1. /12 Asymptotic Analysis

For each of the following assumptions, indicate which of the statements W–Z must be true by writing the letters of any **true** statements in the box provided. Note that more than one statement might be true. If none of the statements are true, write "none" in the box.

(a) $f(n)$ and $g(n)$ are positive functions on the natural numbers.

W, X, Y

W. $f(n) \in O(\max\{f(n), g(n)\})$ ✓

X. $f(n) + g(n) \in \Theta(\max\{f(n), g(n)\})$ ✓

Y. $f(n) + g(n) \in \Omega(g(n))$

Z. $f(n) - g(n) \in \Theta(\min\{f(n), g(n)\})$ ✗

(b) $f(n) \in \Omega(n^3)$ and $g(n) \in \omega(n^2)$

Y, Z

W. $f(n) \in \Omega(g(n))$

X. $g(n) \in O(f(n))$

Y. $f(n) \in \Omega(n^2)$ ✓

Z. $g(n) \in \Omega(n^2)$ ✓

(c) The average-case cost of algorithm A is $\Theta(n)$.

X, Z

W. The worst-case cost of A is $O(n)$

X. The best-case cost of A is $O(n)$

Y. The best-case cost of A is $\Omega(n)$

Z. The worst-case cost of A is $\Omega(n)$

(d) Algorithm A has worst-case cost $\Theta(n\sqrt{n})$ and B has worst-case cost $\Theta(n(\log n)^5)$

W [redacted] Y

W. A is always faster than B

X. A is always slower than B

Y. For infinitely many problem instances, A is faster than B

Z. For infinitely many problem instances, A is slower than B

$\frac{n\sqrt{n}}{(\log n)^5}$ $\frac{n^{0.5}}{10(\log n)^4}$ $\frac{n^{\frac{1}{2}}}{40(\log n)^3}$ $\frac{n^{\frac{1}{2}}}{120(\log n)^2}$ $\frac{n^{\frac{1}{2}}}{240\log n}$ $\frac{n^{\frac{1}{2}}}{\infty}$

© 2015 University of Waterloo

2. /9 Short Answer

(a) Using Θ-notation, give the runtime of the following pseudo-code as a function of n.

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

```
1.  c = 0
2.  for i from 1 → n^2 do
3.      if i is even then
4.          for j from 1 → ⌈(log i)^2⌉ do
5.              c = c + 3
6.          end for
7.      else if i is odd and i < 100 do
8.          for j from 1 → n^5 do
9.              c = c + 3
10.         end for
11.     else
12.         c = c + 1
13.     end if
14. end for
```

(b) Complete the definition of o-notation, i.e., $f(n) \in o(g(n))$ if ...

$f(n) \in o(g(n))$ if $\forall c > 0, \exists n_0 > 0$ s.t $0 \le c f(n) \le g(n)$ for all $n \ge n_0$.

should be <

c is multiplied with g(n)

(c) Prove from the definition of o-notation that $1/n^2 \in o(1/n)$.

Let $f(x) = \frac{1}{n^2}$, $g(x) = \frac{1}{n}$

$\lim_{n\to\infty} \frac{f(x)}{g(x)} = \lim_{n\to\infty} \frac{\frac{1}{n^2}}{\frac{1}{n}} = \lim_{n\to\infty} \frac{1}{n} = 0$

$\therefore f(x) \in o(g(x))$

$\therefore \frac{1}{n^2} \in o(\frac{1}{n})$ by L'Hôpital's rule

$\because \forall n^2 \in o(1/2)$

$\therefore 0 \le c(\frac{1}{n^2}) \le \frac{1}{n}$

$\therefore c(\frac{1}{n^2}) \le \frac{1}{n}$, since $n > 0$

$c \frac{1}{n} \le 1$, since $c > 0$

$\frac{1}{n} \le \frac{1}{c}$

$\rightarrow n \ge c$

Therefore for all $c > 0$, $\exists n_0 = c + 1$, s.t $0 \le c f(n) \le g(n)$

$0 \le c(\frac{1}{n^2}) \le o(\frac{1}{n})$

for all $n \ge n_0$, where $n_0 = c + 1$

so for all $n \ge c + 1$.

from definition

© 2015 University of Waterloo

Q3 Score: **3.0**

3. /9 Sorting Algorithms

Let A be an integer array of size n, and suppose that each integer in A is at least 0 and at most $n^2 - 1$.

(a) What is the worst-case cost of running **CountingSort** on A?
Give your answer using Θ-notation, in terms of n.

$\Theta(n)$

(b) What is the worst-case cost of running **HeapSort** on A?
Give your answer using Θ-notation, in terms of n.

$\Theta(n \log n)$

(c) Describe how to sort A in $O(n)$ time.

Use counting sort.

(d) What is the auxilliary space requirments of the algorithm form part (a)? Give your answer using Θ-notation, in terms of n.

$\Theta(n^2 - 1 - 0 + 1) \Rightarrow \Theta(n^2)$

(e) What is the auxilliary space requirments of the algorithm form part (b)? Give your answer using Θ-notation, in terms of n.

