Dion Niazi 03/01/2017 CS 2150 Lab Section: 001 (9:30 – 10:45)

Testfile4.txt

```
// Dion Niazi dn3gy 01 03 2017 testfile4.txt
true pack test selection
blue-eyed ugly alluring thoughtless
left malicious warlike kill van decision frail whisper oval ambiguous
command arrest condemned scissors annoying best shaky mass thinkable
collect like cows drop press whine close
obey parallel inform appear alluring dark fowl
song homely elegant
print recognize thankful need
prepare
whip
level back
desert spicy travel plant
board
birth
crime
burst scarecrow table
fail
sound
bath last
overconfident fortunate
decisive
bag
found imagine
phone
famous
turkey fixed fair touch action hot pear earn thick
record
manage rhythm birthday
memorise
dear
loutish compete brave
effect power smoke
star clammy
observe
scene
educate downtown jam
```

man haunt eight
slope
rings
smile
join
known wool
care
maddening gaudy
precious
ill-informed savory
observe
pail arithmetic
baby cat dog
elephant
frog
great height exia
anorexic
anorexics
anorthic
another
anoxia
anoxic
ansi
answer
answerable answerably answered answering answers
ant
antacid antacids
antagonise
antagonized antagonises
antagonising
antagonism antagonisms
antagonist
antagonistic
antagonistically antagonists
antagonize antagonized
antagonizes antagonizing Antarctic Antarctica ante anteater
anteaters antecedence antecedent
antecedents
antechamber antechambers
anted

```
antedate antedated
antedates
antedating
antediluvian
antediluvians
anteing antelope antelopes antemeridian antenatal antenna antennae antennas
antenuptial antepenult antepenultimate antepenults anterior anteroom
anterooms antes anthem anthems anther anthers anthill anthills anthologies
anthologise anthologized anthologises anthologizing anthologist anthologists
anthologize anthologized anthologizes anthologizing anthology Anthony
anthracite anthrax anthropocentric anthropoid anthropoids anthropological
anthropologically anthropologist anthropologists anthropology
anthropometries anthropometry anthropomorphic anthropomorphism
anthropomorphous anti antibacterial antiballistic Antibes antibiosis
antibiotic antibiotics antibodies antibody antic antichrist anticipant anticipate
anticipated anticipates anticipating anticipation anticipator anticipators
anticipatory anticlerical anticlericalism anticlimactic anticlimactically
anticlimax anticlimaxes anticline anticlines anticlockwise anticoagulant
anticoagulants anticoagulation anticorrosive antics anticyclone anticyclones
antidepressant antidepressants antidote antidotes antifreeze antigen antigenic
antigens antigraphy Antigua antigun antihero antiheroes antihistamine
antihistamines antiknock antiknocks Antilles antilogarithm antilogarithms
antimacassar antimacassars antimony antineutrino antineutrinos antineutron
antineutrons antinomies antinomy antioxidant antioxidants antiparticle
antiparticles antipasti antipasto antipastos antipathetic antipathies
```

The test file works in proving that AVL trees are far more superior to binary search trees because the average node length is used to compare their speed and in both data structures it varied by a lot. From the information provided from the program, AVL trees average node length was about a 6.5, whereas for binary trees it was about 56.5, which is a huge difference. When using examples where you just add random nodes doesn't really make a huge difference, but the secret is when it's a pattern, like always greater than or less than the specific root node. This makes the BST linear more like a list and it becomes really unbalanced, but AVL trees are always balanced so you wouldn’t fall for a list like structure making retrieval time much faster. That’s why this list at first just had random words that could possibly affect the average node depth, but soon it was realized that such a list like pattern creates a huge time difference and thus a lot of ‘a’ words were used.

Testfile4.txt Results

307 words in this text

BST:

Left links followed = 0

Right links followed = 0

Total number of nodes = 307

Avg. node depth = 56.5212

AVL Tree:

Left links followed = 0

Right links followed = 0

Total number of nodes = 307

Single Rotations = 98

Double Rotations = 125

Avg. node depth = 6.56026

Testfile1.txt Results

```
19 words in this text


BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Avg. node depth = 3.15789

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Single Rotations = 2
Double Rotations = 2
     Avg. node depth = 2.26316
```

Testfile2.txt Results

16 words in this text

BST:

Left links followed = 0

Right links followed = 0

Total number of nodes = 16

Avg. node depth = 6.0625

AVL Tree:

Left links followed = 0

Right links followed = 0

Total number of nodes = 16

Single Rotations = 9

Double Rotations = 0

Avg. node depth = 2.5

Testfile3.txt

```
13 words in this text


BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Avg. node depth = 3.23077

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Single Rotations = 1
Double Rotations = 2
Avg. node depth = 2.23077
```

Testfile5.txt (own file where it's a balanced BST already) Results

```
9 words in this text

BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 9
Avg. node depth = 1.77778

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 9
Single Rotations = 0
Double Rotations = 0
Avg. node depth = 1.77778
```

AVL trees are usually the better choice over binary search trees because of the multiple characteristics it has. Since AVL trees are always balanced, we are able to avoid list like structures, which will then lead to big theta n instead of getting the desired big theta log n. The situation in which AVL trees are much better than BST trees is when the placing of nodes in the same exact order like in a list like structure because it gets balanced and look up time is guaranteed to be log n instead of leading to n. AVL trees prevent the worst case in BST from happening.

BSTs are better than AVL trees in that it's less complicated to implement than a BST. Not all the time the best performance possible is the better option when using a particular data structure over another. When knowing that you don't need to worry about balancing, then a BST is the way to go.

The costs of AVL trees when compared to BST when balancing doesn't matter are that AVL trees would consume more memory. Also AVL trees would have to always balance it so it takes some time to balance whereas for BSTs the node is just inserted in. The balance factor in AVL trees has to be kept up with so it takes in a little more memory and that little memory could make a difference.