CSE 373 19su: Midterm Solutions

Name:

UW email address:

Instructions

- **Do not start the exam until told to do so; immediately stop writing when time is called.**
- No electronic devices (including calculators and cell phones) are allowed.
- You are allowed one sheet of 8.5" x 11" paper (both sides) of notes.
- You have 60 minutes to complete the exam.
- Write your answers neatly in the provided space. Be sure to leave some room in the margins: we will be scanning your answers.
- If you need extra space, you may use the back of a sheet, **but you must clearly indicate in each subproblem where you want us to read the back**
- If you have a question, raise your hand to ask the course staff for clarification.
- Closed-forms of some summations, some logarithm identities, and Master Theorem are on a separate sheet.

Advice

- Write down your assumptions, thoughts and intermediate steps (this makes it easier to award partial credit).
- Clearly circle your final answer.
- The questions are not necessarily in order of difficulty; skip around!

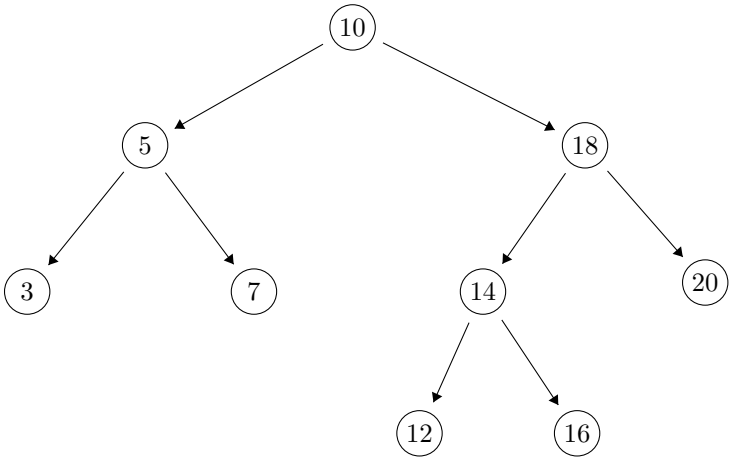
1. AVL Trees

Insertions

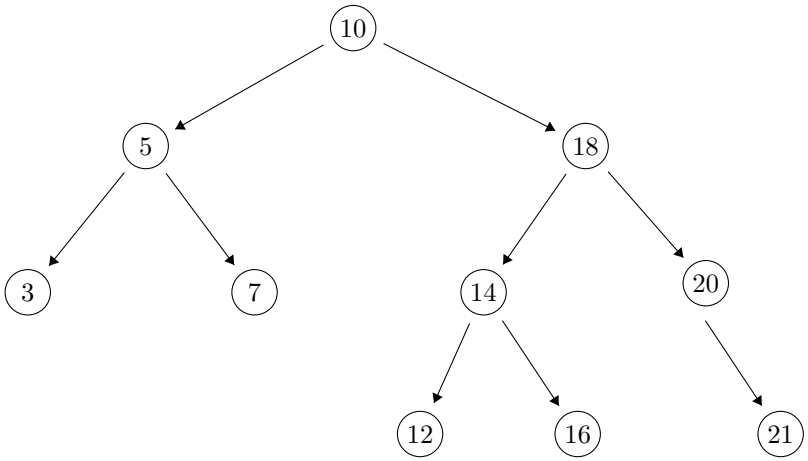
For each of the following AVL trees and keys to insert:

- On the top (given) tree, draw the new node where it will be inserted into the tree (before any rotations occur).
- On the top (given) tree, put a box around **every** imbalanced node in the tree (if any).
- In the bottom (empty) tree, draw the final tree after any rotations are performed. If no rotations are needed, you may write "no rotations" instead of filling in the tree. If a node does not exist in the final tree, leave that circle blank.

