Data Structures

Giri Narasimhan
Office: ECS 254A
Phone: x-3748
giri@cs.fiu.edu
Review
How to compare big-Ohs?

We say that \( f(n) = O(g(n)) \) if

- There exists positive integer \( n_0 \), and
- A positive constant \( c \), such that
- For all \( n \geq n_0 \),
  \[ f(n) \leq c \cdot g(n) \]

In other words, for all large enough values of \( n \), there exists a constant \( c \), such that
- \( f(n) \) is always bounded from above by \( c g(n) \)

Note that the condition may be violated for a finitely many small values of \( n \).
To prove $f(n) = O(g(n))$

1. Pick a value of $c$
2. Find a $n_0$ such that
3. $f(n) \leq c g(n)$ for all $n \geq n_0$
4. If not, go back to step 1 and refine value of $c$

To prove $f(n) \neq O(g(n))$

- Assume that $c$ is fixed to some positive value
- And assume there exists a $n_0$ such that
- $f(n) \leq c g(n)$ for all $n \geq n_0$
- Now try to prove a contradiction to this claim
MaxSubseqSum(A)

Initialize maxSum to 0

N := size(A)

For \( i = 1 \) to N do
  For \( j = i \) to N do
    Initialize thisSum to 0
    for \( k = i \) to \( j \) do
      add \( A[k] \) to thisSum
    if (thisSum > maxSum) then
      update maxSum

\[
\sum_{k=i}^{j} 1 = j - i + 1
\]

\[
\sum_{j=i}^{N} (j - i + 1) = \frac{(N - i + 1)(N - i + 2)}{2}
\]

\[
\sum_{i=1}^{N} \frac{(N - i + 1)(N - i + 2)}{2}
\]

\[
= \sum_{i=1}^{N} \frac{i^2}{2} - \left( N + \frac{3}{2} \right) \sum_{i=1}^{N} i + \frac{1}{2} \left( N^2 + 3N + 2 \right) \sum_{i=1}^{N} 1
\]

\[
= \frac{N^3 + 3N^2 + 2N}{6} = O(N^3)
\]
Example 1

SampleMethod1 (N)
For i = 0 to N do
  For j = i to N do
    sum = i + j
Report sum

Big Oh: ____________

Example 2

public void sampleMethod2(int[] a)
{
  int index = 0;
  for(int i = a.length-1; i >= 0; i--)
    if(a[i] > target && a.length > 0)
      index = i;
}

Big Oh: ____________
Example 3

```java
public void sampleMethod3() {
    System.out.println("This is sample 3.");
    int b = 10;
    int c = 15;
    int d = b - c;
    b = b * c;
    d = d * b;
    c = d + d;
    b = c + d + b + d;
    System.out.println(b + " " + c + " " + d);
}
```

Big Oh: _______________

Example 4

```java
public void sampleMethod4(int n) {
    for(int i = 0; i <= n; i++)
        for(int j = 1; j <= n; j = j * 2)
            for(k = 0; k <= n/2; k++)
                System.out.println(i + j + k);
}
```

Big Oh: _______________
Linear Data Structures
Linear Data Structures

- Lists
  - ArrayList
  - LinkedList
- Stacks
- Queues
Applications of Stacks

◆ Balancing Parentheses
  □ Is this balanced?
    • ((()))(()())()
  □ Push when you see an open parenthesis.
  □ Pop when you see a closed one.
  □ If stack empties at end of string, it is balanced.
  □ Else, it is not.

◆ Postfix Expressions
  □ How to evaluate this:
    • 4 2 * 5 + 7 + 1.5 *
  □ Push when you see a number
  □ Pop two when you see an operator, operate and push result
Applications of Stacks ... 2

- Infix to Postfix Conversion
  - How to convert this:
    - Infix: \((4 \times 2) + 5 + 7\) * 1.5 to Postfix: \(4 \ 2 \ * \ 5 \ + \ 7 \ + \ 1.5 \ *
  - Infix: \(4 \times 2\)
  - Idea: Push operators on a stack and print operands
  - Infix: \(4 \ + \ 2 \ + \ 3\)
  - Idea: Before pushing operators, pop previous operator
  - Infix: \(4 \ + \ 2 \times 5 \ + \ 3\)
  - Idea: Before pushing operators, pop all operators of higher or equal precedence
  - Infix: \(4 \times (2 \ + \ 3) \ + \ 5\)
  - Idea: Push open parenthesis on stack. When closed parenthesis is encountered, pop stack until open parenthesis is popped
Applications of Queues

- Servers
  - Disk server, File Server, Print Server, ...
Hierarchical Data Structures
Trees were invented to ...

