ECS 122A
Algorithm Design and Analysis

Instructor: Qirun Zhang

Course slides (partially) adopted from the notes by David Luebke.

Agenda

- Strassen's method for matrix multiplication
- More on master theorem
- Introduction to heapsort

Course updates

- About midterm

Strassen's method

- Strassen's method – Step 1: Divide

$$A = \begin{array}{cc} & \begin{array}{cc} \frac{n}{2} & \frac{n}{2} \end{array} \\ \begin{array}{c} \frac{n}{2} \\ \frac{n}{2} \end{array} & \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \end{array} \quad \text{and} \quad B = \begin{array}{cc} & \begin{array}{cc} \frac{n}{2} & \frac{n}{2} \end{array} \\ \begin{array}{c} \frac{n}{2} \\ \frac{n}{2} \end{array} & \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \end{array}$$

Strassen's method

- Strassen's method – Step 2: Compute 10 matrices by $\pm$ only:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} \\
S_2 &= A_{11} + A_{12} \\
S_3 &= A_{21} + A_{22} \\
S_4 &= B_{21} - B_{11} \\
S_5 &= A_{11} + A_{22} \\
S_6 &= B_{11} + B_{22} \\
S_7 &= A_{12} - A_{22} \\
S_8 &= B_{21} + B_{22} \\
S_9 &= A_{11} - A_{21} \\
S_{10} &= B_{11} + B_{12}
\end{aligned}
$$

Strassen's method

- Strassen's method – Step 3: Compute 7 matrices by multiplication:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 \\
P_2 &= S_2 \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 \\
P_5 &= S_5 \cdot S_6 \\
P_6 &= S_7 \cdot S_8 \\
P_7 &= S_9 \cdot S_{10}
\end{aligned}
$$

Strassen's method

- Strassen's method – Step 4: Add and subtract the P_i to construct submatrices C_{ij} of the product C

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

The Master Theorem Revisited

- if T(n) = aT(n/b) + f(n) then

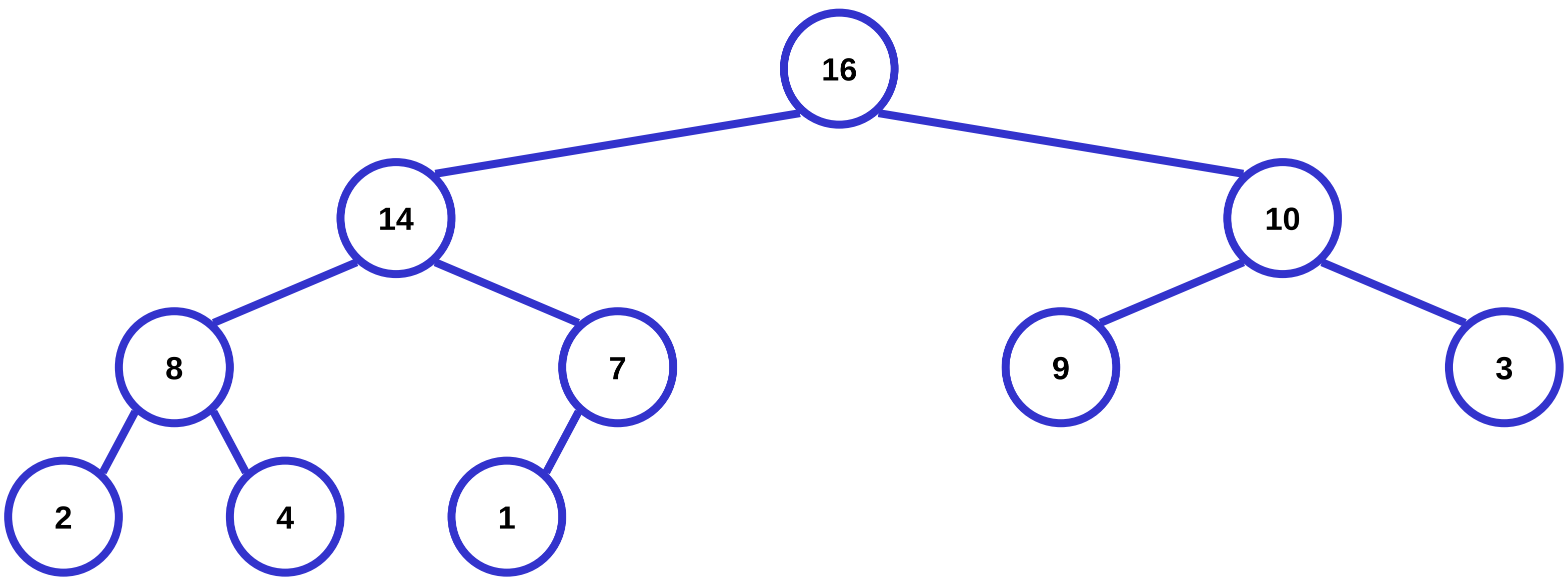
$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \text{ AND} \\ & af(n/b) < cf(n) \text{ for large } n \end{cases} \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\end{matrix}}\right\} \begin{matrix} \varepsilon > 0 \\ c < 1 \end{matrix}$$

Sorting Revisited

- So far we've talked about two algorithms to sort an array of numbers
  - What is the advantage of merge sort?
  - What is the advantage of insertion sort?
- Next on the agenda: *Heapsort*
  - Combines advantages of both previous algorithms

