a) Prove, directly from the definition of o-notation, that $2015n + 1394 \in o(n \log n)$.

Let $c > 0$ be given we need to show that $\exists\ n > n_0$ such that

$2015n + 1394 \leq cn \log n$

$2015n + 1394 \leq 2015n + 1394n$ (for $n \geq 1$)

$= 3409n$

$3409n \leq cn \log n$

$\frac{3409}{c} \leq \log n$

$n \geq 2^{\frac{3409}{c}}$

Pick $n_0 = \max(1395, 2^{\frac{3409}{c}} + 1)$

$\therefore$ Pick $n_0 = \max\{1, 2^{\frac{3409}{c}} + 1\}$

$2015n + 1394 < cn \log n$ for all $c > 0$, when $n >$ for some positive integer n_0

$2015n + 1394 < 2016n$ $\quad n > 1394$

$2016n < cn \log n \iff 2016 < c \log n$ $\quad n >$

(b) Let $f(n) = (n^{3/2})/240$ and $g(n) = 240\, n(\log n)^2$. Compare the growth rates of $f(n)$ and $g(n)$ us the **most appropriate** asymptotic notation. Prove the relationship using l'Hospital's Rule.

$f(n) = \frac{n^{\frac{3}{2}}}{240}$

$g(n) = 240\, n (\log n)^2$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^{\frac{3}{2}}}{240}}{240\, n(\log n)^2}$ we ignore the constants

$= \lim_{n\to\infty} \frac{n^{\frac{3}{2}}}{n(\log n)^2}$

$= \lim_{n\to\infty} \frac{n^{\frac{3}{2}}}{(\log n)^2}$ Apply L'Hopital's Rule

$= \lim_{n\to\infty} \frac{\frac{1}{2} n^{-\frac{1}{2}}}{2(\log n)\frac{1}{n}\ln 2}$ Ignore the constants again

$= \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n \log n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} \cdot \frac{1}{n \log n} = 0$

$\therefore f(n) \neq \Omega(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^{\frac{3}{2}}}{240}}{240 n(\log n)^2} = \left(\frac{1}{240}\right)^2 \lim_{n\to\infty} \frac{n^{\frac{3}{2}}}{n(\log n)^2} = \left(\frac{1}{240}\right)^2 \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{(\log n)^2}$

$= \left(\frac{1}{240}\right)^2 \lim_{n\to\infty} \frac{\frac{1}{2} n^{-\frac{1}{2}}}{2 \log n \cdot \frac{1}{n} \ln 2} = \left(\frac{1}{240}\right)^2 \times \frac{1}{4 \log e \ln 2} \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{\log n}$

$= C \lim_{n\to\infty} \frac{\frac{1}{2} n^{-\frac{1}{2}}}{\frac{1}{n} \log e} = \infty$ $\quad \therefore f(n) \neq w(g(n))$

(c) Using Θ-notation, give the runtime of the following pseudo-code as a function of n. Assume Li
2 takes $\Theta(i)$ time.

answer box $\Theta(n^2 \log n)$

```
cute-function(A, n)
1: for i ← 1 to n do
2:     j ← n^i          — Θ(i)
3:     while j ≥ 1 do
4:         j ← j/2.
5:     end while
6: end for
```

For rough work.

line 4 - $\Theta(i)$

line 3 – line 5

After t iteration

$j = n^i$

$j' = \frac{j}{2^t} = \frac{n^i}{2^t}$

$j \leq 1$

$\frac{n^i}{2^t} \leq 1 \rightarrow n^i \leq 2^t \Rightarrow t \geq \log n^i = i \log n$ times $\Theta(i \log n)$

Total cost

$\sum_{i=1}^{n} (\Theta(i) + \Theta(i \log n)) = \sum_{i=1}^{n} \Theta(i) + \sum_{i=1}^{n} (i \log n)$

$= \Theta(\frac{(1+n)n}{2}) + \Theta(\log n + 2\log n + \dots n \log n)$

$= \Theta(\frac{(n+1)n}{2}) + \Theta(\frac{n+1}{2} \log n)$

$= \Theta(n^2) + \Theta(n^2 \log n)$

$= \Theta(n^2 \log n)$

(d) A clever student writes a code for Quick-Sort in which the pivot is always selected as the median. However, her algorithm for finding the median of n items takes $\Theta(n^2)$. Write down the recursion for the time complexity of this implementation of quick sort and solve it. You should express the time complexity using Θ notation.

