373 18au Midterm practice

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Question	Points	Score
Design	0	
Binary Trees	0	
AVL Trees	0	
Hash	0	
Heap	0	
Runtime Analysis	0	
Total:	0	

Note:

This is a practice exam. It hasn't been reviewed thoroughly, so it's very likely there are typos and other mistakes. If something does not make sense, ask.

1. **Design choices:** For the following, choose the most appropriate data structure to solve the problem and briefly (less than 1-2 sentences) justify your answer.

   Stack, AVL tree, Binary search tree, Heap, Hash table, Linked List

   If there are multiple equally good candidates, list them all.

   (a) To store keys and associated values in a system such that the actual key values can be hidden from the implementor of the data structure.

   > **Solution:** Any data structure would work here. The key observation is that our data structures do not really need to know the exact key value, they just need to be able to compare two keys. The client can make Keys `comparable` without directly exposing key values.

   (b) Sorting an almost sorted list: An almost sorted list is a list where elements are "almost"sorted. This means that elements in the list can be in a position that is at most k places away from their sorted place. Assume that the list is too large to load into RAM to sort the entire list. Which data structure would you use to efficiently read this list and print a sorted list?

   > **Solution:** Heap. Since this is an almost sorted list, the i^{th} element in the fully sorted list is present in almost sorted list between $i-k$ and $i+k$. So to sort this list, we need to iterate through the list while maintaining a rolling set of $2k$ elements. As we iterate, we extract the smallest element from this $2k$ set, add it to the output, and add the next element from input list. Min-heap is the ideal data structure for this.

   (c) Close to homework deadlines gitlab pipelines get stuck. Gitlab pipelines execute jobs in a queue, on a first come first serve basis. This means that if one student submit too many jobs continuously, all other jobs will be held up. One solution is to implement a queue system that instills somewhat fairness is where subsequent jobs from the same student get lower priority. So if one student submits 10 jobs, his/her first job has the highest priority and the 10^{th} job has the lowest priority. Which data structure(s) would be appropriate to implement this? Why?

   > **Solution:** Linked list. The key observation here is that to implement the suggest fairness you need to keep track of how many jobs each student is submitting (think dictionaries) and you need a queue for students instead of a queue for jobs. Linked list can implement both (dictionary and queue).

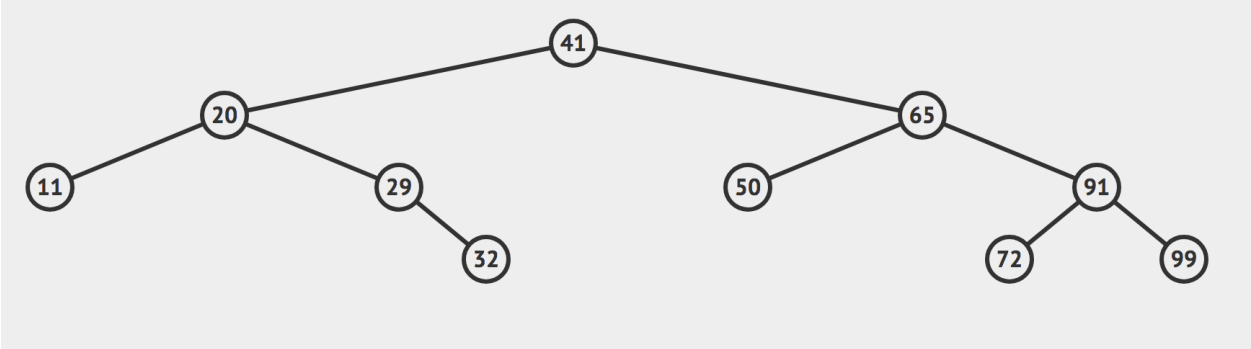


Figure 1: A binary tree

2. (a) What is the minimum number of nodes in a binary tree of height h:

(a) $h+1$

(b) What is the maximum number of nodes in a binary tree of height h:

(b) $2^{h+1}-1$

(c) What is the maximum number of leaf nodes a binary tree of height h can have:

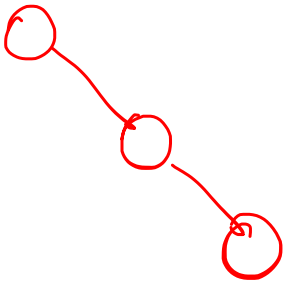
(c) 2^h

(d) Print pre-order of the tree in Figure 1:

(d) _____

41	20	11	29	32	65	50	91	72	99		

Example tree with height 2



3. AVL Trees

(a) Insert the following sequence of values into an empty AVL tree *in the given order*.

