420-J13-AS
Advanced Data Structure
Lecture 03 - Sorting Conclusion

Felix Soumpholphakdy
LaSalle College
24 January 2019

Today

- Herb problem
- Quicksort
- Counting sort
- Summary

Herb Problem

- Your organization gathers and sells herbs
- New herbs are scanned and tagged:
    - Quality $\in [0, 100]$
    - ID = 1000, 1001, 1002, ...
- When someone buys a herb, you <u>must</u> sell the herb with the highest quality
- Find the best herb and return its ID number
- What to optimize:
    - Efficiency of adding a new herb
    - Efficiency of finding the best herb
    - Efficiency of removing the best herb

Quicksort

- One of the most popular sorting algorithms
- Developed by Tony Hoare
- Worst-case is $O(n^2)$
- Best and average-case is $O(n \lg n)$
- Divide-and-conquer

Divide And Conquer

- Divide the array A[p..r] into two subarrays L = A[p..q-1] and R = A[q+1..r]
  Such that: (any element in L) $\leq$ A[q] $\leq$ (any element in R)
- Sort the subarrays by recursive calls to quicksort
- No need to combine the subarrays because they are already sorted in place

Quicksort pseudocode

QUICKSORT(A, p, r)

1. if p < r
2. q = PARTITION(A, p, r)
3. QUICKSORT(A, p, q - 1)
4. QUICKSORT(A, q + 1, r)

Partition pseudocode

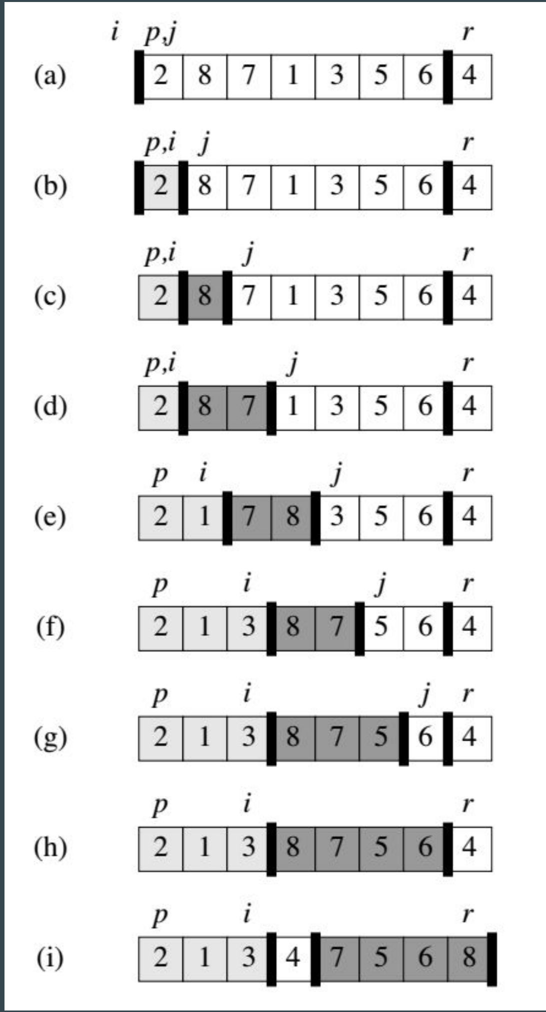
```
PARTITION(A, p, r)

x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i = 1
```

Partition Example

In the loop iteration:

1. if (A[j] ≤ A[r])
   1.1. i++
   1.2. swap A[i] and A[j]
2. j++

When finished swap A[i + 1] and A[r]



Exercises 7.1 (CLRS)

1. Illustrate the operation of PARTITION on the array
   A = <13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11>
2. What value of q does PARTITION return when all elements in the array A[p..r] have the same value?
3. Give a brief argument that the running time of PARTITION on a subarray of size n is $\Theta(n)$
4. How would you modify QUICKSORT to sort into nonincreasing order?

Quicksort Analysis

- Running time depends on the partitioning
- Unbalanced = worst-case = $O(n^2)$
- Balanced = best-case and average-case = $O(n \lg n)$
- Sorts in place

Random

- Using chaos to create order
- Instead of using A[r] as the pivot, randomly use any element in A[p..r]
- Only extra step is swapping A[r] with the randomly selected element
- Because we randomly select the pivot, the partition is balanced on average
- This gives a new O(n lg n) expected running time
- This makes the algorithm non-deterministic: the same input will not always give the same output

Updated Procedures

RANDOMIZED-PARTITION(A, p, r)

1. i = RANDOM(p, r)
2. exchange A[r] with A[i]
3. return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)

1. if p < r
2.   q = RANDOMIZED-PARTITION(A, p, r)
3.   RANDOMIZED-QUICKSORT(A, p, q - 1)
4.   RANDOMIZED-QUICKSORT(A, q + 1, r)

