

2. 向量

(f) 归并排序

I think there is a world market
for about five computers.
- T. J. Watson, 1943

天下大势，分久必合，合久必分

邓俊辉

deng@tsinghua.edu.cn

归并排序：原理

❖ //分治策略

//向量与列表通用

//J. von Neumann, 1945

序列一分为二 //0(1)

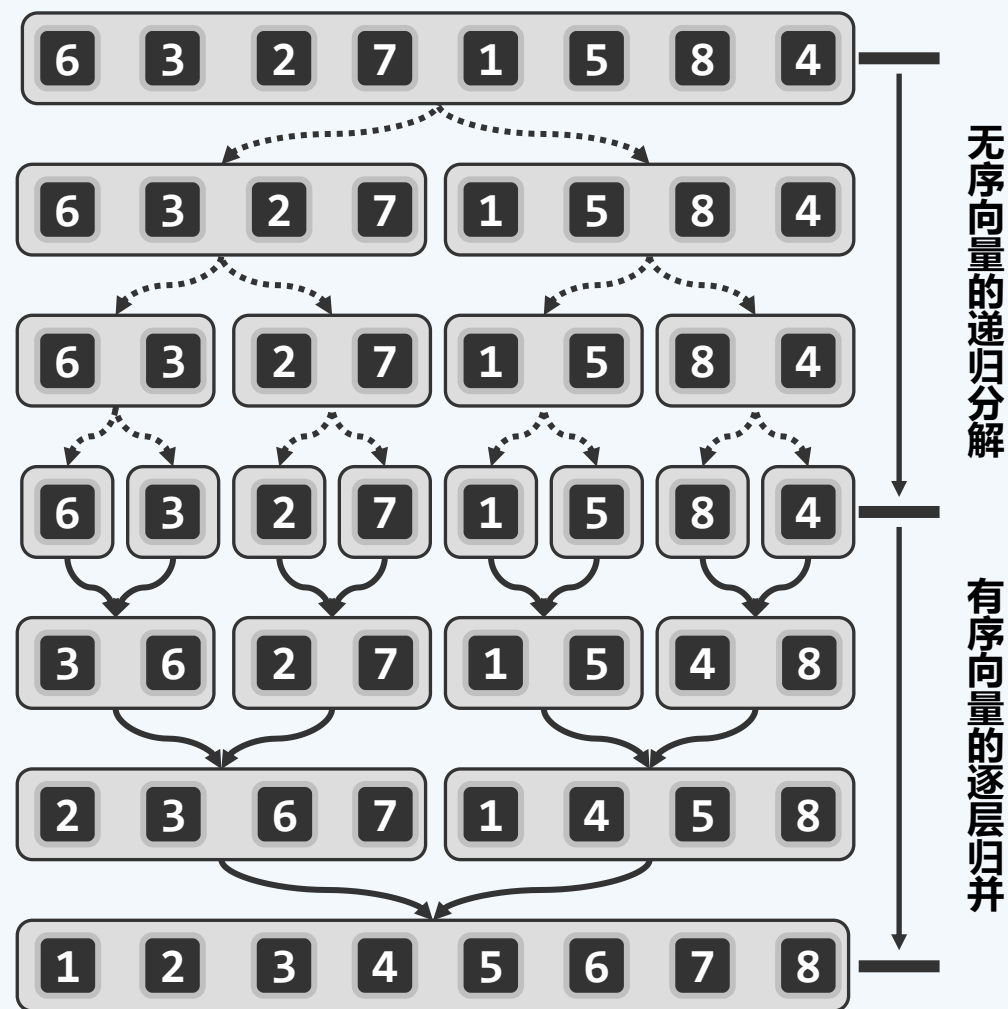
子序列递归排序 //2 × T(n/2)

合并有序子序列 //0(n)

❖ 例1：mergesort

例2：xSortLab

❖ 若真能如此，归并排序的运行时间应该是 $O(n \log n)$



归并排序：实现

❖ template <typename T>

```
void Vector<T>::mergeSort(Rank lo, Rank hi) { //[lo, hi)
```

```
    if (hi - lo < 2) return; //单元素区间自然有序，否则...
```