$\Theta(n)$

(f) What is the auxilliary space requirment of your algorithm form part (c)? Give your answer using Θ-notation, in terms of n.

$\Theta(n^2)$

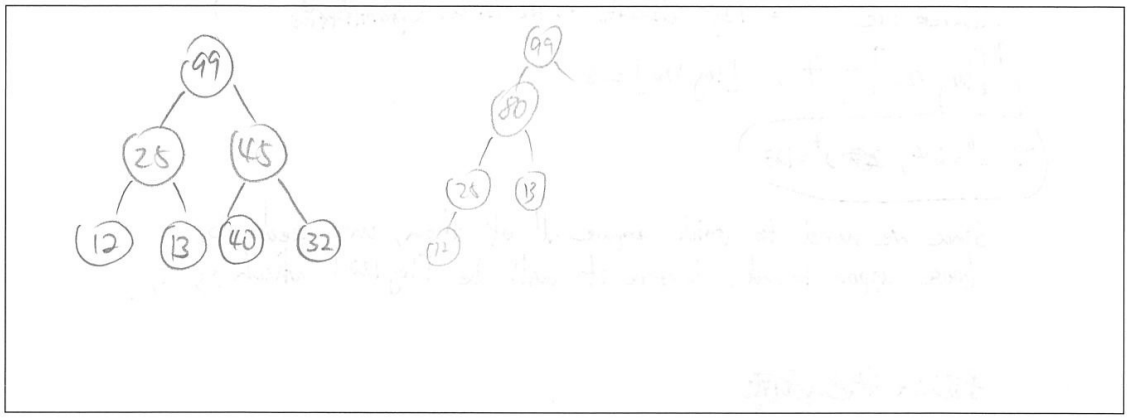
© 2015 University of Waterloo

4. /10 Heaps

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

$$A = [99, 25, 45, 12, 13, 40, 32]$$

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



(b) Write the new array that results after inserting 80 into the original heap.

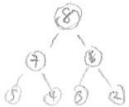
A=[99, 80, 45, 25, 13, 40, 32, 12]

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

A=[45, 25, 40, 12, 13, 32]

(d) Give the exact number of comparisons required in the worst case to heapify an array of length 7 using the bubble-up heapify method.

21



(e) Give the exact number of comparisons required in the worst case to heapify an array of length 7 using the bubble-down heapify method.

21

© 2015 University of Waterloo

5. /4 Lower Bounds

Explain why any algorithm to sort an array of size 5 **in the comparison model** must use at least 7 comparisons in the worst case.

∵ Size = 5

∴ There are 5! = 120 elements in the leaves. (probilities)

$\lceil \log 120 \rceil = 7$, $\lfloor \log 120 \rfloor = 6$

∵ $2^6 = 64$, $2^7 = 128$

Since we need to finish compare all of them, we need to choose upper bound. Therefore it will be $\lceil \log 120 \rceil$ which is 7.

So in the worst case we choose upper bound, 7.

© 2015 University of Waterloo

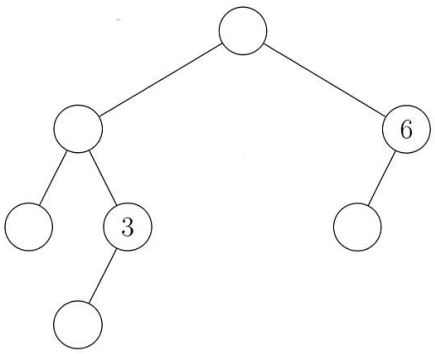
6. /8 AVL Trees: Part 1

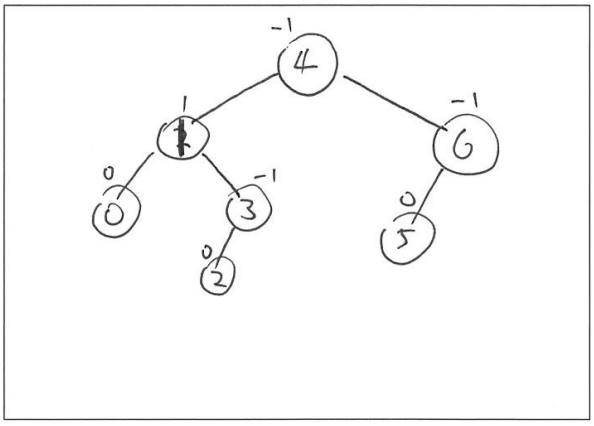
This question asks you to draw some AVL trees.

(a) The following binary tree on seven nodes has only three nodes labelled with integer keys.

i Redraw the tree in the answer box but fill in the unlabelled nodes with distinct nonnegative integers so that the tree is a binary search tree.

ii For the new tree in the answer box, indicate the balance factor of each node.

