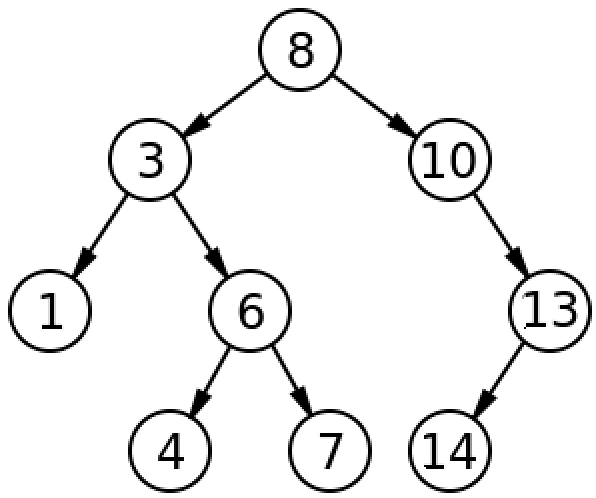
Data structures lab – week 4

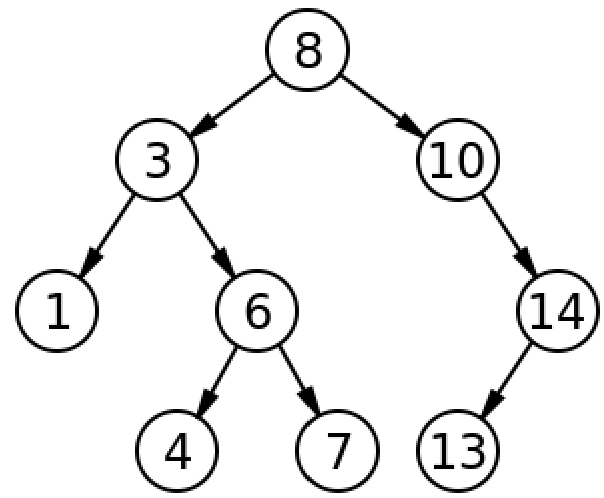
Welcome back!

Can you belive it's week 4 already?

Wake-up quiz

- Which of the following trees is a binary search tree?





- The one on the right is a BST.

Week 3 recap

- Hints for future success
  - More of that today
- From pseudo code to implementation code
- Trees in the forest
- Assignment 2 description
  - Revisited today.

Week 3 class evaluation

- To the slow/easy side – but interesting
- Selected comments (slightly edited):
  - ”Good amount of content covered today!”
  - ”... more & faster, please?”
  - ”Give us C++ code to generate the secret number”
    - Would defeat the purpose of the exercise.
  - ”do more on the assignments”
  - ”the wake-up quizzes were a neat touch.”
- Full survey results found online

Outline

- Last week
- Assignment 1 comments
  - More hints for future success
- Assignment 2
  - Searching for stuff

Hints for success

- Hint number 1: Read the assignment
- ”You should conform **exactly** to the input and output specification.”
  - ”Let me say that again: **conform exactly to the input and output specification**”
    - This is from the website.
- Many had extra stuff in there.

Hints for success

- Hint number 2: Look at your code
- Hint number 3: Comply with standards
- Hint number 4: Use large test cases
- Hint number 5: Use the terminal

Hints for success

- Hint number 6: Use IX and g++
- I compile with g++
- My compiler is very strict
  - Unlike MinGW's/Windows' version
    - Apparently?
- Ergo, compile with the same compiler as mine and do it on the IX server.

Compiling on IX

- Get a CS account
- Log onto the iMacs in Deschutes 100
- Compile your code from the command line
- Result:
  - Bigger chance that I can compile your code
  - Less likely that I will become slightly irritated.
    - I only ever get slightly irritated
      - Lucky you :-)

Compiling on the IX – from home

- Linux (what I do):
  - Transfer files via sftp
    - I use FileZilla
  - Open a terminal
  - ssh username@ix.cs.uoregon.edu
  - Do I need to tell you more?
- Mac:
  - The same

Compiling on IX – from home

- Windows (what I did):
  - Transfer files via sftp
    - FileZilla is also fine on Windows
  - Download PuTTY
    - Link on the website
  - Use PuTTY to ssh to IX
    - See previous slide.

Hints for success

- Hint number 7: Fear the NULL

```
struct linkedList {
    node * head;
    void add(node * x) {
        if (head == NULL)
            // Do something
    }
}
```

- Is this good?

