```
/*
给定由一些正数（代表长度）组成的数组 A，返回由其中三个长度组成的、面积不为零的三角形的最大周长。
如果不能形成任何面积不为零的三角形，返回 0。
示例 1：

输入：[2,1,2]
输出：5

示例 2：

输入：[1,2,1]
输出：0

示例 3：

输入：[3,2,3,4]
输出：10
示例 4：
输入：[3,6,2,3]
输出：8
提示：
    3 <= A.length <= 10000
    1 <= A[i] <= 10^6
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/largest-perimeter-triangle
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 首先将原数组降序排序，再一次检测相邻三个是否能组成三角形，能组成的话则直接返回这个边的和.
- 方法一：选择排序($O(N^2)$)+依次遍历($O(N)$),提交超时了
- 方法二：规避排序($O(N\log N)$)+依次遍历($O(N)$)，提交低分通过

方法一:C_先排序(选择排序)再依次遍历

```
void __swap(int*a ,int*b)
{
    int c = *a;
    *a = *b;
    *b = c;
}

bool test(int a,int b,int c)
{

    if(  (a<=0)
       ||(b<=0)
       ||(c<=0)
```

```
    )
    {
        return false;
    }
    else
    {
        if(a<(b+c))
        {
            return true;
        }
    }
    return false;
}

void selectionSort( int* arr ,int n )
{

    int maxIndex = 0;
    for( int i = 0 ; i < n ; i++)
    {
        // 寻找 [i,n)区间内的最大值
        maxIndex = i;
        for(int j = i+1;j<n;j++)
        {
            if(arr[j] > arr[maxIndex]) //
            {
                maxIndex = j;
            }
            // C++ 内置, 视C++标准支持情况,std 中就有或者 #include <algorithm>

        }
        __swap(&arr[i],&arr[maxIndex]);
    }
}

int largestPerimeter(int* A, int ASize)
{
    int ret_val = 0;
    int a       = 0;
    int b       = 0;
    int c       = 0;
    int i       = 0;
    /* 排序*/
    selectionSort(A,ASize);
    for(i=0;i<ASize-2;i++)
    {
        a = A[i];
        b = A[i+1];
        c = A[i+2];
        if(test(a,b,c))
        {
            ret_val = a + b + c;
            break;
        }
    }
    return ret_val;
}
/*
```

```
选择排序超时了
*/
```

方法二:C_先排序(归并排序)再依次遍历

```
/*
    归并过程,对arr[left,right]的范围进行归并
*/
void __merge( int* arr, int left, int mid,int right)
{
    int     i   =   0                                               ;
    int     j   =   0                                               ;
    int     k   =   0                                               ;
    int*    aux =   (int*)malloc(sizeof(int)*(right-left+1))    ;

    /* 先拷贝出来 */
    for(i = left ; i<=right;i++)
    {
        aux[i-left] = arr[i];
    }

    /* 进行有序数组合成 */
    i = left;
    j = mid + 1;
    for(k=left;k<=right;k++)    /* 要存入位置的索引*/
    {
        if(i>mid)   /* 索引合法性*/
        {
            arr[k] = aux[j-left];
            j++;
        }
        else if(j > right) /* 索引合法性*/
        {
            arr[k] = aux[i-left];
            i++;
        }
        else if( aux[i-left] < aux[j-left]) /* 索引已经合法 */
        {
            arr[k] = aux[i-left];
            i++;
        }
        else
        {
            arr[k] = aux[j-left];
            j++;
        }
    }

    free(aux);
    return ;
}


/*
    递归使用归并排序,对arr[left,right]的范围进行排序
```

```
*/
void __mergeSort( int* arr, int left,int right) // 表征私有,不应该被外部调用,但可以被外部调用
{
    /* 递归到顶*/
    if(left>=right)
    {
        return ;
    }

    int mid = left + (right - left)/2;  // 防止溢出
    __mergeSort(arr,left,mid);
    __mergeSort(arr,mid+1,right);

    /* 优化添加,针对已经有序的不需要重新归并排序*/
    if(arr[mid]<=arr[mid+1])
    {
        return ;
    }
    else
    {
        __merge(arr,left,mid,right); /* 两个有序数组归并过程*/
    }
}

bool test(int a,int b,int c)
{
    if(  (a<=0)
       ||(b<=0)
       ||(c<=0)
    )
    {
        return false;
    }
    else
    {
        if(a<(b+c))
        {
            return true;
        }
    }
    return false;
}


int largestPerimeter(int* A, int ASize)
{
    int ret_val = 0;
    int a       = 0;
    int b       = 0;
    int c       = 0;
    int i       = 0;

    /* 排序*/
    __mergeSort(A,0,ASize-1);

    /*逐一判决*/
    for(i=ASize-1;i>=2;i--)
```

```
    {
        a = A[i];
        b = A[i-1];
        c = A[i-2];
        if(test(a,b,c))
        {
            ret_val = a + b + c;
            break;
        }
    }
    return ret_val;
}

/*
执行结果:
通过
显示详情
执行用时 :164 ms, 在所有 C 提交中击败了10.61% 的用户
内存消耗 :32.1 MB, 在所有 C 提交中击败了5.26%的用户
*/
```

AlimyBreak
2019.08.15