10, 20, -1, 0, 5, 14, 1, 6, ~~8~~, ~~8~~

Typo. I won't ask you to insert duplicated keys in AVL

You do not have to show the intermediary steps, but no work and a wrong answer will result in no credit. Draw your final tree in the following figure, Figure 2. Clearly, there are more nodes in the figure than in your final tree. Leave the unused nodes empty. You may use the space below Figure 2 to draw your intermediary trees or for scratch work.

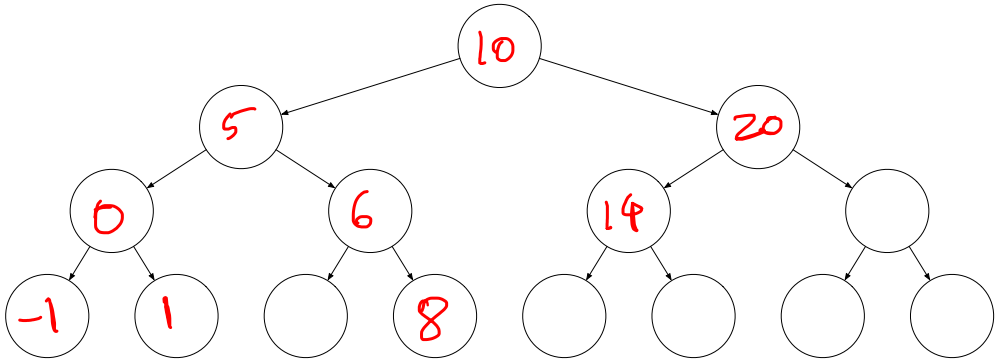
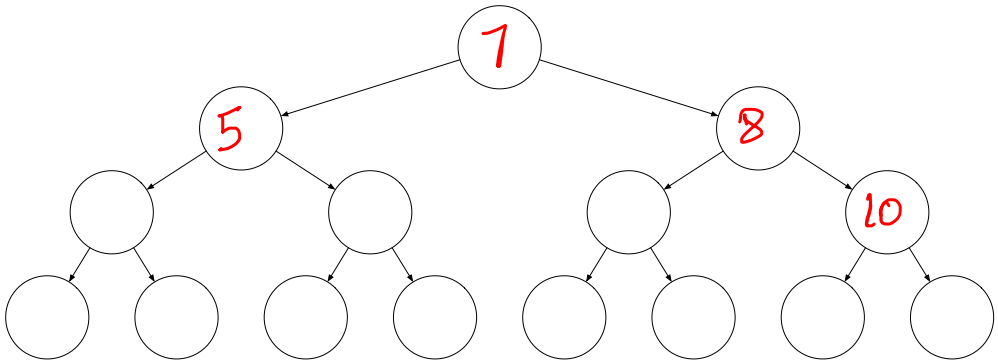


Figure 2: Fill this tree with your final answer. Leave unused nodes empty.

(b) Consider a set $S = 10, 7, 15, 8, 20, 5, 16, 25$. With four keys from this set create a 4-node AVL tree such that any fifth insertion done with **any** remaining key in the set will break the AVL balance property of the tree. Draw your 4-node AVL tree in the following figure. Leave unused nodes empty.



Solution: There are multiple solutions to this. The above figure shows one of them. The key observation here is that you want to choose the keys for your 4-node AVL tree such that the 5th key (whatever it may be) will be inserted only in a location that will break the AVL balance property. The simplest way to do this is to pick the four smallest keys from the set and draw a AVL tree with those.

4. Consider inserting data with integer keys 41, 94, 95, 56, 87, 65, 5, 25, 18, 74, 62, 22, 25, 66, 7 in that order into a hash table of size 20 where the hashing function is $h(key) = key + 3$. Show an open addressing that uses double hashing after doing the insertions. The second hash function is $g(key) = key \ \% \ 23$. Do not worry about resizing.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	18	74		41	62		22	65	66	87	7	25	5				94	95	56

index where a key should be inserted is given by the formula

$h(k,i) = h(k) + i \times g(k)$

they are different functions

$h(k,i)\ \%\ 20$ for $i = 0$

Keys	h(key)	g(key)	
41	44	18	4 ✓
94	97	2	17 ✓
95	98	3	18 ✓
56	59	10	19 ✓
87	90	18	10 ✓
65	68	19	8 ✓
5	8	5	8
25	28	2	8
18	21	18	1 ✓
74	77	5	17
62	65	16	5 ✓
22	25	22	5
25	28	2	8
66	69	20	9 ✓
7	10	7	10

1st collision

2nd collision

$h(k,i)\ \%\ 20$ for $i = 1$

13 ✓

10 collision

17 ✓

$h(k,i)\ \%\ 20$ for $i = 2$

12 ✓

12 ✓ But it's same key so you replace it

4

$h(k,i)$ for $i = 3$

11 ✓

5. Insert the following sequence of values in an empty min-heap in the given order.

99, 77, 92, 97, 35, 42, 51, 14, 10, 80, 29, 15, 53,

Fill Figure 3 with your final answer. You do not have to show the intermediary steps, but no work and a wrong answer will result in no credit. Drawing intermediate trees may result in partial credit. You can use the space below Figure 3 or the back of this page to draw your intermediary steps.

