420-J13-AS
Advanced Data Structure
Lecture 02 - Analysis

...

Felix Soumpholphakdy
LaSalle College
17 January 2019

Analysis Terms

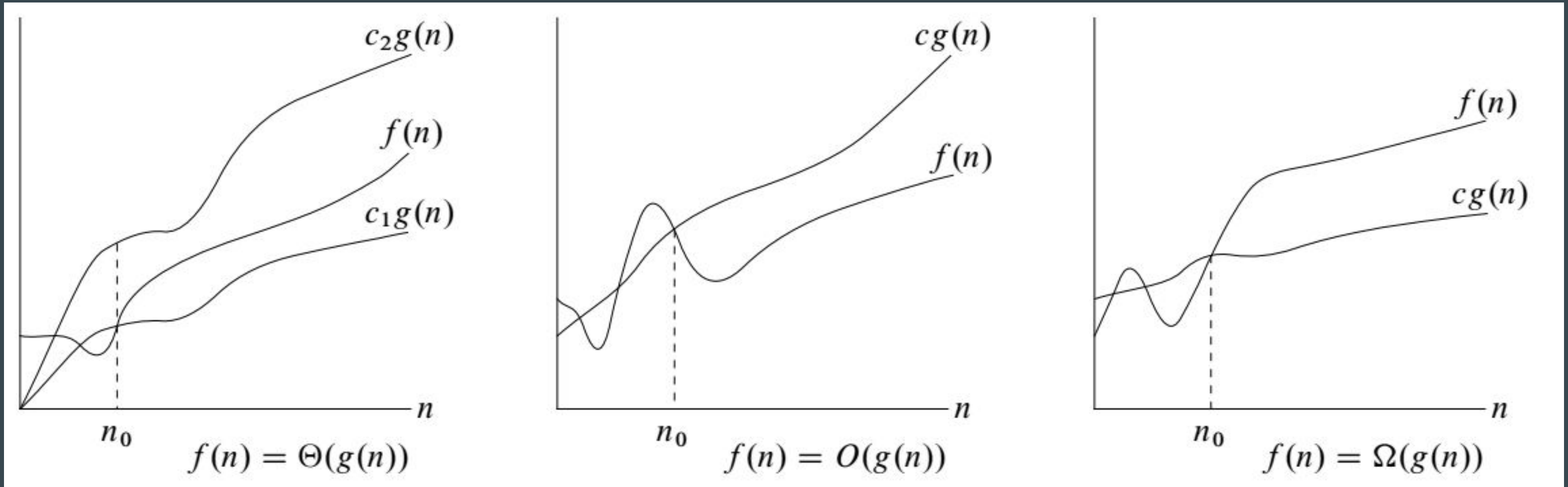
- Input size is number of items in the input
  - Number of bits
  - Number of integers
  - Number of points
- Running time is number of steps executed
  - Add, substract, multiply, divide
  - Load, store, copy
  - Branching

Worst-case

- We mostly focus on the worst-case running time for algorithms
- Three reasons
    - It is a guarantee that the algorithm will not take longer
    - Worst case can occur often
    - Average case is often almost as bad as worst case

Asymptotic Notation

- O-notation (Big Oh) is an asymptotic upper bound
- Ω-notation is an asymptotic lower bound
- Θ-notation is both an upper and lower bound



Insertion Sort Analysis

- Worst-case running time is $O(n^2)$
- Best-case running time is $O(n)$
- Algorithm follows an incremental approach

Divide-And-Conquer Approach

- Recursive in structure
- Three steps for every recursion:
  - Divide the problem into subproblems
  - Conquer the problems by solving them recursively
  - Combine the solutions to the subproblems
- Merge sort:
  - Divide n-element sequence into two halves (n/2)
  - Sort the two subsequences recursively
  - Merge the two sorted subsequences