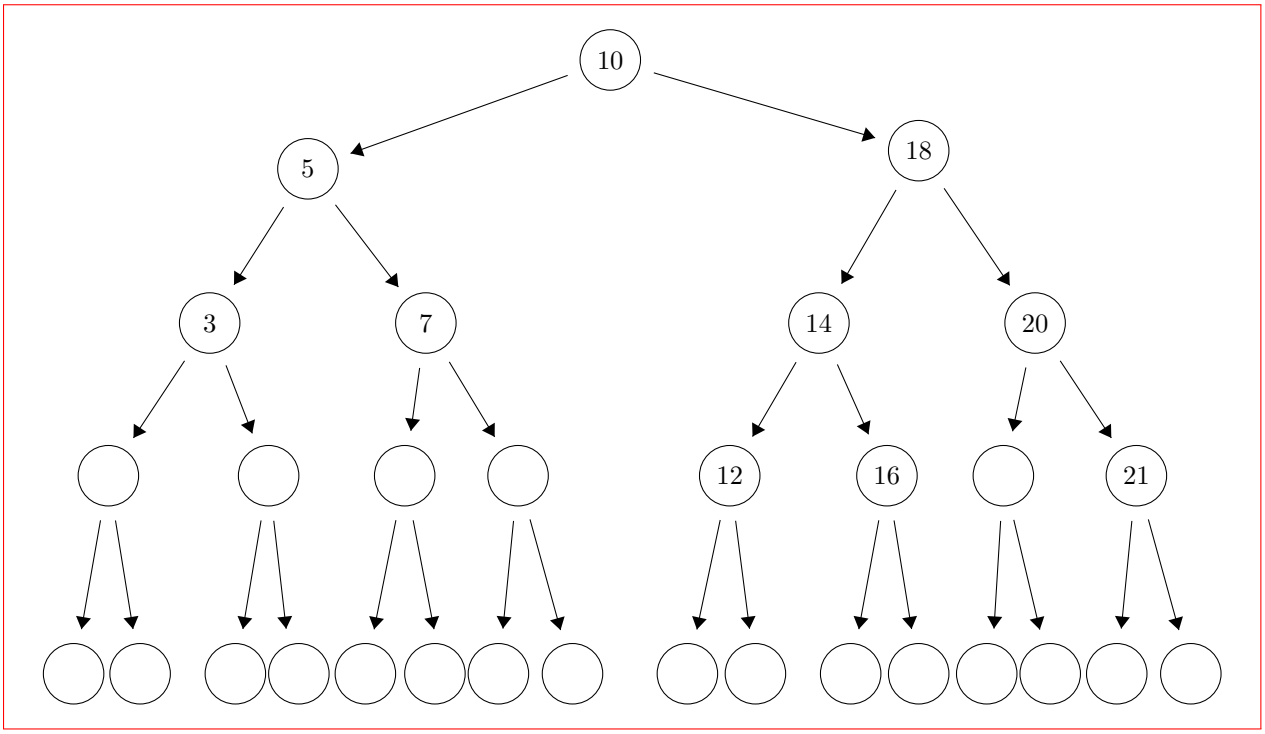
(a) Item to insert: 21



Solution:



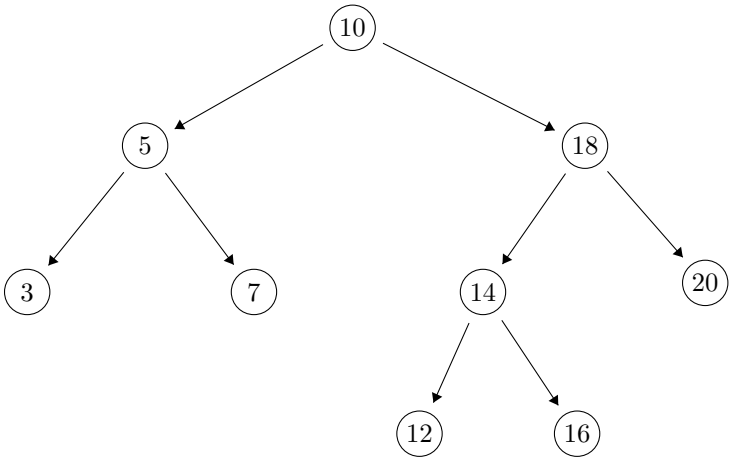
No rotations.