$T(n) = \begin{cases} T(\lfloor \frac{n-1}{2} \rfloor) + T(\lceil \frac{n-1}{2} \rceil) \\ d \end{cases}$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + c(\frac{n}{2})^2$

$T(\frac{n}{4}) = 2T(\frac{n}{4}) + c(\frac{n}{4})^2$

$T(n) = 2T(\frac{n}{2}) + cn^2$

$= 2(2T(\frac{n}{4}) + c(\frac{n}{2})^2) + cn^2$

$= 4T(\frac{n}{4}) + cn^2 + \frac{2cn^2}{4}$

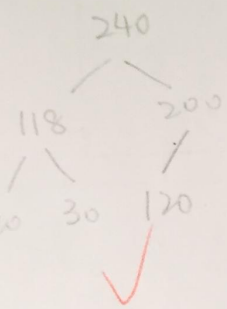
$= 8T(\frac{n}{4}) + cn^2 + \frac{2cn^2}{4} + \frac{4cn^2}{16}$

2. 17/12 Heaps

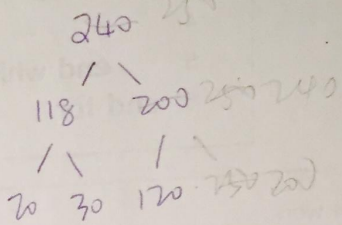
Suppose we have a max-heap stored in the following array:

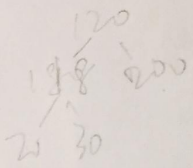
$$A = [240, 118, 200, 20, 30, 120]$$

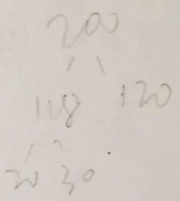
(a) Draw the heap that is stored in A (as a tree).



2







(b) Write the new array that results after inserting 250 into the above heap.

3

$A_{new} = [250, 118, 240, 20, 30, 120, 200]$ [250, 118, 240, 20, 30, 120, 200]

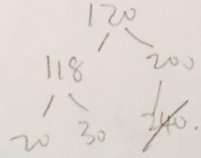
(c) Write the new array that results after calling *delete-max* on the **original** heap.

3

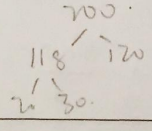
$A_c = [200, 118, 120, 20, 30]$ [200, 118, 120, 20, 30]

For rough work.

swap max with the last element



bubble down.



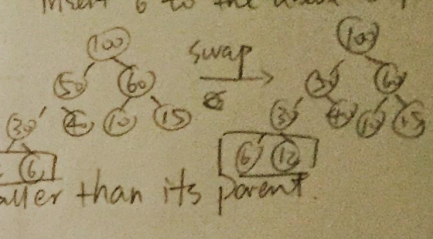
© 2015 University of Waterloo

(d) Consider a max-heap of n distinct keys with the additional property that the key of the left child o a node is always smaller than its right child (an example is $[100, 50, 60, 30, 40, 10, 15, 12]$). Explain how to modify *bubble-up* function so that operation *insert* takes $O(\log n)$ time while the above property is maintained.

4

Before comparing the value with its parent, compare it with its sibling
If insert an element with the index $2i+2$ (right child of a node)
compare it with $2i+1$ (right child of a node)
if $A[2i+2] < A[2i+1]$.
swap $(A[2i+2], A[2i+1])$

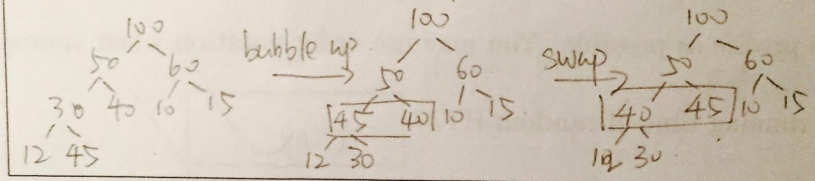
i.e



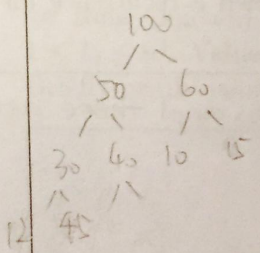
Since if ~~the~~ $A[2i+2] < A[2i+1]$. $A[2i+2]$ is smaller than its parent.

After bubble-up one level, if it stays at the left, comparing it with its right sibling. If it is larger than the right sibling swap.

i.e insert 45 to the above example (original)



For rough work.