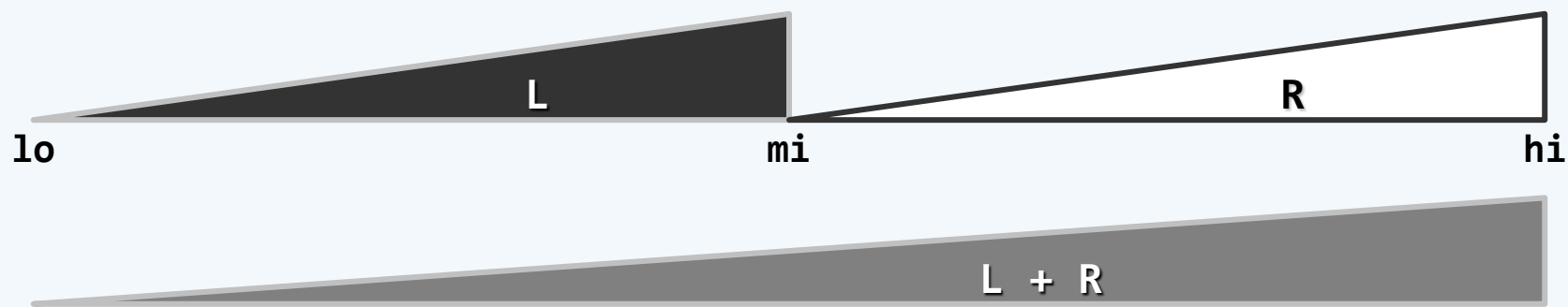
```
    int mi = (lo + hi) >> 1; //以中点为界
```

```
    mergeSort(lo, mi); //对前半段排序
```

```
    mergeSort(mi, hi); //对后半段排序
```

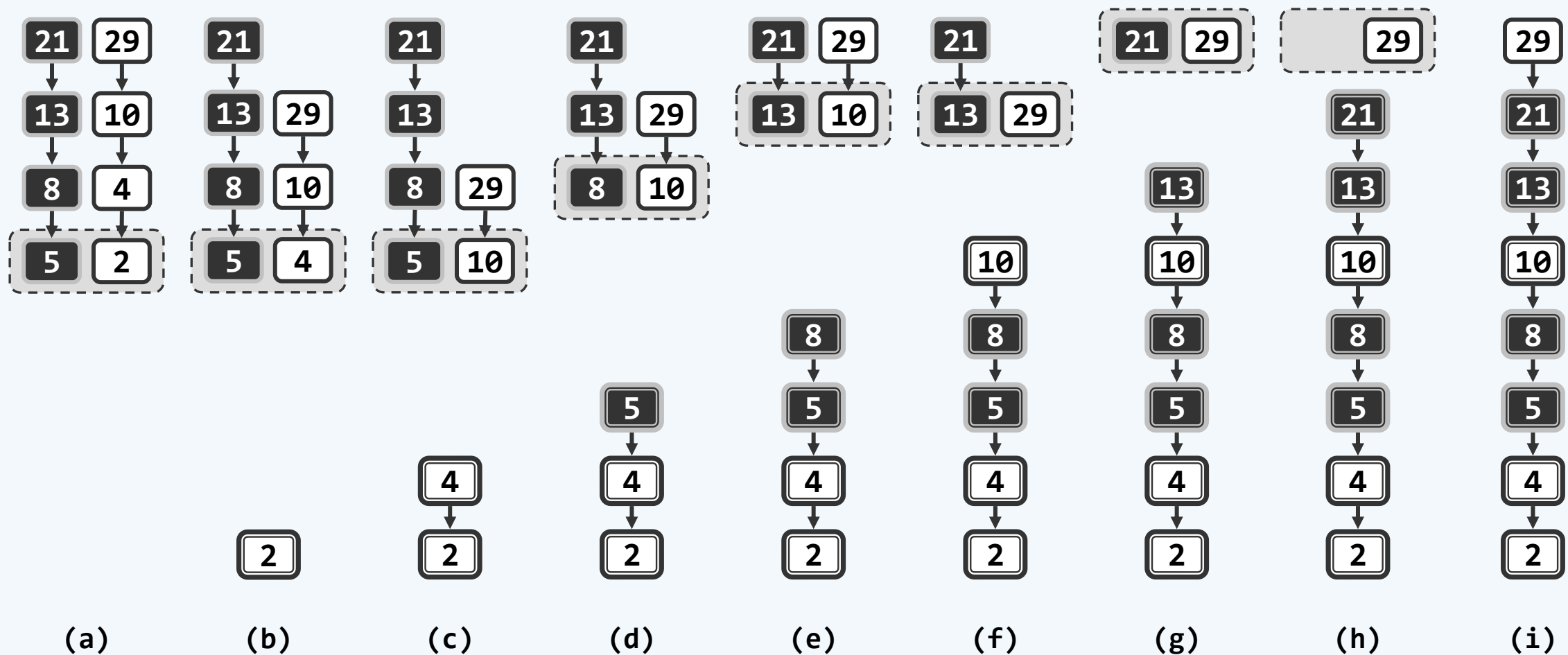
```
    merge(lo, mi, hi); //归并
```

```
}
```



