**22.**

1)

template<class T>

void arrayList<T>::reverse() {

/\*

\* 若偶数长度，则 listSize/2 刚好为长度一半；

\* 若奇数长度，则 listSize/2 的机器值是数学值减 0.5，

\* 在 swap 时刚好可以将数组中位数独立出来

\*/

for (int i = 0; i < listSize / 2; i++) {

swap(element[i],element[listSize-i-1]);

}

}

2)

其时间复杂度为O(n/2)，因为该方法中只有一个for循环，循环n/2此，循环内swap复杂度为O(1)，故整体的复杂度为O(n/2)。

4)

template<class T>

void reverse(arrayList<T>& a) {

/\*使用迭代器对元素进行操作\*/

TAiterator<int> left = a.begin();

TAiterator<int> right = a.end();

/\*该处i的取值逻辑类似成员函数reverse，a.size()/2刚好获得所需循环次数\*/

for (int i = 0; i < a.size()/2; i++) {

swap(\*left, \*right);

--right;

left++;

}

}

5)

时间复杂度同上，都是O(n/2)。

**29.**

1)

template<class T>

void arrayList<T>::merge(const arrayList<T>& a, const arrayList<T>& b){

/\*重新建立数组\*/

delete[] element;

listSize = a.listSize + b.listSize;

element = new T[listSize];

/\*进行边比对边复制的过程，记录下标变化，以便后续扫尾工作\*/

int a\_index = 0;

int b\_index = 0;

int index = 0;

while ((a\_index < a.listSize) && (b\_index < b.listSize)) {

if (a.element[a\_index] <= b.element[b\_index]) {

element[index++] = a.element[a\_index++];

} else {

element[index++] = b.element[b\_index++];

}

}

//如果index等于listsize，说明该数组移动完毕；否则未复制完毕，继续复制剩余元素。

//a，b线性表必只剩一个没有复制完毕，因此此处两个判断只有一个能执行

if (a\_index < a.listSize) {

while ( a\_index < a.listSize ) {

element[index++] = a.element[a\_index++];

}

}

if (b\_index < b.listSize) {

while (b\_index < b.listSize) {

element[index++] = b.element[b\_index++];

}

}

}

2)

其时间复杂度为O(a.listSize + b.listSize)