10
5
18
3
7
14
20
12
16
21

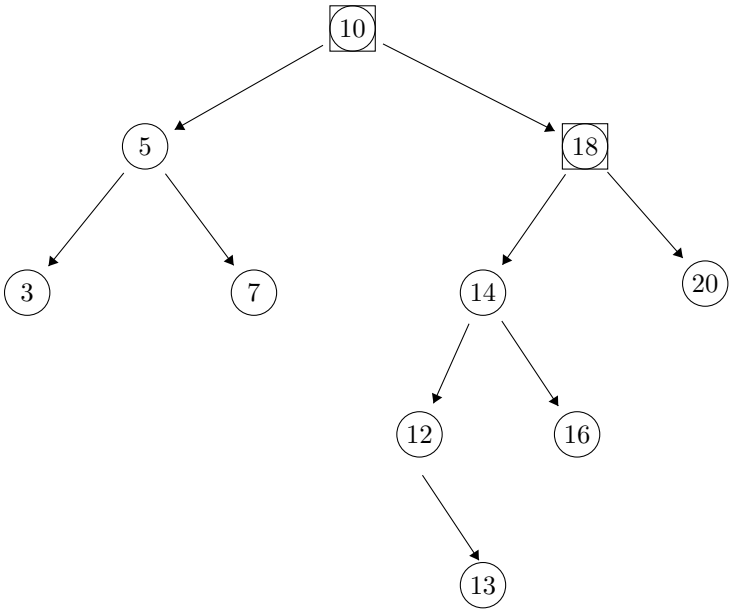
This is a continuation of the AVL insertion problems. The directions below are identical to the previous part.

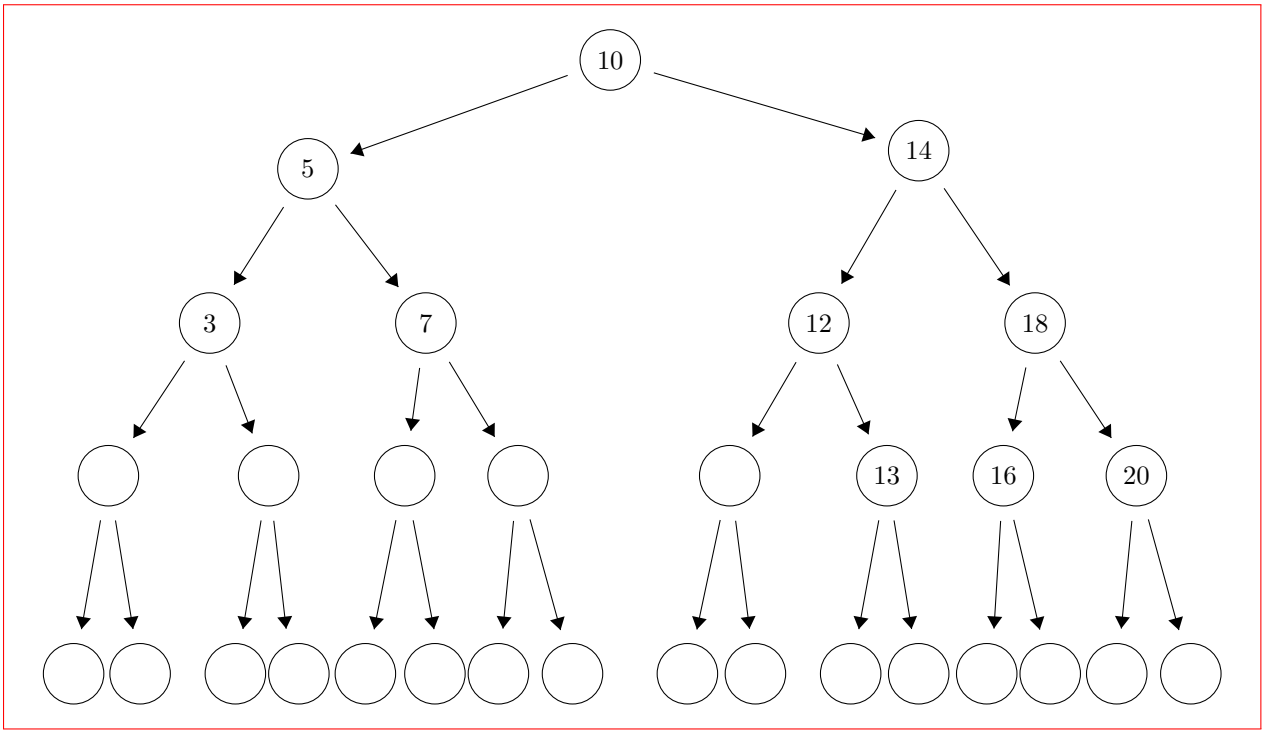
- On the top (given) tree, draw the new node where it will be inserted into the tree (before any rotations occur).
- On the top (given) tree, put a box around **every** imbalanced node in the tree (if any).
- In the bottom (empty) tree, draw the final tree after any rotations are performed. If no rotations are needed, you may write "no rotations" instead of filling in the tree. If a node does not exist in the final tree, leave that circle blank.