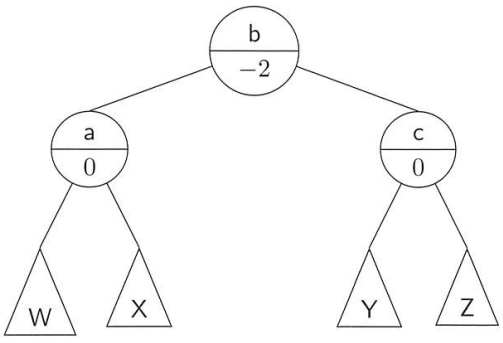


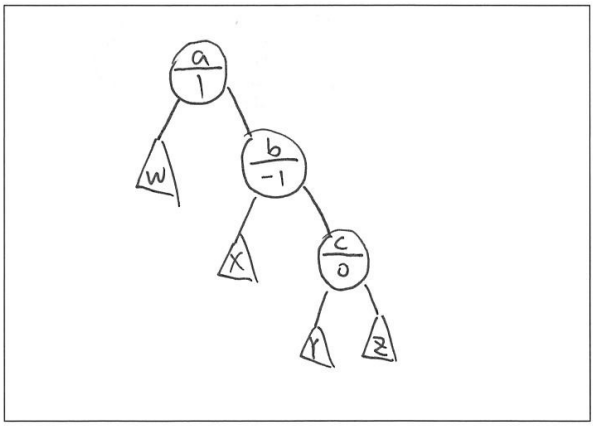


(b) Shown below is a binary tree T on nodes a, b and c and unknown subtrees W, X, Y and Z.

i Draw the tree that results by calling $fix(T)$. The balance factors are shown in the lower part of each node.

ii Indicate the balance factors of a, b and c in the new tree.

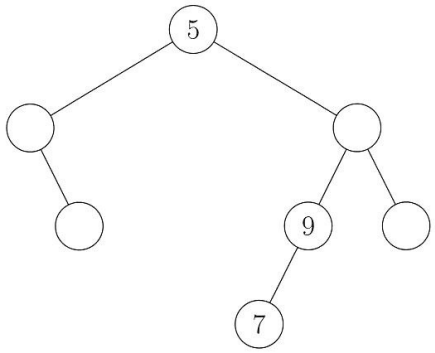




© 2015 University of Waterloo

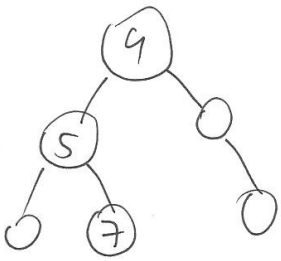
7. /4 AVL Trees: Part 2

The following is an AVL tree with 7 nodes. Only three nodes are labelled with the integer keys they contain.



i Draw the final AVL tree that results after deleting the in-order predecessor of the root node.

ii In the new tree you have drawn, indicate which nodes contain the keys 5, 7 and 9.



© 2015 University of Waterloo

Q8 Score: **4.0**

8. /6 Randomized Algorithms

Consider a size-n array A that contains distinct integers. Recall that the call *shuffle*(A) randomly permutes the elements in A. The boolean function *is-sorted*(A) returns *true* if A is sorted and false otherwise. Both *shuffle* and *is-sorted* have running time $\Theta(n)$.

Consider the following randomized sorting algorithm.

```
random-sort(A)
A: array of size n
1.    shuffle(A)
2.    if is-sorted(A) = false then
3.        random-sort(A)
```

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-sort*? $\Theta(n)$

(b) What is the **worst-case** running time of *random-sort*? ∞

(c) Let $T(n)$ be the **expected** running time of of *random-sort*. Write down a recurrence for $T(n)$ and then solve it. In your recurrence, use the fact that lines 1 and 2 of the algorithm use time cn for some constant c.

$$T(n) = \frac{1}{n!} \cdot \Theta(n) + \left(1 - \frac{1}{n!}\right) \cdot T(n)$$

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

$$= \Theta\left(\sum_{i=1}^{n!} n\right)$$

© 2015 University of Waterloo

9. /8 Algorithm Design

In this question you are asked to consider a max-heap H with $n \geq 1$ items tagged with distinct priorities, stored in an array $A[0 \ldots n-1]$. Let K denote the item in the heap that has k'th largest priority, for some $1 \leq k \leq n$.

(a) Considering H as a tree, give an explicit upper bound for the depth of K in the heap. Your bound should be tight for all values of n and k. $\lceil \log n \rceil$ ✓

(b) Give an explicit upper bound for m such that K is contained in $A[0 \ldots m-1]$. Your bound should be tight for all values of n and k. $n-k$ ✗

(c) Describe in words an algorithm that runs in time $O(k^2)$ to find (but not remove) item K. You may use any algorithms and data structures seen in class.

We/delete

We use delete max in a loop from 1 to k.

bubble down time log k. and return root.

$k \log k \in O(k^2)$

Correct idea, incorrect analysis

© 2015 University of Waterloo

Extra page A.

This page is intentionally left blank for your use. Do not remove it from your booklet. If you require more space to answer a question, you may use this page, but you must **clearly indicate** in the provided answer space that you have done so.

© 2015 University of Waterloo

C11EB731-C193-4B7A-85C9-498C41C462B5

CS240_F15_Midterm

#84 12 of 12

Extra page B

This page is intentionally left blank for your use. Do not remove it from your booklet. If you require more space to answer a question, you may use this page, but you must **clearly indicate** in the provided answer space that you have done so.

© 2015 University of Waterloo