计算机问题求解 – 论题1-14&15

- 算法的效率&问题的难度

课程研讨

- DH第6-7章

问题1：算法的效率

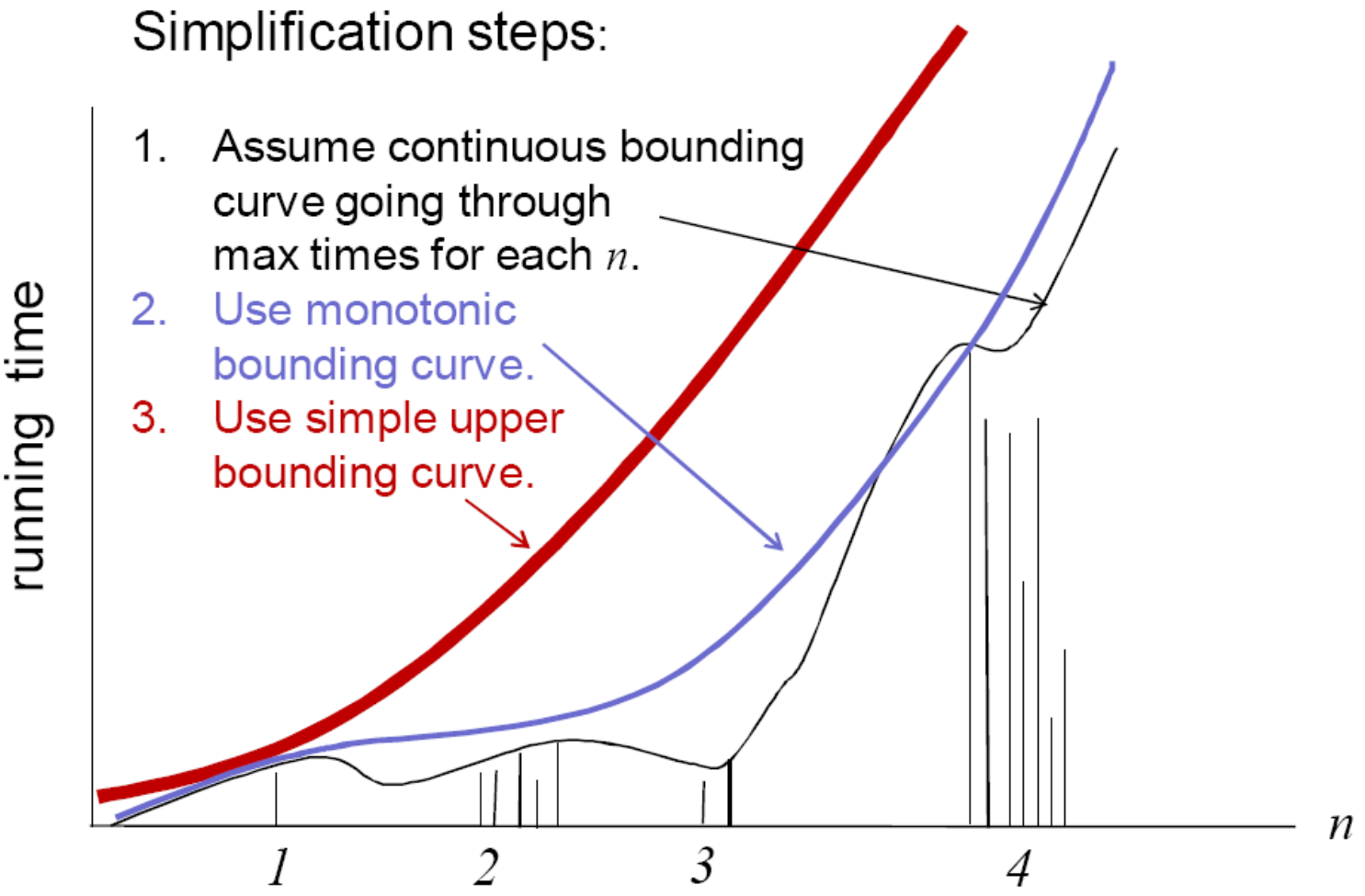
- linear search的时间复杂度是O(N)
  binary search的时间复杂度是O(logN)
  请综合你对DH第129-139页的理解，谈谈你是如何理解这两句话的

问题1：算法的效率

- linear search的时间复杂度是O(N)
  binary search的时间复杂度是O(logN)
  请综合你对DH第129-139页的理解，谈谈你是如何理解这两句话的

- 关键点
  - 对于不同的输入，时间相同吗？
  - 计时的单位是什么？
  - big-O是什么意思？

哪一条线表示big-O?