(b) Item to insert: 13



Solution:

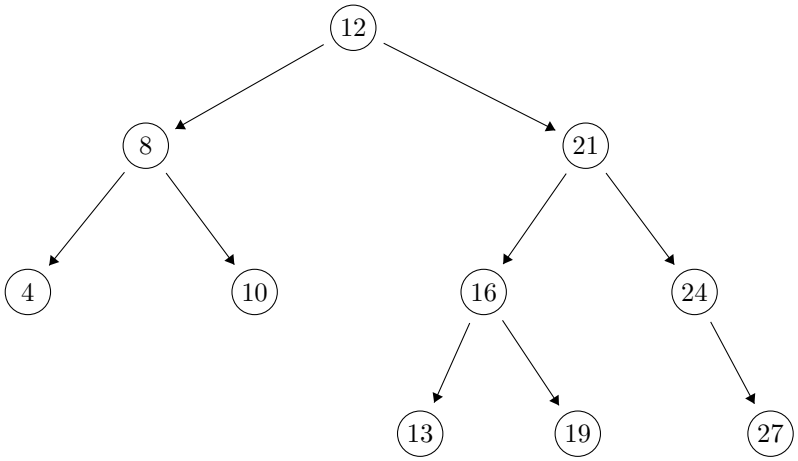



10
5
14
3
7
12
18
13
16
20

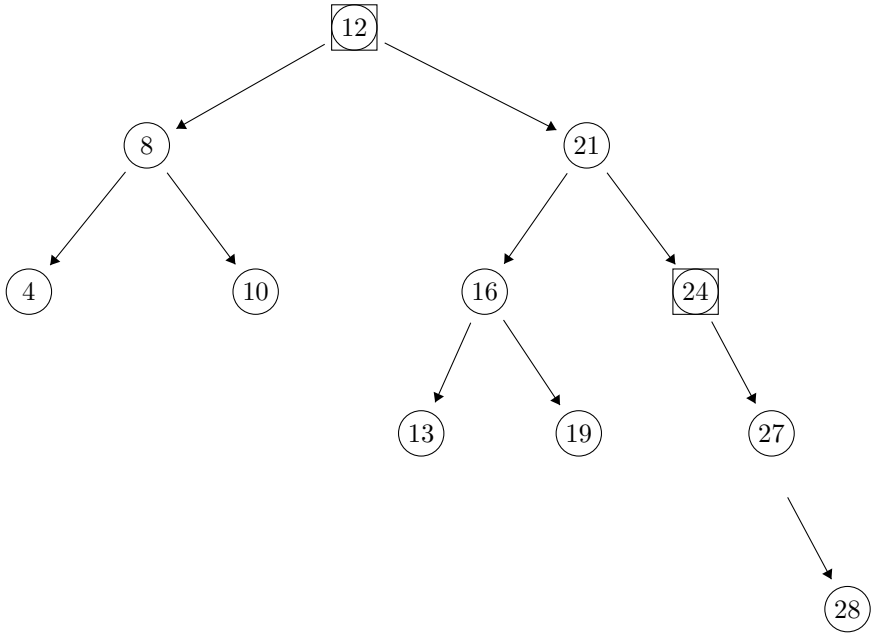
This is a continuation of the AVL insertion problems. The directions below are identical to the previous part.