12

insert 6 -> swap 12-6

3. /10 **Randomized Algorithms**

We say an array of n distinct integers is *head-tail-justified* if the smallest item is at index 0 and the largest item is at index $n-1$ (assuming indices start from 0), e.g., for $n = 6$ a head-tail-justified array might look like $2, 9, 5, 10, 6, 12$. Given an arbitrary array of size n, we would like to rearrange its elements to form a head-tail-justified array. The following randomized algorithm creates a random permutation of the input array and checks if it is head-tail-justified. Recall that shuffle(A, n) returns a random permutation of the input array A of size n. Assume the time-complexity of shuffle(A, n) is $\Theta(n)$.

```
random-HTJ(A, n)
1: B ← shuffle(A, n)        Θ(n)
2: if B is head-tail-justified then
3:     return B
4: else
5:     return randomHTJ(A, n)
6: end if
```

In your answers below, be as precise as possible. You may use order notation when appropriate.

(a) What is the **best-case** running time of random-HTJ? $\Theta(n)$

(b) What is the **worst-case** running time of random-HTJ? $\Theta(\infty)$ the program may not terminate

(c) Let $T(n)$ be the expected running time of random-HTJ. Write down a recurrence for $T(n)$ and then solve it.

For an array of size n, there are $n!$ permutations, and there are $(n-2)!$ permutations to be head-tail-justified.

$$T(n) = cn + \underbrace{\frac{(n-2)!}{n!}}_{\text{sorted}} \cdot d + \underbrace{\frac{n!-(n-2)!}{n!}}_{\text{unsorted}} T(n)$$

$$= cn + \frac{1}{n(n-1)} d + \left(1 - \frac{1}{n(n-1)}\right) T(n)$$

$$\left(1 - 1 + \frac{1}{n(n-1)}\right) T(n) = cn + \frac{1}{n(n-1)} d$$

$$T(n) = \left(cn + \frac{1}{n(n-1)} d\right) n(n-1) \Rightarrow T(n) = cn^2(n-1) + d + O(n^3)$$

© 2015 University of Waterloo

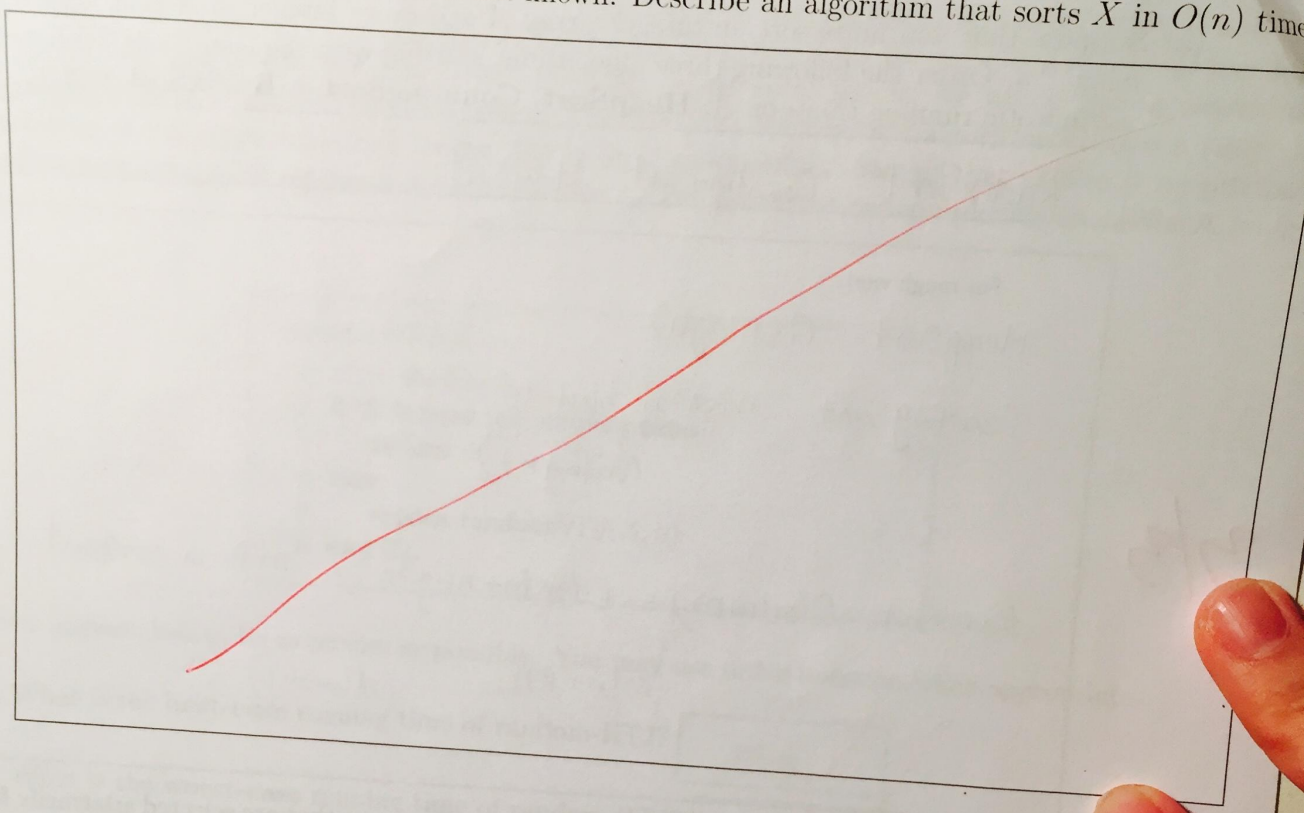
kage.