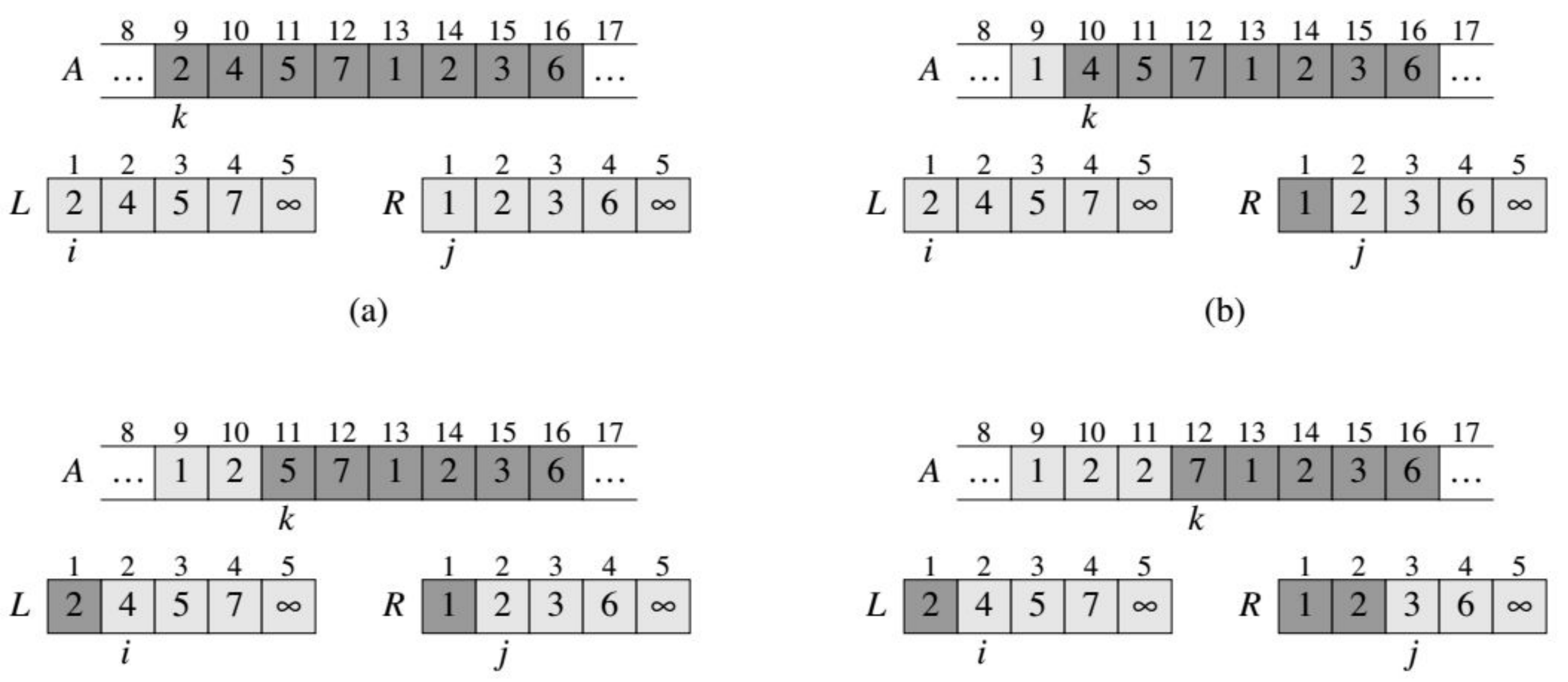
Merge Procedure

- Given two piles of cards that are sorted, we want to combine them
- Look at the top card of each pile and put the smallest one on another pile
- Repeat until all cards end up in that pile
- MERGE(A, p, q, r)
  - A is the array
  - $p \leq q < r$ are the indices of the array
  - Subarrays A[p .. q] and A[q + 1 .. r] are sorted
  - Output is A [p .. r] is sorted
- We use a sentinel value (∞) to simplify the code. Imagine that the last card of every pile is infinity, so we don’t need conditions to check whether a pile is empty or not.