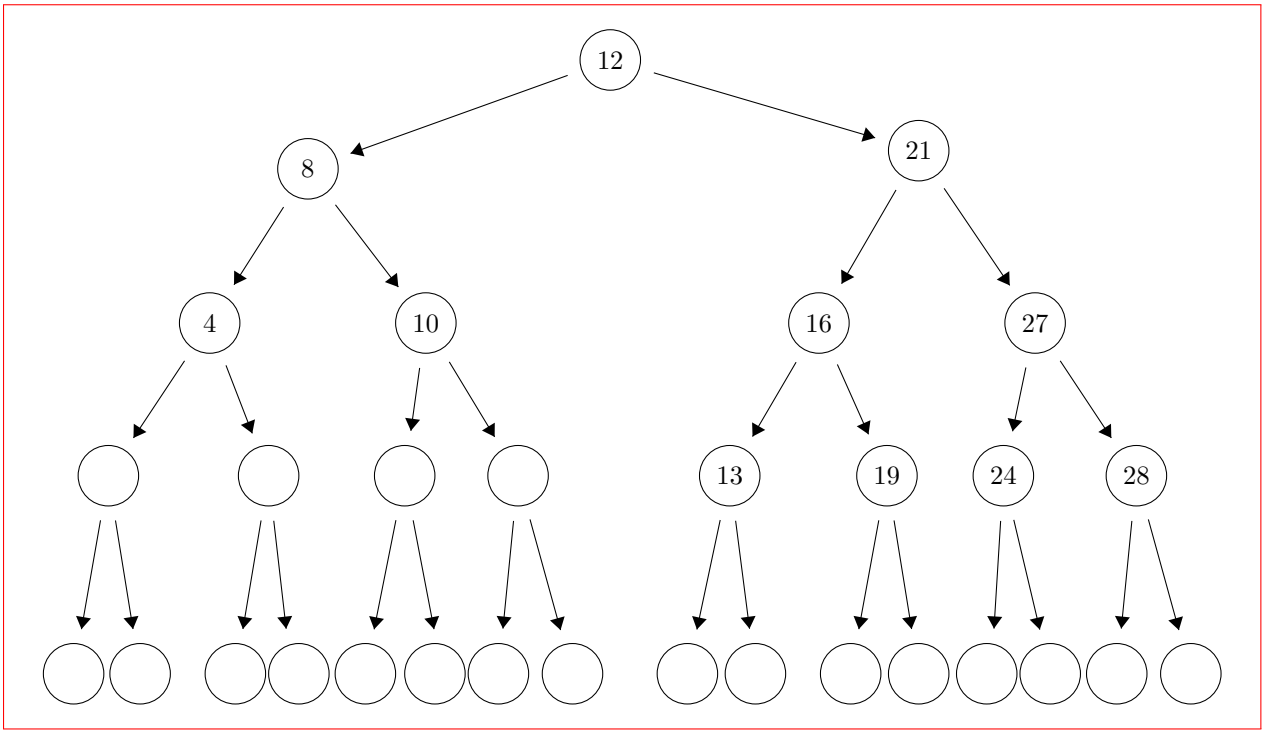
- On the top (given) tree, draw the new node where it will be inserted into the tree (before any rotations occur).
- On the top (given) tree, put a box around **every** imbalanced node in the tree (if any).
- In the bottom (empty) tree, draw the final tree after any rotations are performed. If no rotations are needed, you may write "no rotations" instead of filling in the tree. If a node does not exist in the final tree, leave that circle blank.

(c) Item to insert: 28



Solution:

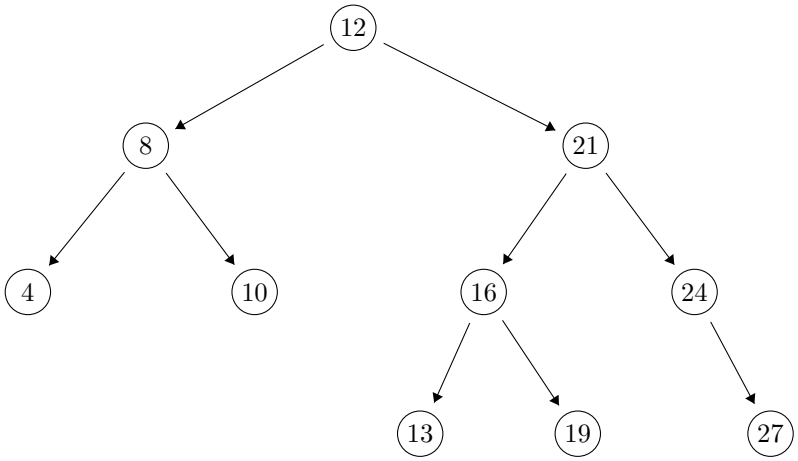



12
8
21
4
10
16
27
13
19
24
28

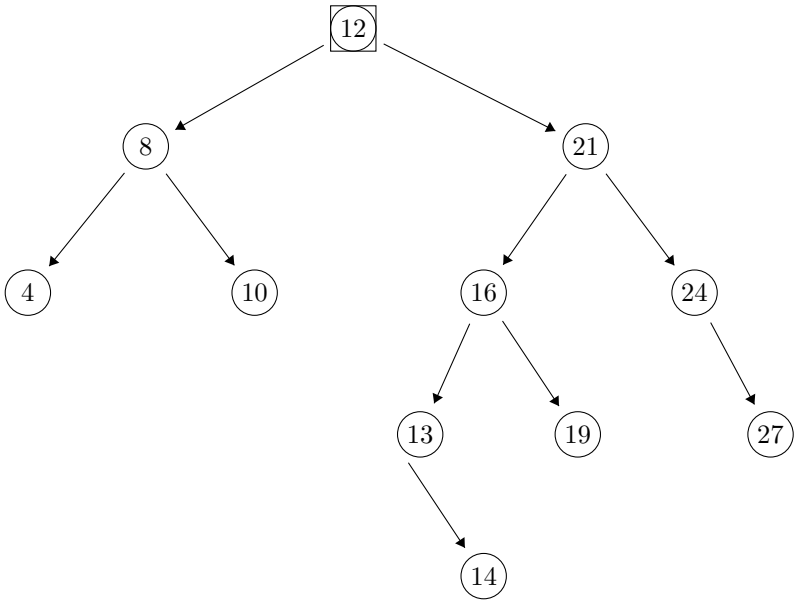
This is a continuation of the AVL insertion problems. The directions below are identical to the previous part.

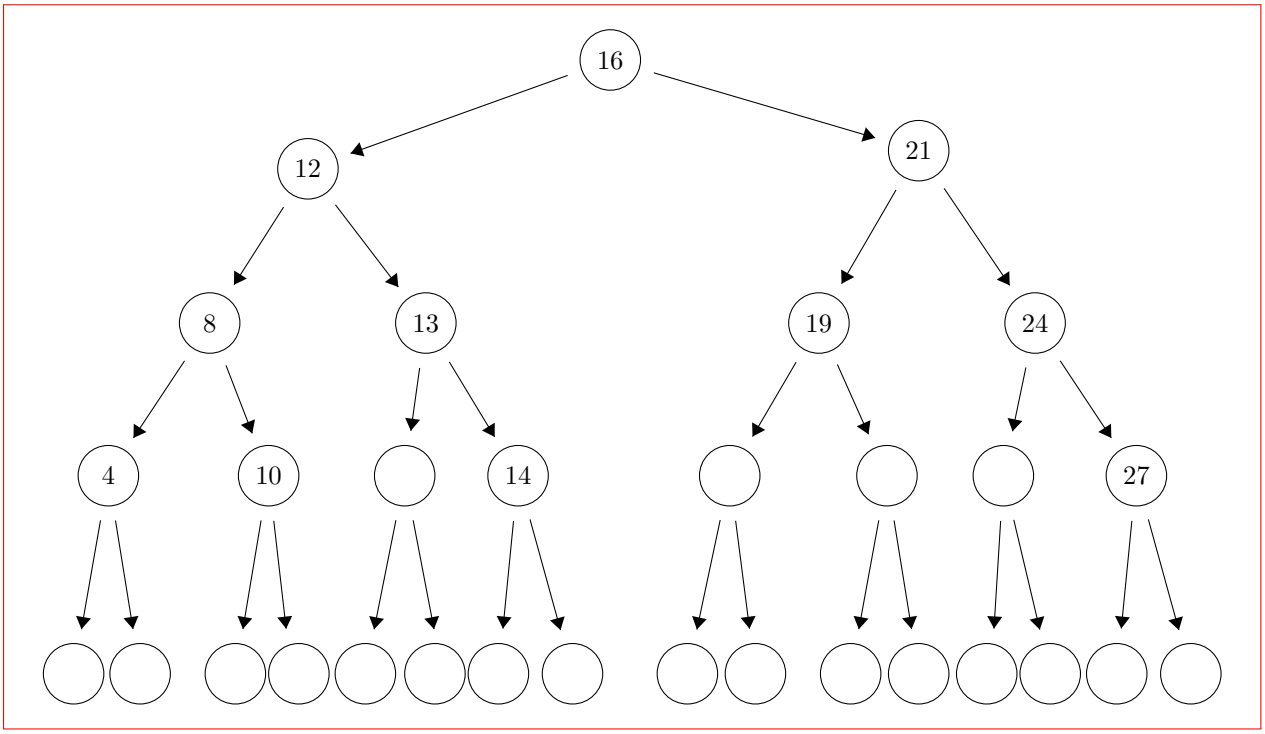
- On the top (given) tree, draw the new node where it will be inserted into the tree (before any rotations occur).
- On the top (given) tree, put a box around **every** imbalanced node in the tree (if any).
- In the bottom (empty) tree, draw the final tree after any rotations are performed. If no rotations are needed, you may write "no rotations" instead of filling in the tree. If a node does not exist in the final tree, leave that circle blank.