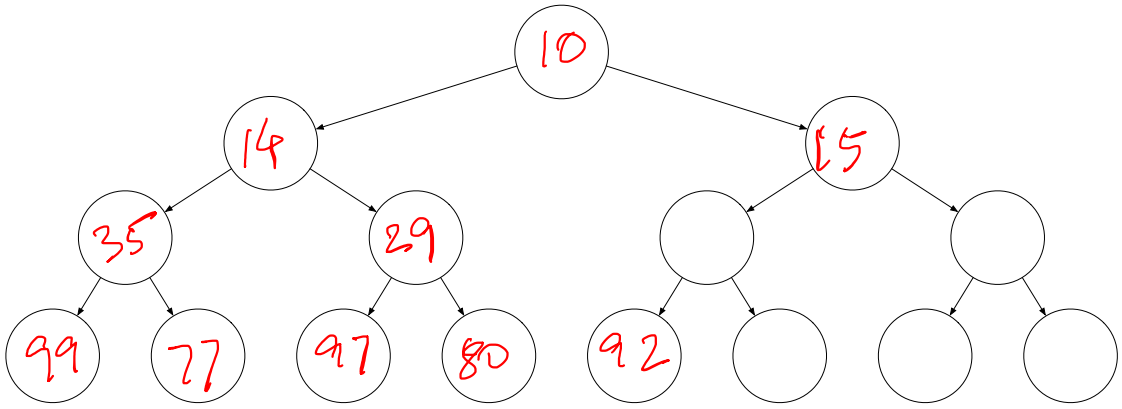


Figure 3: Fill this tree with your final answer. Leave unused nodes empty.

6. For each of the following, describe the worst-case running time with respect to variable n (and any additional specified variables). Please give a tight big-O bound and in the most simplified form. You do not need to justify your answer.

   (a) Give a tight big-O for function $f(n)$. $f(n) = 100n^3 + 4n - 100n^3$: : _____

   (b) Give a tight big-O for function $f(n)$. $f(n) = 25n \log n + 50n + 2n^2$: : _____

   (c) Find recurrence $T(n)$ for the following snippet of the code

```
public int bar(int n) {
    if (n <= 10) {
        return 0;
    }
    for (int i = 0; i < n; i++) {
        System.out.println("!");
    }
    int a = 2 + bar(n/2);
    int b = 1 + bar(n/2)
    return a + b;
}
```

Solution:

$$T(n) = \begin{cases} c_1 & \text{when } n \leq 10 \\ 2T(n/2) + n + c_2 & \text{otherwise} \end{cases}$$

   (d) Demonstrate that $4 + 100n$ is dominated by n by finding a c and n_0. Show your work.

Solution: By definition of "dominated by" and big-O, we need to show $4+100n \leq c \cdot n$ for all $n \geq n_0$ where c and n_0 are constant values.

Scratch work: Our goal is to show that $f(n) = 4 + 100n$ dominates $g(n) = n$. We start by comparing terms.

$$100n \leq 100n \qquad \text{for any } n$$
$$4 \leq n \qquad \text{if } n \geq 4$$

We add together the inequalities to get

$$100n + 4 \leq 101n \qquad \text{if } n \geq 4$$

So our $n_0 = 4$ and $c = 101$.

Thus, $4 + 100n \leq c \cdot n$ is true for our chosen values of $c = 101$ and for all $n \geq n_0$.

   (e) Simplify the following summation to a closed form

$$\sum_{i=20}^{n} i$$

Solution:

$$\begin{aligned}
\sum_{i=20}^{n} i &= \sum_{i=0}^{n} i - \sum_{i=0}^{19} i && \text{Adjusting summation bounds} \\
&= \sum_{i=0}^{n} i - \frac{19 \cdot 20}{2} && \text{Gauss' identity} \\
&= \sum_{i=0}^{n} i - \frac{19 \cdot 20}{2} \\
&= \frac{n(n+1)}{2} - \frac{19 \cdot 20}{2} && \text{Gauss' identity}
\end{aligned}$$

NOTE: This closed form is sufficient for full credit. You do not need to simplify this further. In general, when asked to simplify summations, you just need to simplify enough to get rid of the summations; you do not need to simplify more than that.