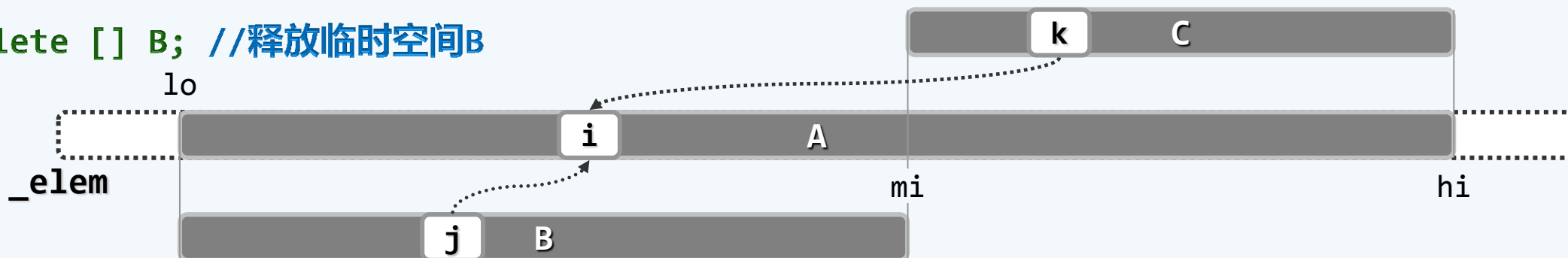
二路归并：原理

❖ 2-way merge：将两个有序序列合并为一个有序序列： $S[lo, hi) = S[lo, mi) + S[mi, hi)$

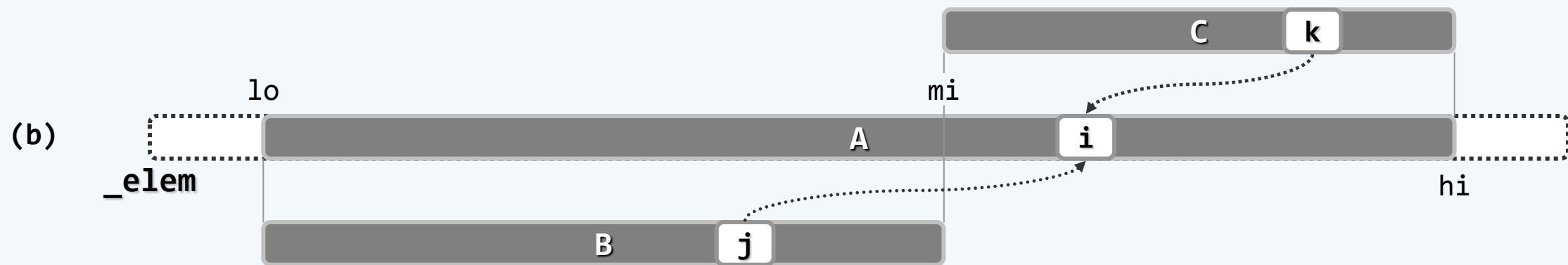
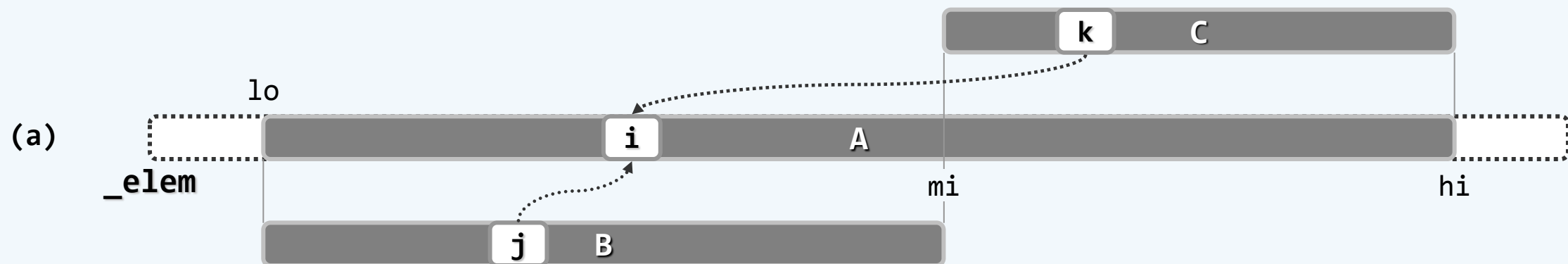