(d) Item to insert: 14



Solution:




16
12
21
8
13
19
24
4
10
14
27

AVL Trees short answer

(a) Give an ordering of the keys: $1, 2, 3, 4, 5, 6, 7$ such that inserting those keys into an empty AVL tree will cause no rotations.

Solution: One possibility (there are others): $4, 2, 6, 1, 3, 5, 7$.

AVL Trees T/F

For each of the following statements:

- State whether the statement is true or false.
- If the statement is false, briefly explain (1-2 sentences)

(a) If you're inserting a **new** key into an AVL tree, the best-case running time is $\mathcal{O}(1)$.

Solution: False. We have to create a new node, which always happens below a leaf node. All leaf nodes are distance $\Omega(\log n)$ from the root, hence the running time is at least $\Omega(\log n)$.

(b) The **maximum** number of rotations on a single `insert` in an AVL tree is $\Theta(1)$.

Solution: True. (Two rotations is the maximum, and two is a constant. Big-$\mathcal{O}/\Omega/\Theta$ notation can be used on any function (not just running time.)

(c) Deleting a node from an AVL tree can take $\Theta(n)$ time, so AVL trees always use lazy deletion.

Solution: False. Deletion takes time $\Theta(\log n)$; there are rotations (similar to the insertion ones) which mean lazy deletion is not required. Open-addressed hash tables are the data-structures we said essentially always use lazy deletion.

2. Hashing

(a) Below is a hash-table implementation of a dictionary. The table uses the hash function $h(x) = x$ and quadratic probing to resolve collisions. A few (key,value) pairs have already been put into the table.

`Put` the following key,value pairs (in order) into the hash table. Show the final result. Assume the table does not resize. If an insertion fails, write "(key,value) failed" below the table, then continue with the next insertion as though the failed insertion were never called.

$$(19, \texttt{`}c'),\ (8, \texttt{`}f'),\ (17, \texttt{`}d'),\ (0, \texttt{`}z')$$

0	1	2	3	4	5	6	7	8	9
$(0, \texttt{`}b')$	$(21, \texttt{`}a')$						$(7, \texttt{`}d')$		$(9, \texttt{`}a')$

Solution:

0	1	2	3	4	5	6	7	8	9
~~$(0, \texttt{`}b')$~~ $(0, \texttt{`}z')$	$(21, \texttt{`}a')$		$(19, \texttt{`}c')$			$(17, \texttt{`}d')$	$(7, \texttt{`}d')$	$(8, \texttt{`}f')$	$(9, \texttt{`}a')$

(b) Suppose you have a hash table of `TableSize` T. Using the hash function $h(x) = x$, you have already inserted the keys $0, 2, 4, 6, \ldots, T/2$. Note that there are $T/4$ keys already in the table.

Give a list of **about** $T/4$ more distinct keys to insert into the hash table such that if you insert those keys into the table in the order you listed:

- A table using separate chaining will have each insertion take $\mathcal{O}(1)$-time.
- AND A table using linear probing will suffer at least one insertion that takes $\Omega(T)$ time.

Solution: The key to this problem is to create a primary cluster in the linear probing table without creating long chains in the separate chaining table.

One possible answer is : $\mathbf{1}, \mathbf{3}, \ldots, \mathbf{T/2 - 1}, \mathbf{T}$. For separate chaining, every insertion except for the last is inserting into an empty chain, so it is certainly $\mathcal{O}(1)$-time. and for the last T, since only 1 element is in the chain at index 0, separate chaining also only takes $\mathcal{O}(1)$-time.

However, for linear probing, when we try to insert T, we will probe at all of $0, 1, \ldots, T/2$, which takes time $\Omega(T)$.

(c) Which collision resolution strategy can resize **least** frequently? Why can its load factor safely get larger than any of the other strategies we have discussed? (1-2 sentences)

Solution: Separate chaining. In separate chaining, each bucket contains a linked list, which can store an arbitrary number of items, and therefore λ can exceed 1. Open-addressing strategies will always fail to insert if $\lambda = 1$.

