WELCOME TO CSE 100!

Advanced Data Structures in C++
Instructors

- Debashis Sahoo ([dsahoo@ucsd.edu](mailto:dsahoo@ucsd.edu))
  - PhD (Electrical Engineering, Stanford University)
  - Assistant Professor (Department of Pediatrics)
- Office: BRF II – 2119 (Biomedical Research Facility II)
- Office hours:
  - Thursday: 10am – 12pm
  - Or by appointment
Instructors

- Nuno Bandeira (bandeira@ucsd.edu)
  - PhD (2007, Computer Science and Bioinformatics, UCSD)
- Office: 4210 EBU3B
- Office hours:
  - Mon: 3pm – 5pm
  - Or by appointment
Information about TAs and tutors

TAs:
Ning Liu
Mingxun Wang
Wenjia Ouyang

Tutors:
Chao Zhang
Shengyang Shi
Shunxin Lu

Office/lab hours available on the course website

http://cseweb.ucsd.edu/classes/fa15/cse100-a
CLICKERS OUT

Set your frequency to BB

A+ B- (AA-)

A- B+ (AA-)
Have you been in a class that used peer instruction before?

A. Yes
B. No
C. I’m not sure
iClickers: You must bring them

- Buy an iClicker at the Bookstore
- Register it on TED by Sept 28 (Monday).

Text on reserve

- Data Structures and Algorithms C++, Fourth Edition, by Adam Drozdek
- C++ for Java Programmers, Mark Allen Weiss
About This Class

You must **attend** class
You must **prepare** for class
You must **participate** in class
Course Logistics

- Less than 75% iClicker response ≡ missing a class
- You may miss up to 2 classes with no penalty
- One midterm

Grading
- Reading quizzes: 5%
- Participation points: 5%
- Programming Assignments: 35%
- Midterm exam: 20%
- Final exam: 35%
Assignment 1: Due Oct 8 @ 8pm

- Programming in C++
- Binary Search Tree (BST)
- Pair programming
Pair Programming guidelines

Basic rules
- All code written with two programmers at one machine
- You must plan *ahead of time* when you will get together
- You can change partner for each PA
- Don't be a jerk

Selecting partners: Factors to consider
- Schedule compatibility
- Roughly equal “eagerness”
- Roughly equal experience
- Partner from other section

You must choose your partner by MONDAY
Reading Quizzes

• Every Tuesday before class
• administered offline via TED
Why study data structures?

Discuss with your group

1. Interns hire quickly
2. Strong resume
3. Efficiency
4. Make money
5. Cure Cancer
Topics for the course

• In CSE 100, we will build on what you have already learned about programming: procedural and data abstraction, object-oriented programming, and elementary data structures and algorithm design, implementation, and analysis.

• We will build on that, and go beyond it, to learn about more advanced, high performance data structures and algorithms:
  • Balanced search trees: AVL, red-black, B-trees
  • Binary tries and Huffman codes for compression
  • Graphs as data structures, and graph algorithms
  • Data structures for disjoint-subset and union-find algorithms
  • Hash tables, hash functions, and hashing techniques
  • Randomized data structures: skip lists, treaps
  • Amortized cost analysis
  • The C++ standard template library (STL)
If you only remember one thing today

http://cseweb.ucsd.edu/classes/fa15/cse100-a

Read the syllabus. Know what’s required. Know how to get help.
Data structures

• A data structure is... a structure that holds data

• A data structure is an object that offers certain useful operations through its “Application Programmer Interface” (API), for example: storing, retrieving, and deleting data of a certain type

• A data structure may offer certain performance guarantees on its operations, for example certain best-, worst-, or average-case time or space costs
  • To meet those performance guarantees, a data structure may need to be implemented in a particular way

• In CSE 100 we will study the performance guarantees that are permitted by various data structure implementations

• We will begin by reviewing trees...
A review of trees

• A tree is a hierarchical (not just linear, and not unstructured!) data structure

• A tree is a set of elements called *nodes*, structured by a "parent" relation:
  - If the tree is nonempty, exactly one node in the set is the *root* of the tree
  - The root of a tree is the unique node that has no parent
  - Every node in the set except the root has *exactly one other node* that is its parent
Which of the following is/are a tree?

A.  

B.  

C.  

D. A & B
E. All of A-C
Which of the following is/are a tree?

A. 

B. 

C. 

D. A & B

E. All of A-C
Tree terminology: definitions

- **Children** of a node $P$: the set of nodes that have $P$ as parent
- **Descendant** of a node $P$:
  - If a node $C$ is a child of $P$, then $C$ is a descendant of $P$
  - If a node $C$ is a child of a descendant of $P$, then $C$ is a descendant of $P$
- **Ancestor** of a node $C$: if $C$ is a descendant of $P$, then $P$ is an ancestor of $C$
- **Root** of a tree: the unique node in the tree with no parent
- **Leaves** of a tree: the set of nodes with no children
- **Subtree**:
  - The empty tree is a subtree of every tree
  - Any node of a tree together with its descendants is a subtree of the tree
- **Level or depth** of a node (using ‘zero-based’ counting):
  - The level of the root is 0.
  - The level of any non-root node is $1 +$ the level of its parent (this is equal to the number of edges on the path from the root to the node)
- **Height** of a node: the height of a node is the number of edges on the longest path from the node to a leaf
- **Height** of a tree: the height of the root of the tree
Which of the following is/are a binary tree?

