排序数组

给你一个整数数组 nums，请你将该数组升序排列。

示例 1：

输入：nums = [5,2,3,1]

输出：[1,2,3,5]

示例 2：

输入：nums = [5,1,1,2,0,0]

输出：[0,0,1,1,2,5]

解题思路：

比较多种排序算法，最终只有堆排序，快排，归并排序符合时间复杂度要求

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
 //直接插入排序
 /*
int* sortArray(int* nums, int numsSize, int* returnSize){
    int temp;
    int end;
    int i;
    for(i=1;i<numsSize;i++)
    {
        temp=nums[i];
        end=i-1;
        while(end>=0&&nums[end]>temp)
        {
            nums[end+1]=nums[end];
            end--;
        }
        nums[end+1]=temp;
    }
    *returnSize=i;
    return nums;
}

 //希尔排序优化  直接插入排序
int* sortArray(int* nums, int numsSize, int* returnSize){
    int step=numsSize/2;
    int temp;
```

```
    int end;
    int i;
    while(step>=1)
    {
        for(i=step;i<numsSize;i++)
        {
            temp=nums[i];
            end=i-step;
            while(end>=0&&nums[end]>temp)
            {
                nums[end+step]=nums[end];
                end-=step;
            }
            nums[end+step]=temp;
        }
        step--;
    }
    *returnSize=numsSize;
    return nums;
}
//选择排序 大的尾部  小的头部
int* sortArray(int* nums, int numsSize, int* returnSize)
{
    int start=0;
    int end=numsSize-1;
    while(start<end)
    {
        int minidx=start;
        int maxidx=start;
        for(int i=start+1;i<=end;i++)
        {
            if(nums[i]>nums[maxidx])
                maxidx=i;
            if(nums[i]<nums[minidx])
                minidx=i;
        }
        int tmp=nums[start];
        nums[start]=nums[minidx];
        nums[minidx]=tmp;

        if(maxidx==start)
            maxidx=minidx;

        int temp=nums[end];
```

```
        nums[end]=nums[maxidx];
        nums[maxidx]=temp;

        start++;
        end--;
    }
    *returnSize=numsSize;
    return nums;
}
void shiftdown(int* arr, int n, int curpos)
{
    int child = 2 * curpos + 1;
    while (child < n)
    {
        if (child + 1 < n&&arr[child + 1] > arr[child])//找孩子中最大值
            child++;
        if (arr[child] > arr[curpos])
        {
            //大的值向上，当前值向下调整
            int tmp = arr[curpos];
            arr[curpos] = arr[child];
            arr[child] = tmp;

            //更新当前位置
            curpos = child;
            child = 2 * curpos + 1;
        }
        else
            break;
    }
}

int* sortArray(int* nums, int numsSize, int* returnSize)
{
    //建大堆
    for (int i = (numsSize - 2) / 2; i >=0; i--)
    {
        shiftdown(nums,numsSize,i);
    }
    //堆中最后一个元素
    int end = numsSize - 1;
    while (end > 0)
    {
        //交换堆顶和最后一个元素
```

```
        int temp = nums[0];
        nums[0] = nums[end];
        nums[end] = temp;

        //除去最后一个元素 继续维护大堆
        shiftdown(nums,end,0);
        end--;

    }
    *returnSize=numsSize;
    return nums;
}
int* sortArray(int* nums, int numsSize, int* returnSize)
{
    for(int i=0;i<numsSize;i++)
    {
        int flag=1;
        for(int j=0;j<numsSize-i-1;j++)
        {
            if(nums[j]>nums[j+1])
            {
                int tmp=nums[j];
                nums[j]=nums[j+1];
                nums[j+1]=tmp;
                flag=0;
            }
        }
        if(flag==1)
            break;
    }
    *returnSize=numsSize;
    return nums;
}

//: 三数取中 -> 划分均衡
int getmid(int* arr, int begin, int end)
{
    int mid = (end - begin) / 2;
    if (arr[begin] < arr[end])
    {
        if (arr[begin] > arr[mid])
            return begin;
        else if (arr[mid] > arr[end])
            return end;
```

```
        else
            return mid;
    }
    else
    {
        if (arr[end] > arr[mid])
            return end;
        else if (arr[mid] > arr[begin])
            return begin;
        else
            return mid;
    }
}
int paration(int* arr, int begin, int end)
{
    int key = arr[begin];
    int start = begin;
    while (begin < end)
    {
        //从后往前找到第一个小于基准值的数据
        while (begin < end&&arr[end] >= key)
        {
            end--;
        }
        //从前往后找到第一个大于基准值的数据
        while (begin < end&&arr[begin] <= key)
        {
            begin++;
        }
        //交换两个位置的值
        int tmp = arr[end];
        arr[end] = arr[begin];
        arr[begin] = tmp;
    }
    //从相遇位置的数据和基准值进行交换
    int temp = arr[begin];
    arr[begin] = arr[start];
    arr[start] = temp;
    return begin;//返回划分位置  保证一边大一边小
}
//递归法
void quicksort(int* arr,int begin,int end)
{
    if (begin >= end)
```

```
        return;
    int div = paration(arr, begin, end);
    quicksort(arr,begin,div-1);
    quicksort(arr, div + 1, end);
}
int* sortArray(int* nums, int numsSize, int* returnSize)
{
    int begin=0;
    int end=numsSize-1;
    quicksort(nums,begin,end);
    *returnSize=numsSize;
    return nums;
}*/

//合并前提 两个数组有序
void merge(int* arr,int left,int right)
{
    int mid = left+(right - left) / 2;
    int len = right - left + 1;
    //创建临时数组
    int* temp = (int*)malloc(sizeof(int)*len);

    int k = 0;      //临时数组下标
    int p = left;   //左数组首元素下标
    int q = mid + 1;//右数组首元素下标
    //先有一半存入临时数组
    while (p <= mid && q <= right)
    {
        if (arr[p] < arr[q])
        {
            temp[k++] = arr[p++];//谁小谁先放入临时数组中
        }
        else
        {
            temp[k++] = arr[q++];
        }
    }
    //剩余元素排最后
    while (p <= mid)
    {
        temp[k++] = arr[p++];
    }
    while (q <= right)
    {
```

```
        temp[k++] = arr[q++];
    }
    //拷贝回原数组
    memcpy(arr+left,temp,sizeof(int)*len);
    free(temp);
    temp = NULL;
}
//针对无序数组  先拆分直至有序  然后依次合并
void Mergesort(int* arr, int l, int r)
{
    if (l == r) //区间只有一个元素,无需排序
    {
        return;
    }
    int m = l + (r - l) / 2;
    Mergesort(arr, l, m);  //拆分左数组
    Mergesort(arr, m+1, r); //拆分右数组
    merge(arr, l, r);//合并
}
int* sortArray(int* nums, int numsSize, int* returnSize)
{
    int begin=0;
    int end=numsSize-1;
    Mergesort(nums,begin,end);
    *returnSize=numsSize;
    return nums;
}
```