October 5, 2015

time, or for a sub

/12 Binary

(c) Suppose that you must sort a set $X = x_1, \ldots, x_n$ of integers. You know that there exists a cons
c such that X contains $n - \frac{n}{\log n}$ integers that are smaller than n^c. Remaining elements of X can t
arbitrarily large. The constant c is not known. Describe an algorithm that sorts X in $O(n)$ time

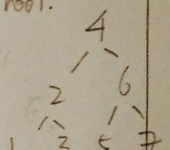


here exists a co
elements of X ca
sorts X in $O(n)$ time

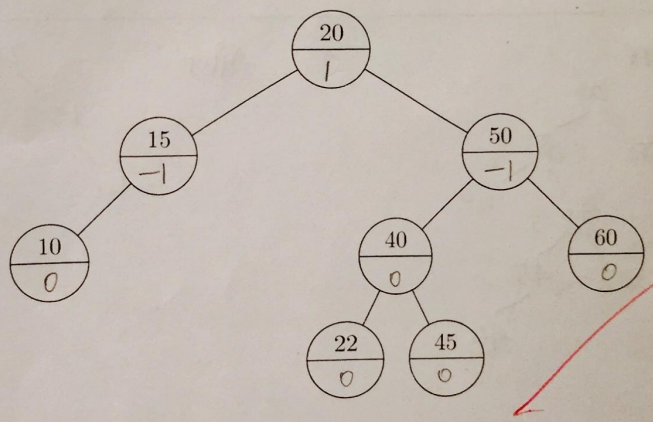
/12 Binary Search Trees and AVL Trees

(a) Suppose that we want to insert integers $1, 2, 3, 4, 5, 6, 7$ into an initially empty AVL tree T_A. Elements can be inserted into T_A in an arbitrary order. In what order should we insert these elements so that the total number of rotations is minimized?

Insert the median at the root, in this case 4.
divide the rest into 2 groups, smaller than root and larger than the root.
keep finding the median of a group with elements less than the root (1, 2, 3)
insert to the root as left-subtree
keep finding the median of a group with elements larger than the root (5, 6, 7)
insert to the root as right-subtree
If there are ≤ 2 elements left, put the value smaller than the node at left, larger than the node at right

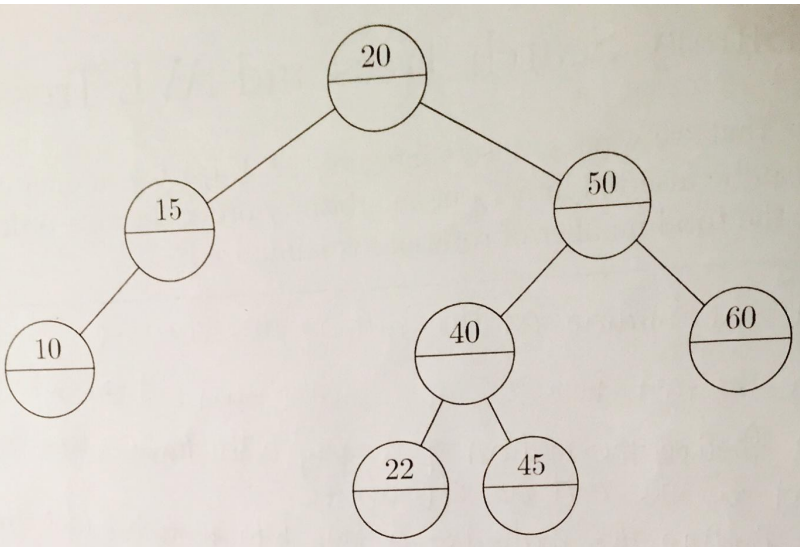


(b) Consider the following AVL tree T:

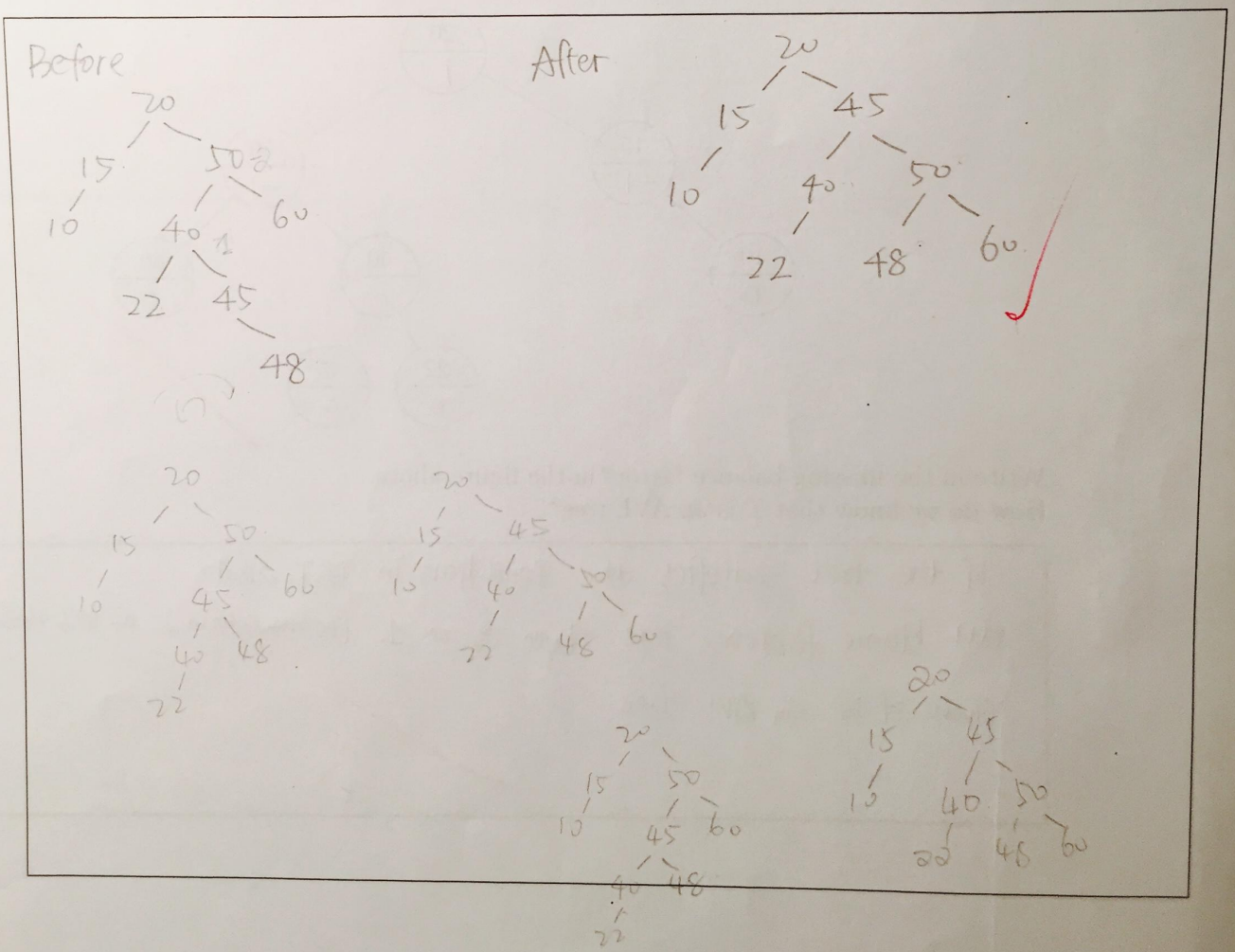


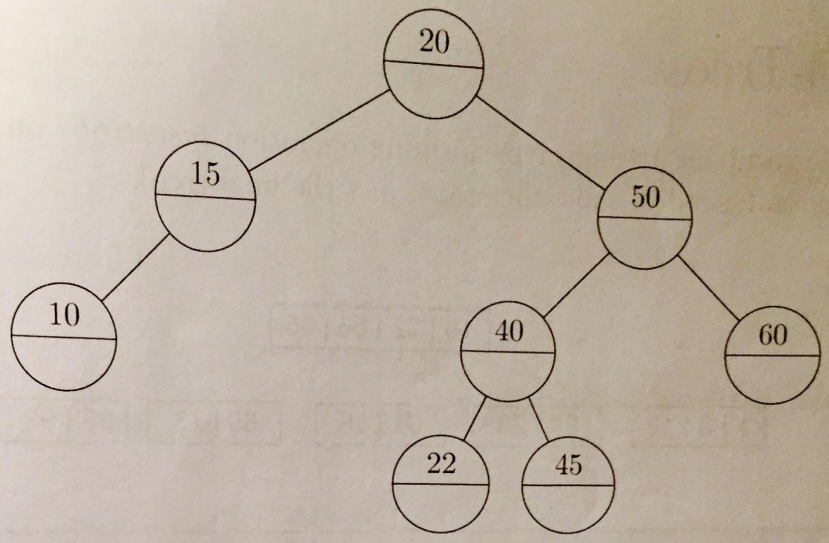
Write in the missing balance factors in the figure above.
How do we know that T is an AVL tree?