Hints for success

- Hint number 7: Fear the NULL
- In Java:
  - Variables automatically initialized to null
  - null problems = NullPointerException
- In C++:
  - Variables NOT automatically initialized
  - NULL problems = Segmentation Fault
    - What the heck?

Hints for success

```
struct linkedList {
    node * head;
    linkedList() {
        head = NULL;     ← Good
    }
    void add(node * x) {
        if (head == NULL)
            // Do something
        }
    }
```

Hints for success

- Hint number 8: Use a debugger
  - GDB is a good choice
  - Eclipse uses GDB by default
    - As far as I remember
- Command-line GDB can be difficult
  - But it's very doable
  - Compile code with -g option
  - Commands you need for basic debugging:
    - Run, backtrace, step, list, print, break

Hints for success

- Hint number 9: Start earlier
- In the submission notes: ”This or that was ambiguous” or what ever
  - Could have been resolved with an email
    - This tells me: He/she started too late
    - This thought is in my mind the entire time while grading.
    - This is not beneficial to you
- Programming takes longer than essays
  - Especially debugging.

Wake-up quiz – BSTs

- We have seen that LinkedLists and BSTs have similar search times in worst case.
- What about insert time?
  - a) A LL has faster insert time than a BST
  - b) A BST has faster insert time than a LL
  - c) They have the same insert time.
- Correct answer is a
  - Is it a fair comparison?

Binary search trees

- Every node has at most 2 children.
- Every node consists of:
  - A *key*
  - A pointer to the left child, *left*
  - A pointer to the right child, *right*
  - A pointer to the parent, *p*
- Btw, how is this implemented in C++?

Binary search trees

```
struct BSTNode {
   int key;
   BSTNode * left;
   BSTNode * right;
   BSTNode * p;
}

struct BinarySearchTree {
   BSTNode * root;
}
```

Don't forget the constructors!

Binary search trees

- Every node *x* satisfies the ***binary-search-tree-property (bstp):***
  - For every node *y* in the left subtree of *x*:
    - *y.key* <= *x.key*
  - For every node *y* in the right subtree of *x*:
    - *y.key* >= *x.key*

Binary search trees

- What does the **bstp** give us?
  - O(h) insert operation
  - O(h) find operation
  - O(h) delete operation
- But *h* could be the number of nodes in the tree = slow.

BST versus LL

- Let's summarize

Data structure	Insert	Find	Delete
Linked List	O(1)	O(n)	O(1)
Binary Search Tree	O(h)	O(h)	O(h)

- Why would we ever use a BST?
  - If h = lg n, then it's pretty good
    - Requires balancing
      - Or random insertions
  - Assuming worst case, what else could we possibly want to do?

BST versus LL

- Because of the **bstp**, the tree is sorted!
- Inorder-Tree-Walk runs in O(n)
- For a Linked List... O(n^2)
- Is this significant in reality?
  - Let's try it!

BST versus LL – tested

- We want to test insertion
  - Both in worst and average case for BST.
- We want to test insertion + sorted printing
  - Both in worst and average case for BST.
- We hope we can see a difference
  - This is our hypothesis

BST versus LL – test recipe

1) Implement LL
2) Implement BST
3) Run tests
4) Look at results
5) Conclude

(1) Linked List testing

- Implement a Linked List
- Each node:
  - *next* pointer
  - *key* integer
- The List:
  - *head* pointer
  - *tail* pointer
  - *size,* for convenience

(1) Linked List testing

- Implement insert(*list*,*x*) in O(1) time.
  - Just insert *x* at the tail of the *list*.
- Implement printInOrder(*list*) in O(n^2) time.
  - For *i* = 0 to *list*.size
    - Search for minimum element that has not been printed
    - Print the element

(2) Binary search tree testing

- Implement a binary search tree
- Each node:
  - *left, right* and *p* pointer
  - *key* integer
- The BST:
  - *root* pointer