MERGE(A, p, q, r)

```
n_1 = q - p + 1
n_2 = r - q
let L[1 .. n_1 + 1] and R[1 .. n_2 + 1] be new arrays
for i = 1 to n_1
    L[i] = A[p + i - 1]
for j = 1 to n_2
    R[j] = A[q + j]
L[n_1 + 1] = ∞
R[n_2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] < R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

MERGE(A, p, q, r) Example



MERGE-SORT(A, p, r)

```
if p < r
    q = (p + r)/2
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Merge Sort Analysis

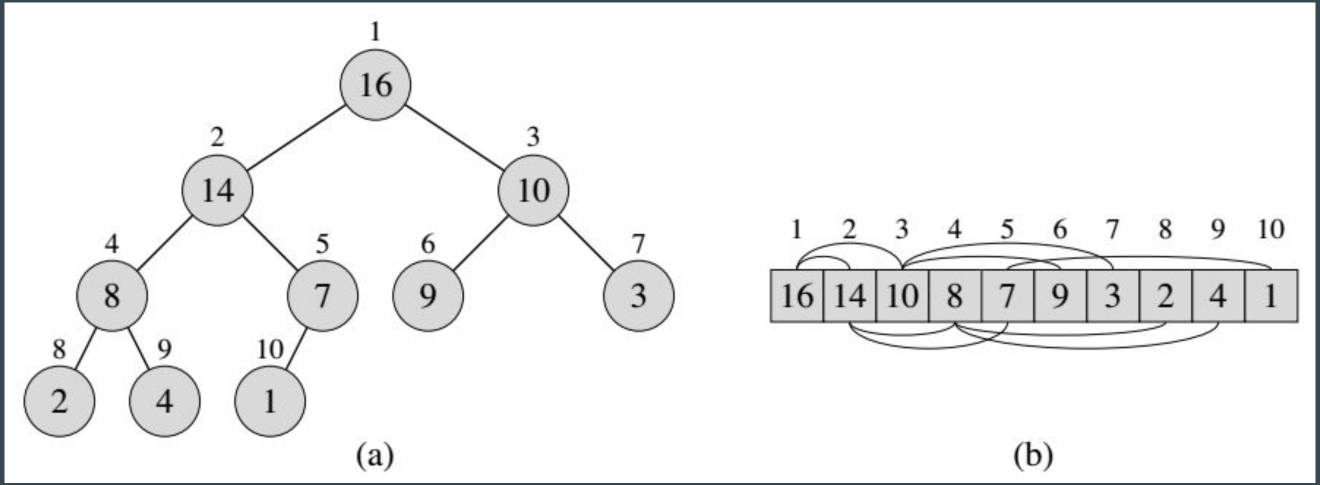
- Worst-case running time is O(n log n)
- Best-case running time is O(n log n)
- Algorithm follows a divide-and-conquer approach

Heapsort

- Running time is O(n lg n)
  - Like merge sort
  - Unlike insertion sort
- Can sort in place
  - Like insertion sort
  - Unlike merge sort
- Uses the heap data structure

Heap

- Binary tree
- Given the index i
  - PARENT(i) => return i/2
  - LEFT(i) => return 2i
  - RIGHT(i) => return 2i + 1



Heap Property

- Max-heap property is every node i other than the root: $A[PARENT(i)] \geq A[i]$
- Min-heap property is every node i other than the root: $A[PARENT(i)] \leq A[i]$
- Height of a tree with n elements is lg(n)

Exercises 6.1 (CLRS)

1. What are the minimum and maximum numbers of elements in a heap of height *h*?
2. Show than an n-element heap has height lg(n)
3. Show that in any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree

Maintaining The Heap Property

- MAX-HEAPIFY(A, i) maintains the heap property
  - A is the array
  - i is an index into the array
- When called, we assume that LEFT(i) and RIGHT(i) are max-heaps
- The procedure lets the value at A[i] float down