Heaps

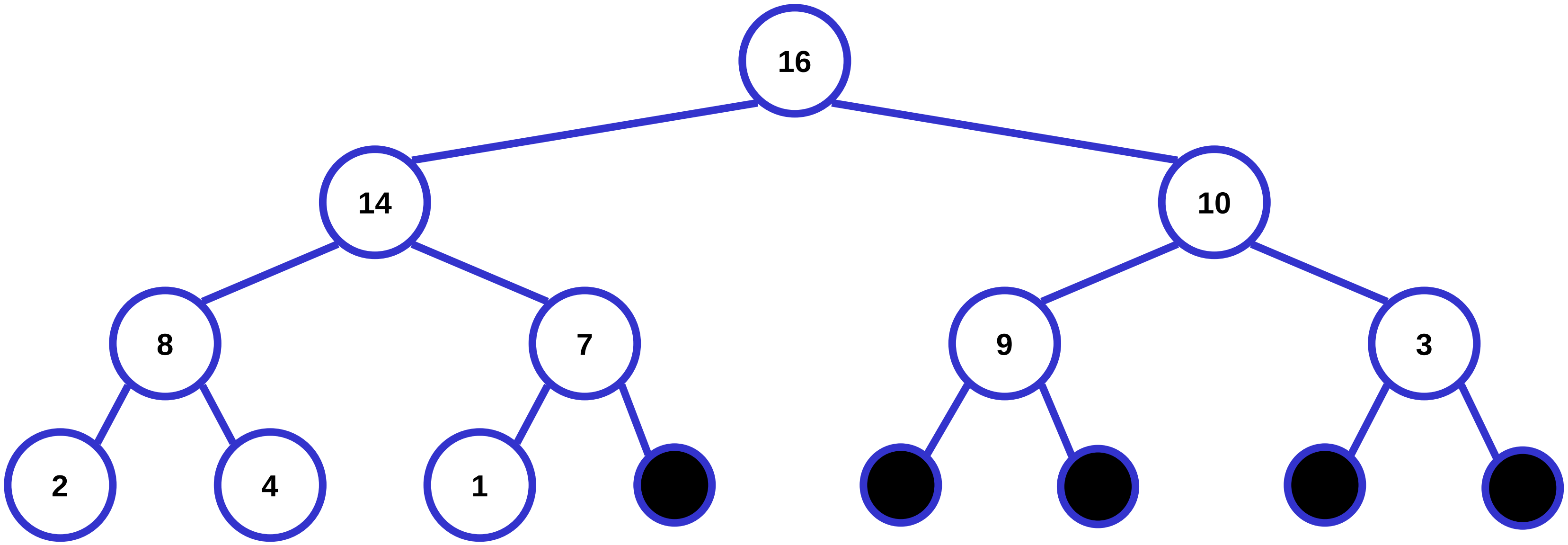
- A *heap* can be seen as a complete binary tree:



  - *What makes a binary tree complete?*
  - *Is the example above complete?*

Heaps

- A *heap* can be seen as a complete binary tree:

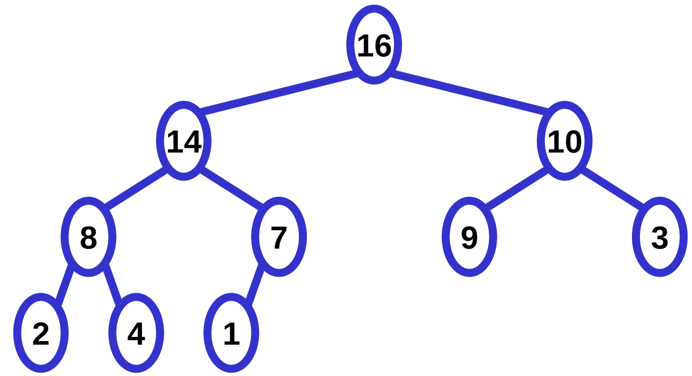


  - The book calls them "nearly complete" binary trees; can think of unfilled slots as null pointers

Heaps

- In practice, heaps are usually implemented as arrays:

A =	*16*	*14*	*10*	*8*	*7*	*9*	*3*	*2*	*4*	*1*	=



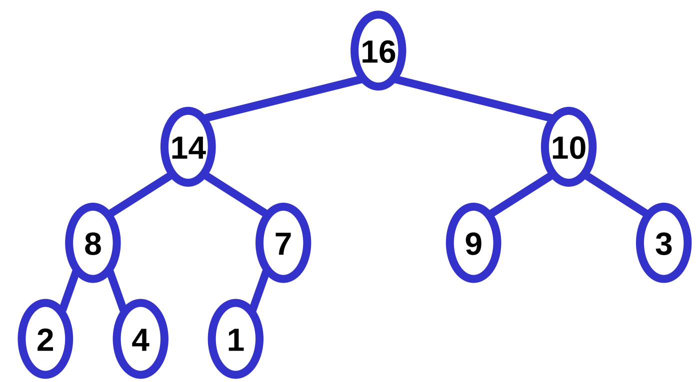
Heaps

- To represent a complete binary tree as an array:
  - The root node is $A[1]$
  - Node i is $A[i]$
  - The parent of node i is $A[i/2]$ (note: integer divide)
  - The left child of node i is $A[2i]$
  - The right child of node i is $A[2i + 1]$

A =

16	14	10	8	7	9	3	2	4	1

=



Referencing Heap Elements

- So…

```
Parent(i) { return ⌊i/2⌋; }
Left(i) { return 2*i; }
right(i) { return 2*i + 1; }
```

The Heap Property

- Heaps also satisfy the *heap property*:

  $A[Parent(i)] \geq A[i]$ for all nodes $i > 1$

  - In other words, the value of a node is at most the value of its parent
  - *Where is the largest element in a heap stored?*
- Definitions:
  - The *height* of a node in the tree = the number of edges on the longest downward path to a leaf
  - The height of a tree = the height of its root

Heap Height

- *What is the height of an n-element heap? Why?*
- This is nice: basic heap operations take at most time proportional to the height of the heap

The End