(2) Binary search tree testing

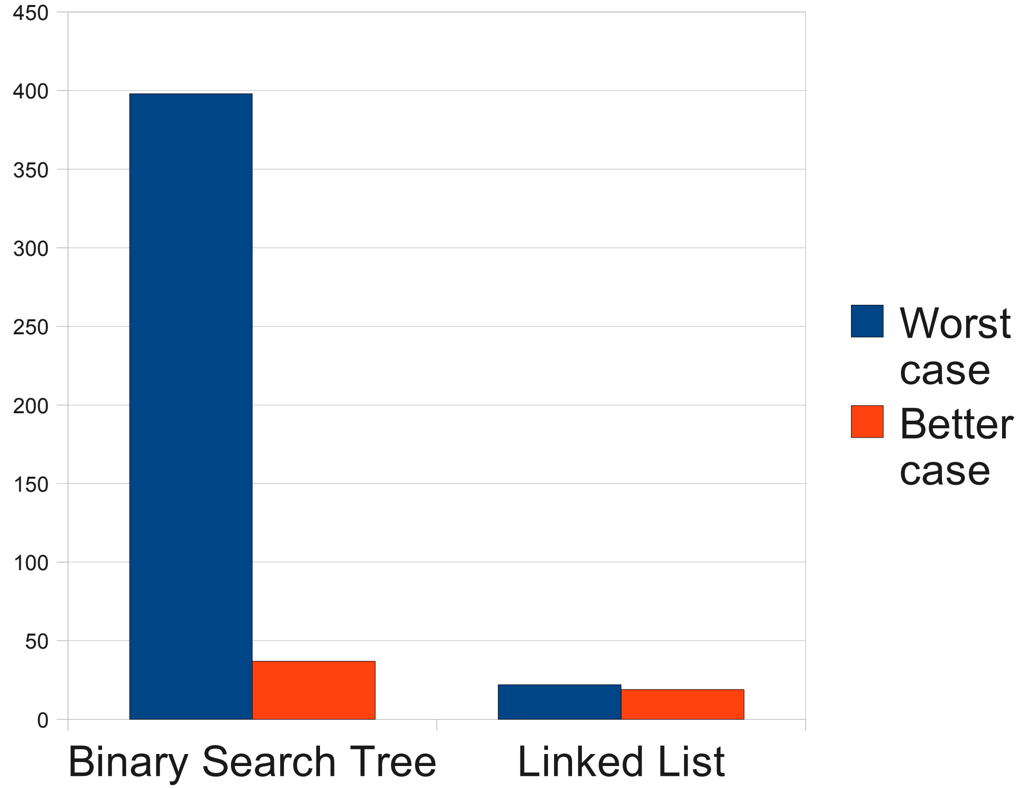
- Implement insert(*bst*,*x*) in O(h) time.
  - Copy almost exactly from Cormen
    - Tree-Insert, section 12.3
- Implement printInOrder(*bst*) in O(n) time.
  - Copy almost exactly from Cormen
    - Inorder-Tree-Walk, section 12.1

(3) Run tests

- time ./bst < testcase_slow > out
- time ./bst < testcase_better > out
- time ./LL < testcase_slow > out
- time ./LL < testcase_better > out
  - I write to file ”out” to reduce time to print to the console