Simplification steps:
1. Assume continuous bounding curve going through max times for each n.
2. Use monotonic bounding curve.
3. Use simple upper bounding curve.
running time
n
1
2
3
4

问题1：算法的效率 (续)

- 想想看，有没有时间复杂度为O(1)的search？

问题1：算法的效率 (续)

- 如何理解big-O的鲁棒性？
  （分别有什么优缺点？）

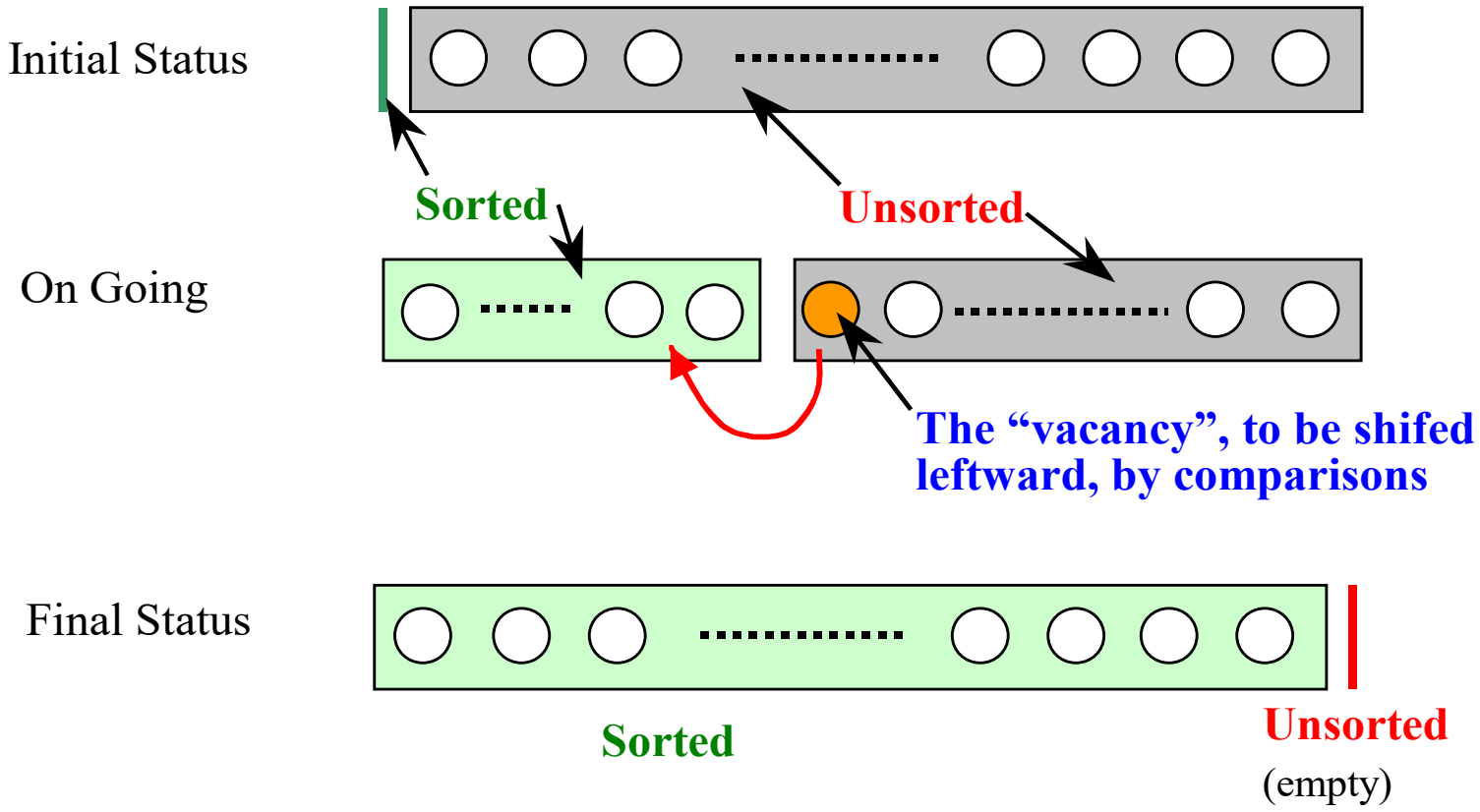
问题1：算法的效率 (续)