If the tree satisfies the conditions in BST, and
all blance factors are -1, or 0, or 1 (no more than 2, no less than -2)
Then it is an AVL tree.



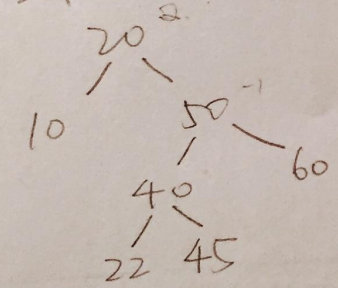
(c) Perform operation *insert*(48) on T (redrawn above).
Draw the tree before and after each rotation performed (no need to show balance factors). There is no need to show the two rotations of a double-rotation if a double rotation is used (just draw the tree before and after). There is no need to draw the original tree.



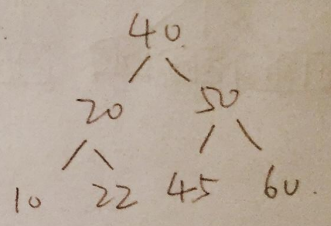


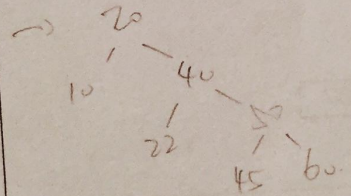
(d) On the original tree (redrawn above), perform operation *delete*(15) on T.
Draw the tree before and after each rotation performed (no need to show balance factors).
There is no need to show the two rotations of a double-rotation (just draw the tree before and after