MAX-HEAPIFY(A, i)

```
L = LEFT(i)
R = RIGHT(i)
if (L ≤ A.heap-size and A[L] > A[i])
    largest = L
else largest = i
if (R ≤ A.heap-size and A[R] > A[largest]
    largest = R
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Exercises 6.2 (CLRS)

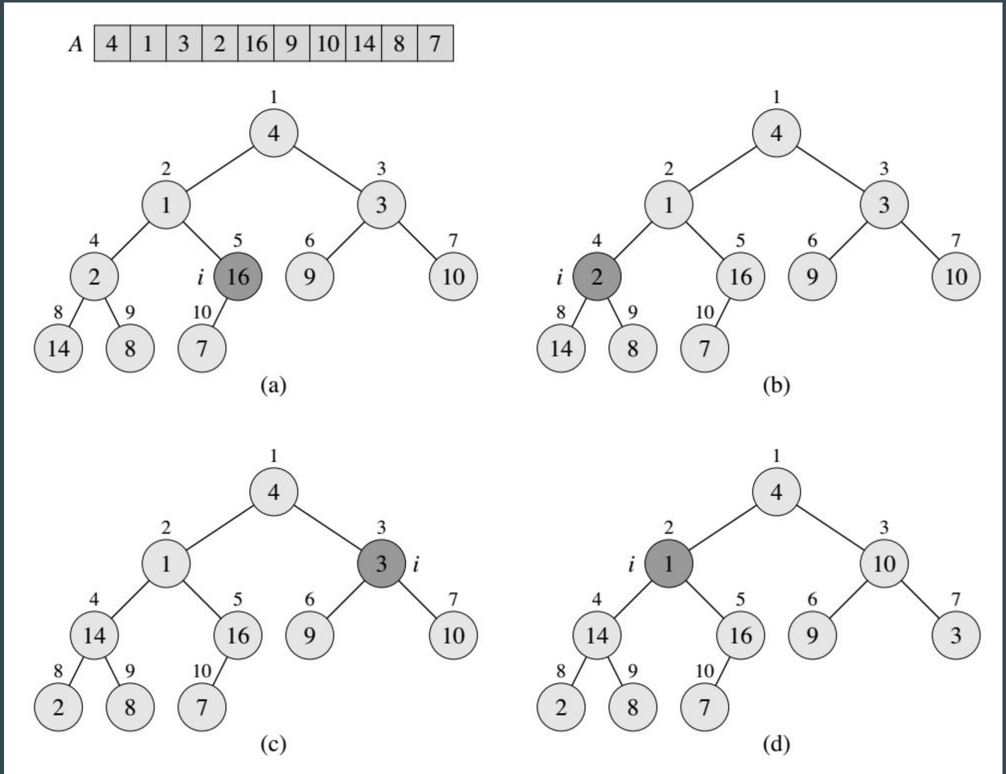
1. Illustrate the operation MAX-HEAPIFY(A, 3) on the array
   A = <27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>
2. Starting with the procedure MAX-HEAPIFY, write pseudocode for the procedure MIN-HEAPIFY(A, i), which performs the corresponding manipulation on a min-heap
3. What is the effect of calling MAX-HEAPIFY(A, i) when A[i] is larger than its children?

Building a heap

- Use procedure MAX-HEAPIFY in a bottom-up manner to convert A[1 .. n] into a max-heap
- The elements in the subarray A[(n/2 + 1) .. n)] are all leaves of the tree
- The procedure BUILD-MAX-HEAP goes through the remaining nodes

BUILD-MAX-HEAP(A)

```
A.heap-size = A.length
for i = A.length/2 downto 1
    MAX-HEAPIFY(A, i)
```



Exercises 6.3 (CLRS)

1. Illustrate the operation of BUILD-MAX-HEAP on the array A = <5, 3, 17, 10, 84, 19, 6, 22, 0>
2. Why do we want the loop index i in line 2 of BUILD-MAX-HEAP to decrease from A.length/2 to 1 rather than increase from 1 to A.length/2 ?

Heapsort Algorithm

- First build a max-heap with BUILD-MAX-HEAP
- Because the maximum element is at A[1] we can swap it with A[n]
- Discard node n from the heap by decrementing A.heap-size
- Restore the max-heap property with MAX-HEAPIFY(A, 1)

HEAPSORT(A)

1. BUILD-MAX-HEAP(A)
2. for i = A.length downto 2
3.   exchange A[1] with A[i]
4.   A.heap-size = A.heap-size - 1
5.   MAX-HEAPIFY(A, 1)

Heapsort Analysis

- HEAPSORT takes time O(n lg n)
- BUILD-MAX-HEAP takes O(n)
- Each (n - 1) calls to MAX-HEAPIFY takes time O(lg n)

Exercises 6.4 (CLRS)

1. Illustrate the operation of HEAPSORT on the array
   A = <5, 13, 2, 25, 7, 17, 20, 8, 4>

References

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