- 你会分析insertion sort的时间复杂度吗？

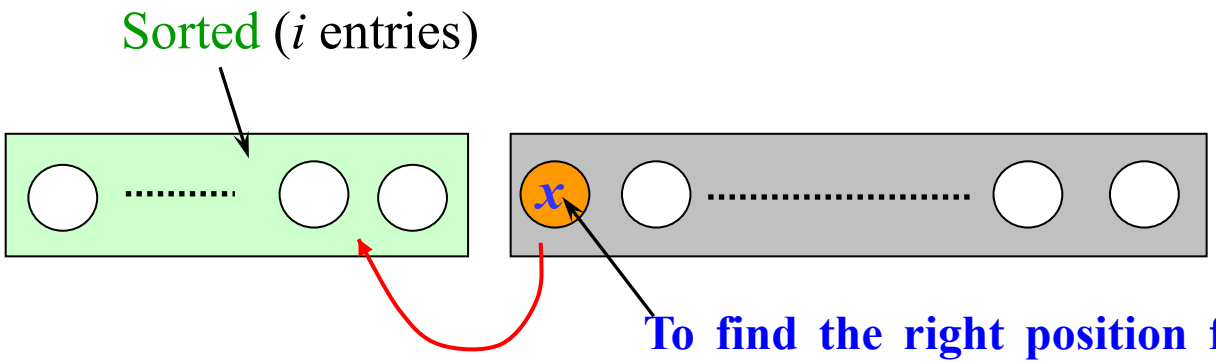
6 5 3 1 8 7 2 4

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

As Simple as Inserting



Worst-Case Analysis

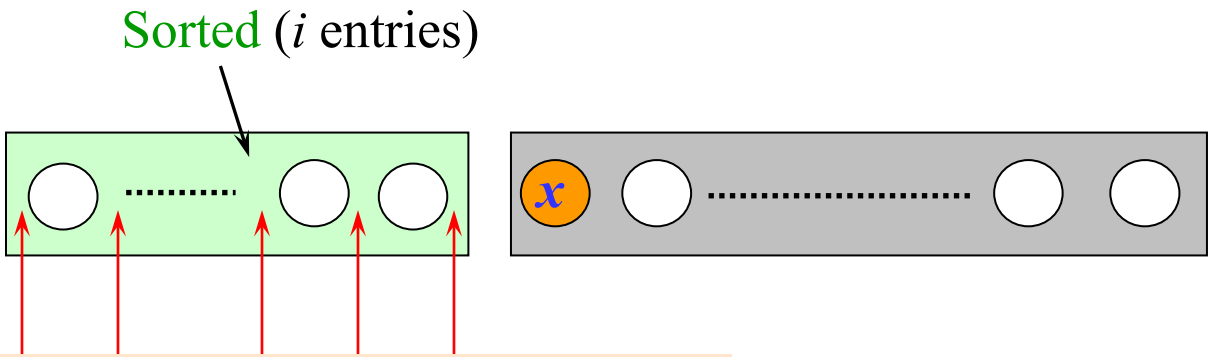


- At the beginning, there are n-1 entries in the unsorted segment, so:

$$W(n) \le \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

The input for which the upper bound is reached does exist, so: $W(n) \in \Theta(n^2)$

Average Behavior



***x* may be located in any one of the *i*+1 intervals(inclusive), assumingly, with the same probabiliy**

- Assumptions:
  - All permutations of the keys are equally likely as input.
  - There are not different entries with the same keys.

Note: For the (i+1)th interval (leftmost), only i comparisons are needed.

Average Complexity

- The average number of comparisons in **shiftVac** to find the location for the *i*th element:

$$\frac{1}{i+1}\sum_{j=1}^{i} j + \frac{1}{i+1}(i) = \frac{i}{2} + \frac{i}{i+1} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

for the leftmost interval

- For all n-1 insertions:

$$A(n) = \sum_{i=1}^{n-1}\left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) = \frac{n(n-1)}{4} + n - 1 - \sum_{j=2}^{n}\frac{1}{j}$$

$$= \frac{n(n-1)}{4} + n - \sum_{j=1}^{n}\frac{1}{j} = \frac{n^2}{4} + \frac{3n}{4} + \ln n \in \Theta(n^2)$$

问题1：算法的效率 (续)

- 你会分析merge sort的时间复杂度吗？

6 5 3 1 8 7 2 4

问题1：算法的效率 (续)

- 你会分析merge sort的时间复杂度吗？

```
function merge_sort(list m)
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted sublists.
    return merge(left, right)
```

问题1：算法的效率 (续)