二路归并：基本实现

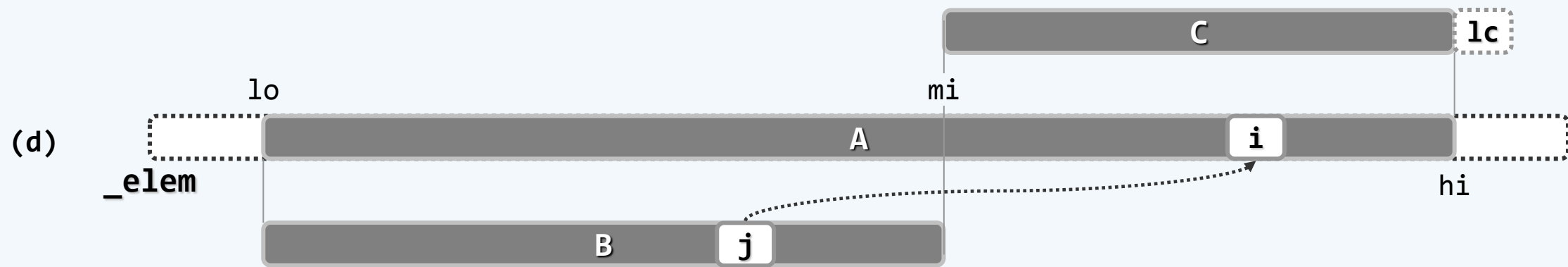
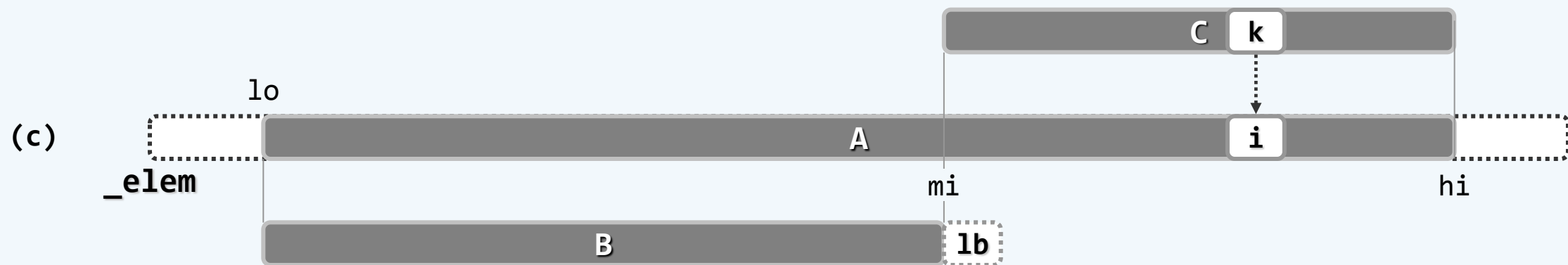
```
template <typename T> void Vector<T>::merge(Rank lo, Rank mi, Rank hi) {  
    T* A = _elem + lo; //合并后的向量A[0, hi - lo) = _elem[lo, hi)  
    int lb = mi - lo; T* B = new T[lb]; //前子向量B[0, lb) = _elem[lo, mi)  
    for (Rank i = 0; i < lb; B[i] = A[i++]); //复制前子向量B  
    int lc = hi - mi; T* C = _elem + mi; //后子向量C[0, lc) = _elem[mi, hi)  
    for (Rank i = 0, j = 0, k = 0; (j < lb) || (k < lc); ) { //B[j]和C[k]中小者转至A的末尾  
        if ( (j < lb) && (lc <= k || (B[j] <= C[k])) ) A[i++] = B[j++]; //C[k]已无或不小  
        if ( (k < lc) && (lb <= j || (C[k] < B[j])) ) A[i++] = C[k++]; //B[j]已无或更大  
    } //该循环实现紧凑；但就效率而言，不如拆分处理  
    delete [] B; //释放临时空间B  
}
```