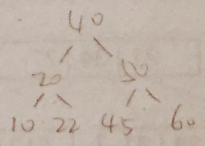
Before



After

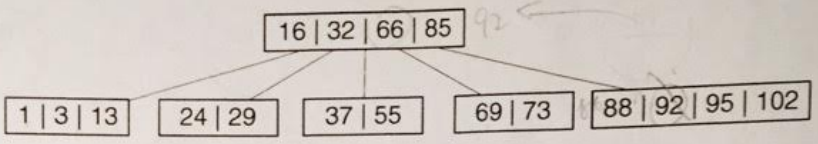


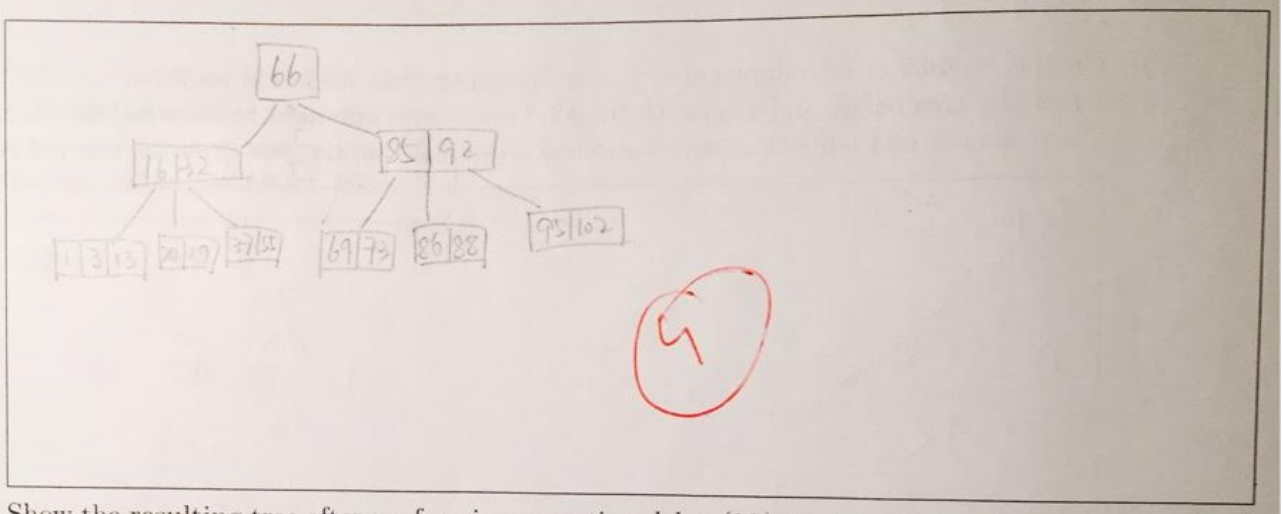




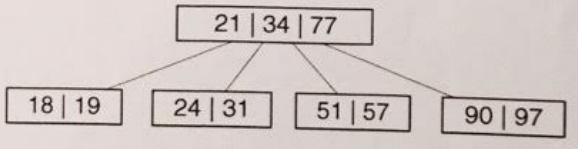
6. /8 B-Trees

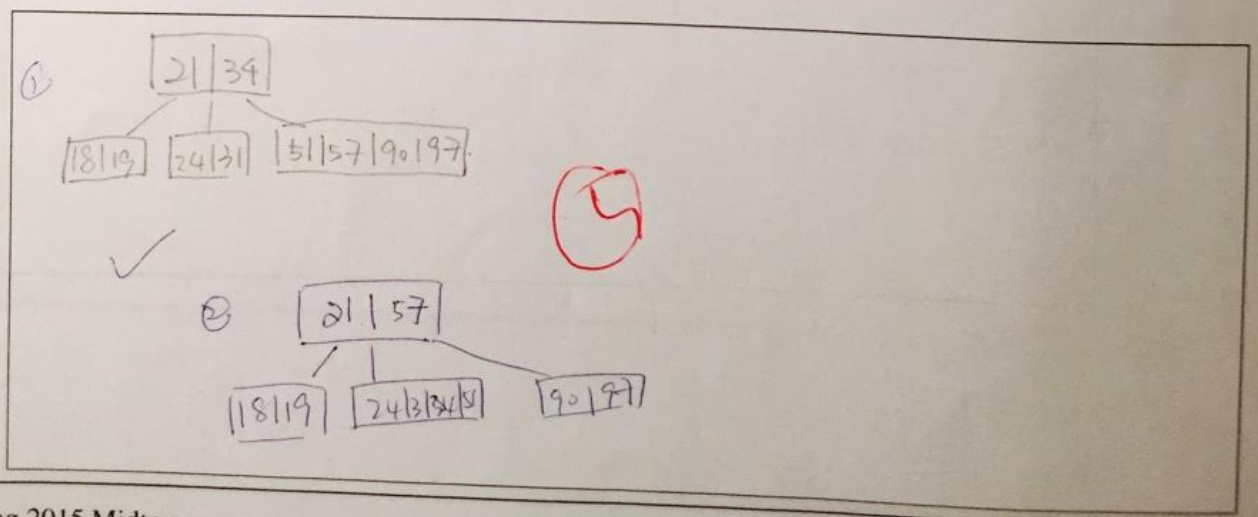
(a) Show the resulting tree after performing operation *insert(86)* on the following B-tree of max-size 4 (you do not need to show the steps; just the final tree):





(b) Show the resulting tree after performing operation *delete(77)* on the following B-tree of min-size 2 (you do not need to show the steps; just the final tree):





