，直接调用 简单排序（例如插入排序）

· 在程序中定义一个Cutoff的阈值 —— 课后去实践 一下，比较不同的Cutoff对效率的影响

代码：

```
#include<iostream>
#define ElementType int

using namespace std;
void Swap(int *x, int *y)
{
        int temp = *y;
        *y = *x;
        *x = *y;
}


ElementType Median3(ElementType A[], int Left, int Right)
{
        int Center = (Left + Right) / 2;
        if (A[Left] > A[Center])
                Swap(&A[Left], &A[Center]);
        if (A[Left] > A[Right])
                Swap(&A[Left], &A[Right]);
        if (A[Center] > A[Right])
                Swap(&A[Center], &A[Right]);
        /* 此时A[Left] <= A[Center] <= A[Right] */
        Swap(&A[Center], &A[Right - 1]); /* 将基准Pivot藏到右边*/
        /* 只需要考虑A[Left+1] … A[Right-2] */
        return  A[Right - 1];  /* 返回基准Pivot */
}

void InsertionSort(ElementType A[], int N)
{ /* 插入排序 */
        int P, i;
        ElementType Tmp;
```

```
	for (P = 1; P < N; P++) {
		Tmp = A[P]; /* 取出未排序序列中的第一个元素*/
		for (i = P; i > 0 && A[i - 1] > Tmp; i--)
			A[i] = A[i - 1]; /*依次与已排序序列中元素比较并右移*/
		A[i] = Tmp; /* 放进合适的位置 */
	}
}

void Qsort(ElementType A[], int Left, int Right)
{ /* 核心递归函数 */
	int Pivot, Cutoff, Low, High;
	Cutoff = 100;//阈值

	if (Cutoff <= Right - Left) { /* 如果序列元素充分多，进入快排 */
		Pivot = Median3(A, Left, Right); /* 选基准 */
		Low = Left; High = Right - 1;
		while (1) { /*将序列中比基准小的移到基准左边，大的移到右边*/
			while (A[++Low] < Pivot);
			while (A[--High] > Pivot);
			if (Low < High) Swap(&A[Low], &A[High]);
			else break;
		}
		Swap(&A[Low], &A[Right - 1]);  /* 将基准换到正确的位置 */
		Qsort(A, Left, Low - 1);  /* 递归解决左边 */
		Qsort(A, Low + 1, Right);  /* 递归解决右边 */
	}
	else InsertionSort(A + Left, Right - Left + 1); /* 元素太少，用简单排序 */
}

void QuickSort(ElementType A[], int N)
{ /* 统一接口 */
	Qsort(A, 0, N - 1);
}

int main()
{
	int n;
	cin >> n;
	int* a = new int[n];
	for (int i = 0; i < n; i++)
	{
		cin >> a[i];
	}
	QuickSort(a, n);

	for (int i = 0; i < n; i++)
	{
		cout << a[i]<<" ";
	}
	delete[] a;
	return 0;
}
```

直接调用库函数qsort（）

头文件stdlib.h

qsort(*s, n, sizeof(s[0]), cmp);

其中第一个参数s是一个地址，即参与排序的首地址；
n是需要排序的数量；
sizeof(s[0])则是每一个元素占用的空间大小；
指向函数的指针，用于确定排序的顺序。

//其中cmp函数应写为：

```
int cmp(const void *a, const void *b)
{
   return *(int*)a - *(int*)b; //由小到大排序
   //return *(int *)b - *(int *)a; 由大到小排序
}
```

例子:

```
#include <stdio.h>
#include <stdlib.h>

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

int main()
{
  int n;

printf("排序之前的列表：\n");
  for( n = 0 ; n < 5; n++ ) {
    printf("%d ", values[n]);
  }

qsort(values, 5, sizeof(int), cmpfunc);

printf("\n排序之后的列表：\n");
  for( n = 0 ; n < 5; n++ ) {
    printf("%d ", values[n]);
  }

 return(0);
}
```