Sorting in Linear Time

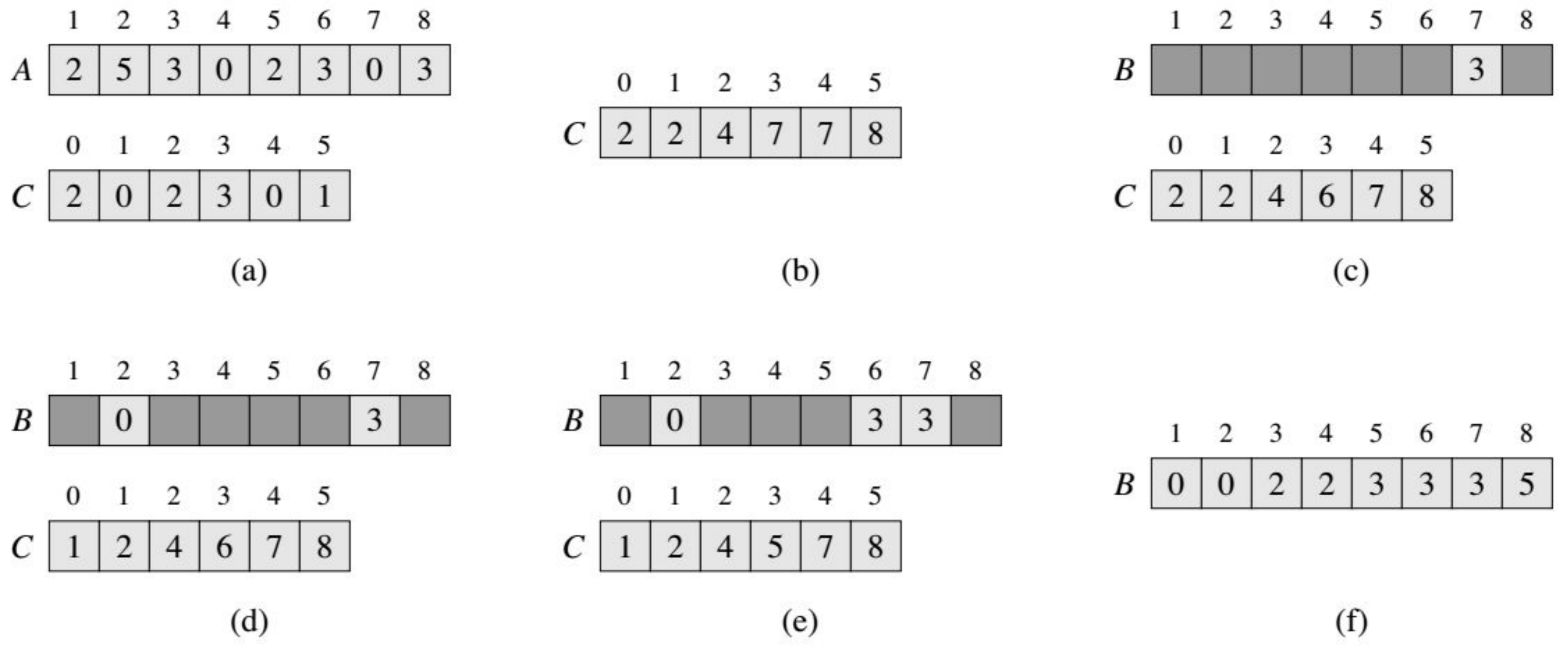
- The sorting algorithms we saw use comparisons to determine the order
- It is proven that comparison sorts are $O(n \lg n)$ in the worst case
- Non-comparison sorts exist which can be faster than $O(n \lg n)$

Counting Sort

- A non-comparison sort
- Assumes that all elements are in the range [0, k] where k is an integer
- Counts how many elements precede any element x and puts x at A[count]
- Also handles the case where an element is duplicated

Counting Sort

A = input array, B = output array, C = temporary array for counting



COUNTING-SORT(A, B, k)

```
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i .
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i .
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

Counting Sort Analysis

- Counting is O(n)
- Cumulative sum is O(k)
- Placing elements is O(n)
- Running time is O(n + k) ⟶ O(n), when k = O(n)
- Stable sort: numbers with the same value maintain their relative order in the input and the output

Exercises 8.2 (CLRS)

1. Illustrate the operation of COUNTING-SORT on the array
   A = <6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2>

Summary

Algorithm	Advantage	Disadvantage
Insertion sort	O(n) best-case	$O(n^2)$ worst-case
Merge sort	Stable	Not in-place
Heapsort	In-place	Not stable
Quicksort (randomized)	Often the fastest	Non-deterministic
Counting sort	O(n) worst-case	Has prerequisites

References

- Cormen, T. (2009). Introduction to algorithms. Cambridge, Mass.: MIT Press.
- https://en.wikipedia.org/wiki/Quicksort
- https://en.wikipedia.org/wiki/Counting_sort