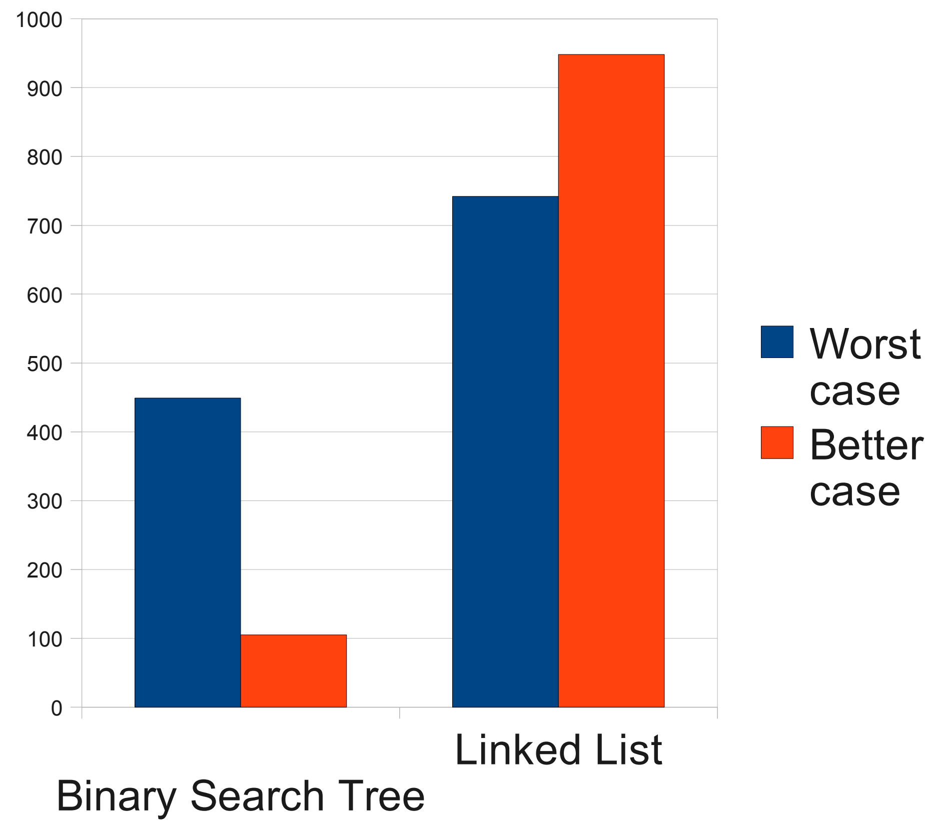
(4) Look at results

- Insertion only



(4) Look at results

- Insertion + sorted printing



(5) Conlusion

- BST is much faster than LL for printing in sorted order
  - Even in the worst case!
- BST is REALLY bad for insertion in the worst case
- Confirms our wakeup quiz from before.
- Congratulations, you've just seen your first $\Omega(n \lg n)$ sorting algorithm!

Searching for stuff

- Why did I show you all that?
- Does it look familiar?
- Similar to assignment 2
  - Comparison of find instead of sorted print
  - Worst/best case performance for BST

Wake-up quiz – assignment 2

- 100,000 find operations on a particular LL implementation takes 1 seconds.
- How much do we expect for 200,000?
  - a) 1.1 seconds
  - b) 2 seconds
  - c) 4 seconds
  - d) 8 seconds
- Correct answer is b.

Assignment 2

READ the assignment.
When you've read it, read it again.

Assignment 2 – task

- We want to test *find*
  - Both in worst and average case for BST.
  - Both for LL and BST
- We want to analyze the *find* running time for BST and LL
- We want to compare the *find* running time for BST and LL
- We hope we can see a difference
  - This is our hypothesis

Assignment 2 – recipe

1) Implement LL
2) Implement BST
3) Run tests
4) Look at results
5) Conclude

Assignment 2

- Implement a binary search tree data structure.
  - Support insert and search
  - Do not bother about deletion
  - Do not balance the tree!
- Expand your linked list from A1 to include searching.
- Compare running time for search with BST and LL.

Assignment 2 – testing

- I have supplied 8 testcases
- ”Slow”, simulates worst case for BST
  - 10 thousand node insertions
  - 50, 100, 150 or 200 find operations
  - Named 10k_50k_slow, 10k_100k_slow etc.
- ”Better”, simulates random insertion
  - Same number of and insertions and finds
  - Named 10k_50k_better, 10k_100k_better etc.

Assignment 2 – evaluation

- To be able to evaluate your solutions, your programs will have to produce some output
- For each find operation, print the number of nodes you looked at to get there.
  - Including the target node itself
  - By the way, this can also be used as a measure of speed
  - Would probably be a good idea to return this value from the find operation.
- Well explained on the website

Assignment 2 – questions

- Should our BST have any functions for balancing itself?
  - No, not this time.
    - It is even IMPORTANT that you do not!
- How would we do that?
  - Don't
    - But go to Chris' lecture tomorrow
      - He will talk about AVL trees. Something not found in the book!

Assignment 2 – questions

- ”How should we be handling inputs less than 1?”
  - Expect testcases to be similar to the ones I have supplied
    - Ergo: input size > 1
- ”Is it OK to measure the performance of data structures in terms of milliseconds instead of seconds?”
  - Yes