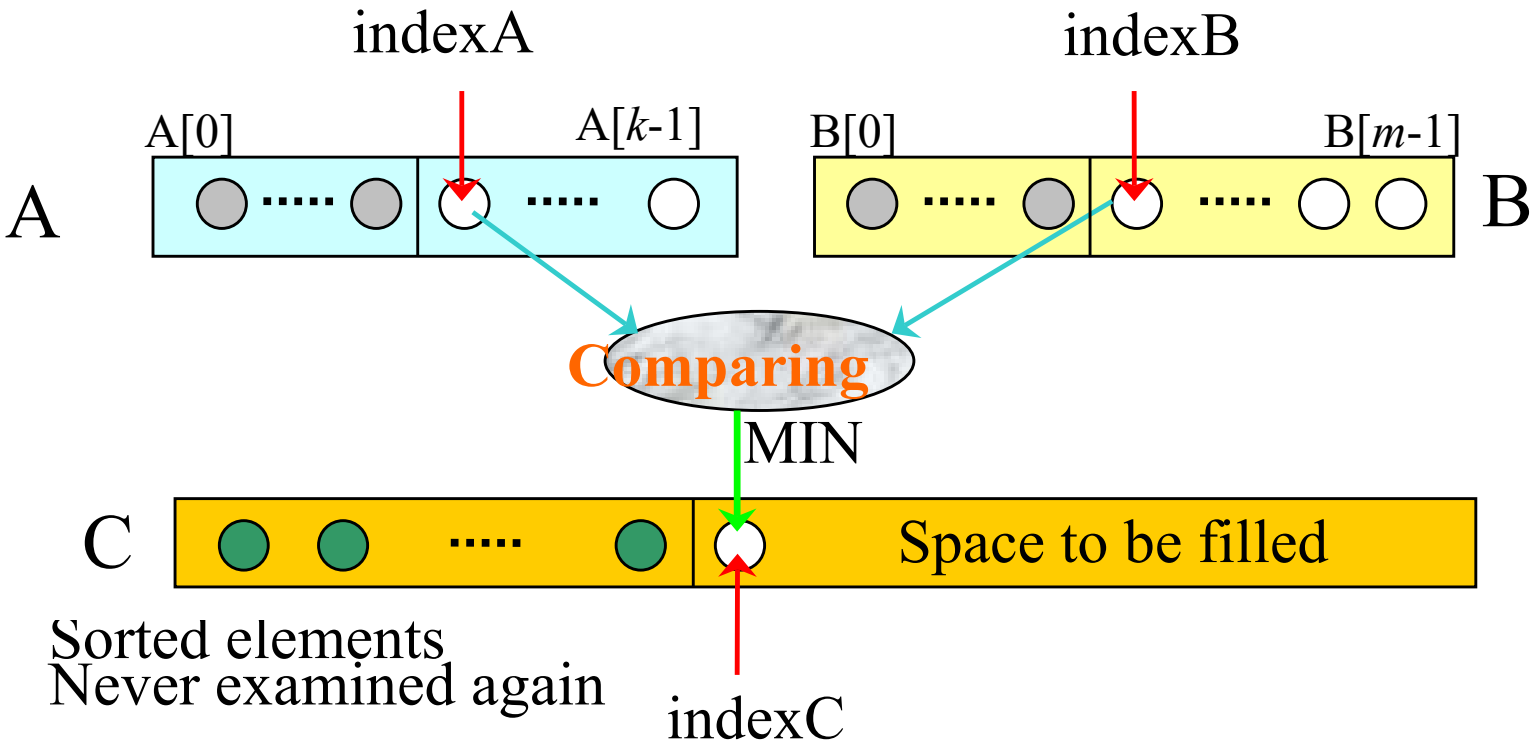
- 你会分析merge sort的时间复杂度吗？

```
function merge(left, right)
    var result := empty list

    while left is not empty and right is not empty do
        if first(left) ≤ first(right) then
            append first(left) to result
            left := rest(left)
        else
            append first(right) to result
            right := rest(right)

    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)
    while left is not empty do
        append first(left) to result
        left := rest(left)
    while right is not empty do
        append first(right) to result
        right := rest(right)
    return result
```

Merging Sorted Arrays



引申问题

- 一个序列的元素远小于另一个序列，这样的merge如何处理？

问题2：问题的难度

- reasonable和tractable都表述多项式时间，区别是什么？

- An algorithm whose order-of-magnitude time performance is bounded from above by a polynomial function of *N*, where *N* is the size of its inputs, is called a **polynomial-time algorithm**, and will be referred to here as a **reasonable** algorithm.
- As far as the algorithmic problem is concerned, a problem that admits a reasonable or polynomial-time solution is said to be **tractable**, whereas a problem that admits only unreasonable or exponential-time solutions is termed **intractable**.

问题2：问题的难度(续)

- “算法的效率和问题的难度互为上下界”结合之前的sort，谈谈你对这句话的理解

问题2：问题的难度 (续)

- 对于intractable problem，怎么办？