© 2015 University of Waterloo

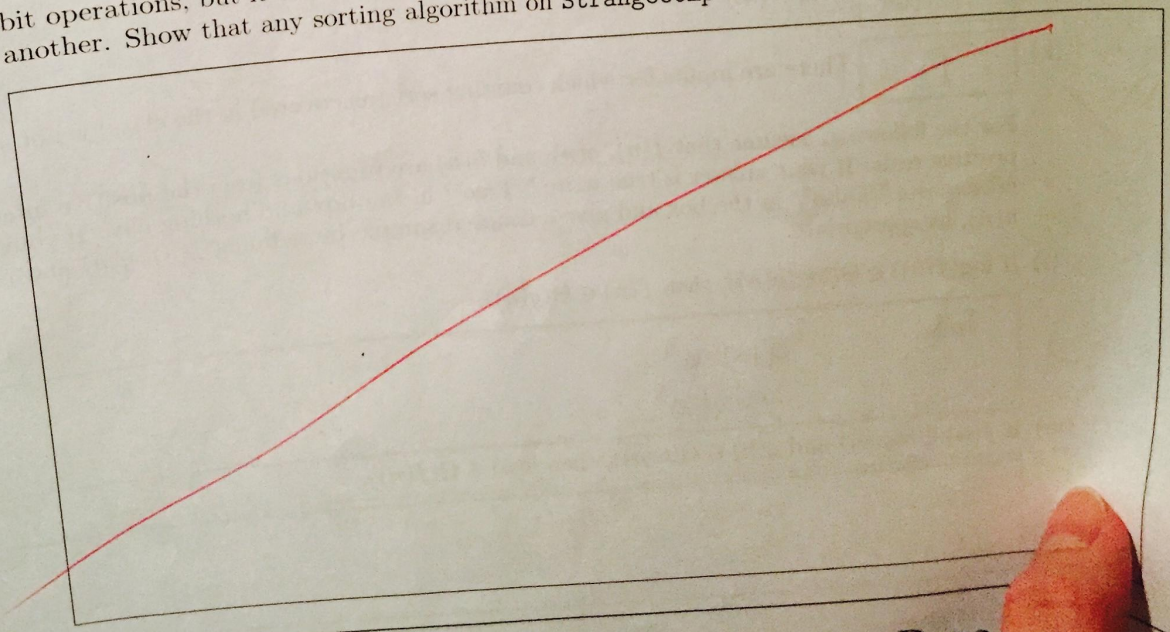
/4+4 2-3 Trees and Lower Bounds

(a) Show that it is not possible to insert all elements of an unsorted array A into a 2-3 tree in $o(n \log n)$ time. We denote by n the size of A.

Give an ordered array of keys, consider the following algorithm to sort them:

- Use an algorithm to create a 2-3 tree out of the array
- Traverse through the 2-3 tree using an in-order walk to retrieve a sorted array. — $O(n)$

(b) **[Bonus]** An inventor Dr. Strange constructed a computer `StrangeComp` that can compare b numbers in constant time. That is, it can read b numbers $x_1, \ldots, x_b$ from memory and sort them in increasing order $x_{i_1} < x_{i_2} < \ldots < x_{i_b}$. `StrangeComp` does not support arithmetic and bit operations, but it can copy b array elements in $O(1)$ time from one location in the memory to another. Show that any sorting algorithm on `StrangeComp` takes $\Omega((n/b) \log_b n)$ time.



8. /14 True/False

Determine whether each of the statements is true or false. Write "True" or "False" in the box. You will **not** lose extra marks for incorrect answers.

(a) False — If a problem has a lower bound of $\Omega(n^2)$, then no algorithm exists that solves it in $O(n^2)$.

(b) True — Assuming unique keys, a tree of size $n \geq 3$ cannot be a heap and a binary search tree at the same time.

(c) True — Randomization helps to place n items in an initially empty Binary Search Tree in $O(n \log n)$ expected time.

(d) False — A relaxed AVL tree which allows balance factors in $\{-2, -1, 0, 1, 2\}$ has height $O(\log n)$.

(e) False — Finding the minimum item when doing selection (i.e., $k = 1$) with quick-select-1 (when pivot is chosen to be the first element) takes time $\Theta(\log n)$ in the worst case.

(f) True — An array can be heapified (to form a max-heap) by sorting it in decreasing order.

(g) False — In the worst case, the number of rotations required after deleting a KVP from an AVL tree is 1.

(h) True — Time complexity of Merge-Sort is still $\Theta(n \log n)$ if the algorithm is changed to recurse on a small array of size $\lfloor n/4 \rfloor$ and a large array of size $n - \lfloor n/4 \rfloor$ (and then merge the sorted sub-arrays).

(i) True — $2.01^n \in O(2^n)$

(j) True — $2^n \in O(n!)$

(k) False — There are inputs for which counting sort (as covered in the class) is not stable.

For the following, assume that $f(n)$, $g(n)$, and $h(n)$ are functions from the positive integers to the positive reals. If your answer is true write "'True"' in the box and nothing else. If your answer is false, write "'False"' in the box and give a counterexample by defining $f(n)$, $g(n)$ and possibly $h(n)$, as appropriate.

(l) If $\log(f(n)) \in \Theta(\log(g(n))$, then $f(n) \in \Theta(g(n))$.

False $f(n) = n^3$ $g(n) = n^5$

(m) If $f(n) \in o(g(n))$ and $g(n) \in O(h(n))$, then $h(n) \in \Omega(f(n))$.

True $f(n) = n$ $g(n) = n^2$ $h(n) = n^3$

© 2015 University of Waterloo

/10 **Bonus** The *multiple selection problem* is a generalization of the selection problem. We are given an unsorted array A of size n and d indexes $i_1 < i_2 < \ldots < i_d$. We must find d numbers formed by the i_k-th smallest elements in A for $k = 1, 2, \ldots, d$. Describe an algorithm that solves the multiple selection problem in $O(n \log d)$ time.