A.  
B.  
C.  
D.  A and C  
E.  B and C
Important binary tree properties

• Consider a “completely filled” binary tree (every level that has any nodes at all has as many nodes as possible):
  • How many nodes at level 0?
  • How many nodes at level 1?
  • How many nodes at level 2?
  • How many nodes at level 3?

• Generalizing, how many nodes at level \( L \)?

• And so, how many nodes in a completely filled binary tree of height \( H \)?
  \[
  \sum_{L=0}^{H} 2^L = 2^{H+1} - 1 = N
  \]

• And so, what is the height of a completely filled binary tree with \( N \) nodes?
  \[
  H = \log_2(N+1) - 1
  \]
In a completely filled binary tree with $N$ nodes...

- Generalizing, how many nodes at level $L$?
  $$2^L$$

- And so, how many nodes in a completely filled binary tree of height $H$?
  $$N = \sum_{L=0}^{H} 2^L = 2^{H+1} - 1$$

- And so, what is the height of a completely filled binary tree with $N$ nodes?
  $$2^{H+1} = N + 1$$
  $$H + 1 = \log_2(N + 1), \text{ so}$$
  $$H = \Theta(\log N), \text{ and } H = \Omega(\log N), \text{ and so } H = \Theta(\log N)$$
Reviewing “big-O” notation

• Write:

\[ g(N) = O(f(N)) \]

• And say: " \( g(N) \) is ‘big-O’ of \( f(N) \)” if there are positive constants \( c, n_0 \) such that for all \( N \geq n_0 \)

\[ g(N) \leq cf(N) \]

… that is, \( g \) eventually grows no faster than \( f \) (times a constant).

… \( f \) gives an asymptotic upper bound on the rate of growth of \( g \).

… the order of \( g \) is at most the order of \( f \)
Reviewing “big-Ω” notation

• Write:

\[ g(N) = \Omega(f(N)) \]

• And say: "g(N) is ‘big-omega’ of f(N)" if there are positive constants c, \( n_0 \) such that for all \( N \geq n_0 \)

\[ g(N) \geq cf(N) \]

… that is, g eventually grows at least as fast as f (times a constant).

… f gives an asymptotic lower bound on the rate of growth of g.

… the order of g is at least the order of f.
Reviewing “big-Θ” notation

• Write:

\[ g(N) = \Theta(f(N)) = \Omega(f(N)) = O(f(N)) \]

• And say: "g(N) is ‘big-theta’ of f(N)" if g(N) is both ‘big-O’ and ‘big-omega’ of f(N).

... f gives a good qualitative estimate -- a “tight bound” -- on the rate of growth of g
... the order of g is the same as the order of f
About you…

What is your familiarity/confidence with proving running time bounds (e.g. Big-O)?
A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.
In a completely filled K-ary tree with M nodes...

- Generalizing, how many nodes at level L?

- And so, how many nodes in a completely filled binary tree of height H?

\[ N = \sum_{L=0}^{H} K^L = \frac{K^{H+1} - 1}{K - 1} \]

- And so, what is the height of a completely filled binary tree with N nodes?

\[ K^{H+1} = N(K - 1) + 1 \]
\[ H + 1 = \log_K(N(K - 1) + 1), \text{ so} \]
\[ H = O(\log N), \text{ and } H = \Omega(\log N), \text{ and so } H = \Theta(\log N) \]
Tree properties, continued

- A completely filled K-ary tree with N nodes has the minimum height possible of any K-ary tree with N nodes... In fact for any K-ary tree,

\[ H \geq \log_K (N(K - 1) + 1) - 1 \]

and so

\[ H \geq \lceil \log_K N \rceil - 2 \]

- But what is the maximum height possible for a K-ary tree with N nodes? __________
Binary Search Tree – What is it?

What are the numbers in the nodes?
Binary Search Tree – What is it?

For any node,
Keys in node’s left subtree \( \leq \) Node key
Node key \( \leq \) Keys in node’s right subtree

Do the keys have to be integers?
Which of the following is/are a binary search tree?

A. 42
   32
   12

B. 42
   12
   32

C. 42
   12
   32
   65

D. 42
   32
   56
   12
   45

E. More than one of these
Binary Search Trees

- What are the operations supported?
  - Insert, search, delete
- What are the running times of these operations?
  - $O(\log N)$, $O(N)$, $O(N)$ / Are $O(\log N)$
- How do you implement the BST i.e. operations supported by it?
Binary Search Trees

• What is it good for?
  • If it satisfies a special property i.e. Balanced, you can think of it as a dynamic version of the sorted array
PA1: Implementing BST operations in C++

- You need to implement
  - find() – find an element
  - size() – returns total number of elements
  - clear() – deletes all the elements
  - empty() – checks if the BST is empty
  - successor() – returns the next element in an in-order traversal
- And the iterator pattern (we will talk about it later this week)
Under the hood: Searching an element in the BST

To search for element with key $k$
1. Start at the root
2. If $k=\text{key(root)}$, found key, stop.
3. Else If $k< \text{key(root)}$, recursively search the left subtree: $T_L$
   Else recursively search the right subtree: $T_R$

Search for 41.
Now search for 43.
Under the hood: Finding the successor of an element in the BST

Which node is the successor of 56? How would you find it?
Traversing the BST in sorted order

Different methods of tree traversal:
- Pre order traversal
- Post order traversal
- In order traversal

successor() – returns the next element in an in-order traversal
In-order traversal of BST

Which of the following results from an in-order traversal of a BST?

A. Nodes are visited in the order in which they were inserted into the BST
B. Nodes are visited in order of the number of children that they have
C. Nodes are visited in sorted order of their keys
D. None of the above