- Store hierarchical information
  - File system
  - Software Hierarchy
  - Administrative Hierarchy
  - Geographical Hierarchy
  - Decision trees
  - Parse trees
  - Family genealogy
  - Tree of life
  - ...

- To deal with inefficiencies of Linear Data Structures and to store dynamic information
  - To be explained later!
Decision Tree Model
for Car Mileage Prediction

Weight == heavy?

Yes
- High mileage

No
- Horsepower <= 86?
  - Yes
  - High mileage
  - No
  - Weight == heavy?
    - Yes
    - High mileage
    - No
    - Horsepower <= 86?
      - Yes
      - High mileage
      - No
      - Low mileage

Different Kinds of Trees
Tree

Def: Connected hierarchical structure without cycles

Def: A tree is an abstract mathematical object with
- Set of nodes, V, with special root node, r
- Set of directed edges, E, such that for each edge e in E, e = (u, v), where u is parent of v, or v is a child of u, and
- Every node has a unique parent except the root node, r.

Def: A binary tree is a tree where every node has at most two children

Def: A subtree of a tree, T = (V,E) is the tree rooted at some node from V.
Tree Terminology

- **Root**: node with no parent
- **Leaf and internal node**: node with no child and with at least one parent
- **Parent, Child**: each directed edge defines parent-child
- **Sibling**: a node that shares the same parent
- **Ancestor (Descendant)**: all nodes on path to root (leaf)
- **Left child, right child**: if children are ordered, then this terminology is relevant in binary trees
- **Full binary tree**: each node has 2 children or is a leaf
- **Complete binary tree**: tree filled level by level, left to right
Tree Terminology

- **Depth of node**: length of path from root to node
- **Height of tree (or subtree)**: length of longest path from root to leaf
- Like a linked list, a tree can be defined recursively
  - Subtrees rooted at nodes are also trees
Properties of Trees

- If $n$ is the number of nodes in a tree, then
  - Number of edges in a tree = $n - 1$
- Number of leaves in a full binary tree = 1 more than number of internal nodes
- Number of empty subtrees = $n + 1$
Implementing Binary Trees

Generalizing linked lists

- Instead of next/prev pointers, each node has 2 pointers
  - Left and right child; if needed, pointer to parent

Can be implemented using Arrays

- To be discussed later!
Tree Traversals

- **Inorder traversal**
  - Left subtree - Node - Right subtree
  - 6, 4, 2, 7, 5, 8, 1, 3

- **Preorder traversal**
  - Node - Left subtree - Right subtree
  - 1, 2, 4, 6, 5, 7, 8, 3

- **Postorder traversal**
  - Left subtree - Right subtree - Node
  - 6, 4, 7, 8, 5, 2, 3, 1

[https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/pix/non_rec_trav.bmp](https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/pix/non_rec_trav.bmp)
static void preorder(BinaryNode rt) {
    if (rt == null) return; // Empty subtree - do nothing
    visit(rt); // Process root of subtree
    preorder(rt.left()); // Process all nodes in left subtree
    preorder(rt.right()); // Process all nodes in right subtree
}
Tree Traversal Applications

- Expression Parse trees
  - Inorder traversal?
  - Postorder traversal?

- Printing directory structure
  - Figure 4.7 in Weiss book

https://people.eecs.berkeley.edu/~bh/ss-pics/parse0.jpg
Binary Search Trees

- Binary Tree where each node stores a value
- Value stored at node is **larger** than all values stored in nodes of **left** subtree
- Value stored at node is **smaller** than all values stored in nodes of **right** subtree