二路归并：正确性

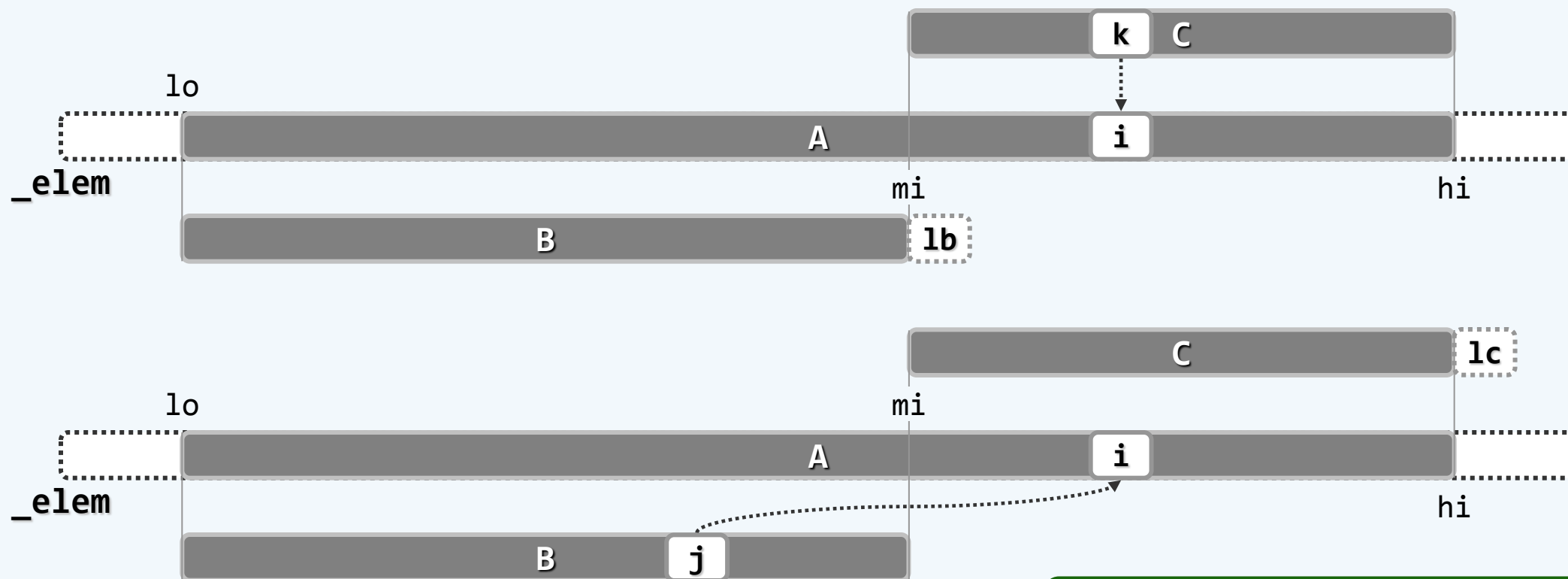


二路归并：正确性



二路归并：精简实现

❖ for (Rank i = 0, j = 0, k = 0; (j < lb) || (k < lc);) {
 if ((k < lc) && (lb <= j || (C[k] < B[j]))) A[i++] = C[k++];
 if ((j < lb) && (lc <= k || (B[j] <= C[k]))) A[i++] = B[j++];
} //交换循环体内两句的次序，删除冗余逻辑



二路归并：复杂度

❖ 算法的运行时间主要消耗于for循环，共有两个控制变量

初始： $j = 0, k = 0$

最终： $j = lb, k = lc$

亦即： $j + k = lb + lc = hi - lo = n$

❖ 观察：每经过一次迭代， j 和 k 中至少有一个会加一（ $j + k$ 也必至少加一）

❖ 故知：merge()总体迭代不过 $O(n)$ 次，累计只需线性时间！

❖ 这一结论与排序算法的 $\Omega(n \log n)$ 下界并不矛盾——毕竟这里的B和C均已各自有序

❖ 注意：待归并子序列不必等长

亦即：即允许 $lb \neq lc, mi \neq (lo + hi)/2$

❖ 实际上，这一算法及结论也适用于另一类序列——列表（下一章）

综合评价

❖ 优点

实现最坏情况下最优 $O(n\log n)$ 性能的第一个排序算法

不需随机读写，完全顺序访问——尤其适用于

列表之类的序列

磁带之类的设备

只要实现恰当，可保证稳定——出现雷同元素时，左侧子向量优先

可扩展性极佳，十分适宜于外部排序——海量网页搜索结果的归并

易于并行化

❖ 缺点

非就地，需要对等规模的辅助空间——可否更加节省？

即便输入完全（或接近）有序，仍需 $\Theta(n\log n)$ 时间——改进...