5. 对下面数据表，写出采用快速排序算法排序的每趟的结果，并标明每趟的数据移动情况。

(50 30 120 25 85 40 100 12 90 15 60 35 105 78 10 28)

表格

描述已自动生成

7.已知数组A[n]中的元素为整型，设计算法将其调整为三部分，其中左边所有元素为3的倍数，中间所有元素除3余1，右边所有元素除3余2，并要求时间复杂度为O(n)。

遍历数组A，将3的倍数交换到数组到右边，再次遍历数组，将除三取余为2的交换到左边。

Int count=0;

for(int i=0; i<length; i++)

{

if(arr[i]%3==0)

{

count++;

}

else

{

int temp = arr[i-count];

arr[i-count] = arr[i];

arr[i] = temp;

}

}

count=0;

for(int i= length-1; i>0; i--)

{

if(arr[i]%3==2)

{

count++;

}

else

{

int temp = arr[i+count];

arr[i+count] = arr[i];

arr[i] = temp;

}

}

11. 由初始建堆过程的讨论可知，调整过程可以是递归形式的，请写出这一递归形式的算法。

按照完全二叉树，将数字依次填入。填入完成后，**从最后一个非叶子结点开始调整。**小的数字往上移动；有子节点的情况，需要递归进行调整。

void heapsort(vector<int>& v, int pos, int end, int flag)

{

if (pos >= end) return;

// flag = 0表示第一次初始建堆，从底层递归判断

// 第二次以后由于只是交换了首尾节点，因此子节点是满足最大堆特性，这一步就不再需要了

if (!flag)

{

heapsort(v, pos \* 2, end, flag);

heapsort(v, pos \* 2 + 1, end, flag);

}

// 判断左右子树，将较大值上移

if (pos \* 2 + 1 < end)

{

// 与子节点中较大值交换

if (v[pos] < max(v[pos \* 2], v[pos \* 2 + 1]))

{

if (v[pos \* 2 + 1] > v[pos \* 2])

{

swap(v, pos, pos \* 2 + 1);

// 交换后，可能子节点不满足约束，需要继续递归

heapsort(v, pos \* 2 + 1, end, flag);

}

else

{

swap(v, pos, pos \* 2);

heapsort(v, pos \* 2, end, flag);

}

}

}

else if (pos \* 2 < end && v[pos] < v[pos \* 2])

{

swap(v, pos, pos \* 2);

heapsort(v, pos \* 2, end, flag);

}

}

12. 对长度为10000的数据表，如果在不用完整排序的情况下，要求找出其中最大的10个数，应选用何种排序算法最节省时间？

堆排序，时间复杂度O（log2k），k->10000，复杂度低