Assignment 2 – questions

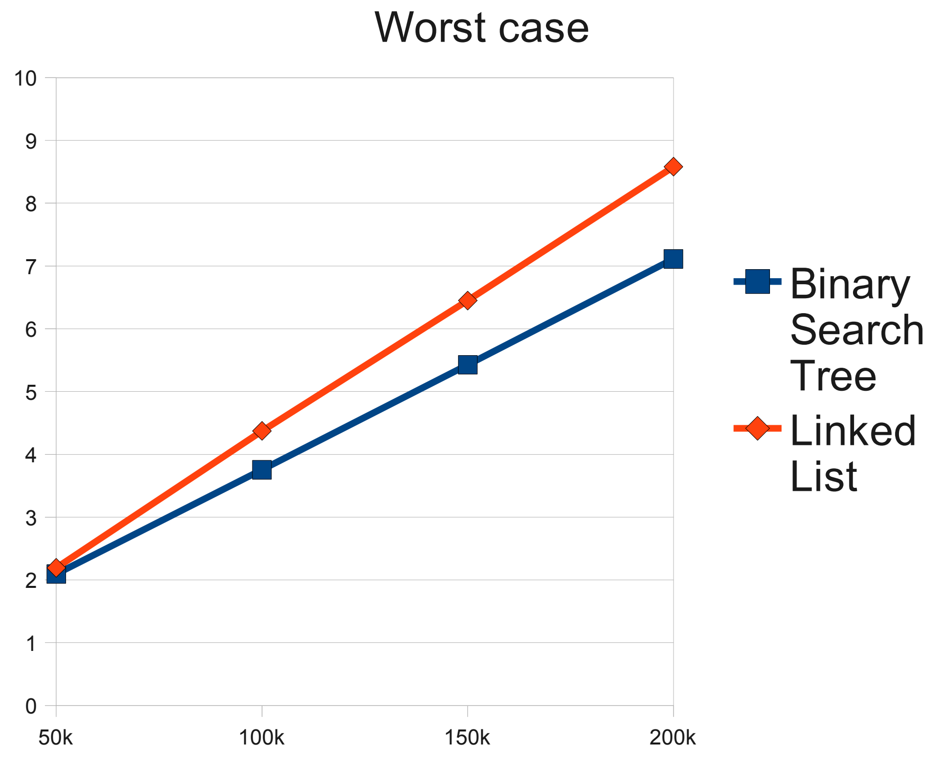
- ”Do we need to submit circle list and bst with their respective main files?”
  - The assignment description says exactly what to turn in.
    - A LinkedList implementation
    - A BST implementation
    - A small discussion
  - Elaborated on the website.

Assignment 2 – questions

- ”Since I have a computer running on 4 cores, even the worst case with linked-list gives me 0 seconds”
  - This is of course possible
  - I ”only” have a Core 2 Duo with 2GHz.
  - I have uploaded a tool so you can create extra testcases
    - Check the website

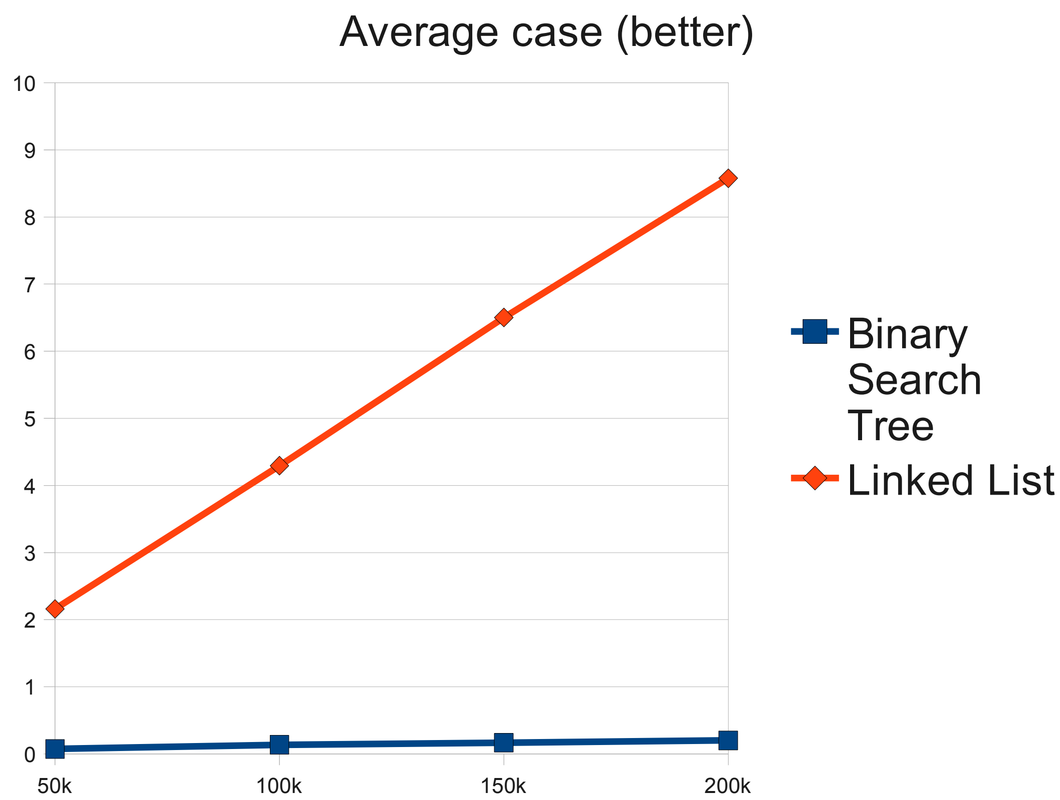
Assignment 2

- My results



Assignment 2

- My results



Thank you

Questions?