3. Closed Form of a Recurrence

In this problem, we will use the tree method to find the closed-form of the following recurrence.

$$T(n) = \begin{cases} 5 & \text{if } n \leq 2 \\ T(n-2) + n^2 & \text{otherwise} \end{cases}$$

You may assume that n is even in all of your calculations.

(a) What is the size of the **input** to each node at level i? What is the amount of **work** done by each node at the i^{th} recursive level? As in class, we call the root level $i = 0$. This means that at $i = 0$, your expression for the input should equal n.

Solution: Input size $n - 2i$. Work $(n - 2i)^2$.

(b) What is the total number of nodes at level i?

Solution: 1.

(c) What is the total work done across the i^{th} recursive level?

Solution: $(n - 2i)^2$.

(d) What value of i does the last level of the tree occur at? Show your work.

Solution: We want $n - 2i \leq 2$, which leads to $i = \frac{n-2}{2} = \frac{n}{2} - 1$.

(e) What is the total work done across the base case level of the tree (i.e. the last level)?

Solution: 5.

(f) Combine your answers from previous parts to get an expression for the total work. Simplify the expression to a closed form. (You may want to consult the identities sheet) Show your work.

Solution:

Total work done = work done in the base case + work done in all recursive cases

$$
\begin{aligned}
&= 5 + \sum_{i=0}^{\frac{n}{2}-2} (n-2i)^2 \\
&= 5 + \sum_{i=0}^{n/2-2} n^2 - \sum_{i=0}^{n/2-2} 4in + \sum_{i=0}^{n/2-2} (2i)^2 \\
&= 5 + n^2(n/2-1) - 4n \sum_{i=0}^{n/2-2} i + 4 \sum_{i=0}^{n/2-2} i^2 \\
&= 5 + n^2(n/2-1) - 4n \left(\frac{(n/2-2)(n/2-1)}{2} \right) + 4 \left(\frac{(n/2-1)(n/2-2)(n-3)}{6} \right)
\end{aligned}
$$

(g) From the closed form you computed above, give a big-Θ bound. No need to prove it, but do show your work. (You may want to consult the identities sheet.)

Solution:

$$
\begin{aligned}
T(n) &= 5 + n^2(n/2-1) - 4n \left(\frac{(n/2-2)(n/2-1)}{2} \right) + 4 \left(\frac{(n/2-1)(n/2-2)(n-3)}{6} \right) \\
&= 5 + n^3/2 - n^2 - 2n \left(n^2/4 - 3n/2 + 2 \right) + 2/3 \left((n^2/4 - 3n/2 + 2)(n-3) \right) \\
&= 5 + n^3/2 - n^2 - n^3/2 + 3n^2 - 4n + 2/3 \left(n^3/4 - 3n^2/2 + 2n - 3n^2/4 + 9n/2 - 6 \right) \\
&= \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3} + 1
\end{aligned}
$$

$\Theta(n^3)$.

(h) Let's try to use Master Theorem to check our answer. Either

- write the inequality and which case of Master Theorem applies, and the final big-Θ
- **OR** explain why Master Theorem doesn't apply (1-2 sentences)

Master Theorem is stated for you on the back of your formula sheet.

Solution: Master Theorem doesn't apply; we're subtracting from the input size, not dividing the input size.

4. Big-O Definitions

True/False.

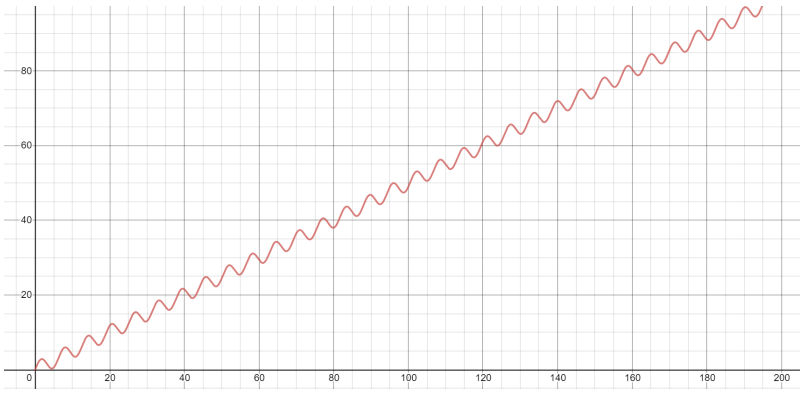
For each of the following statements:

- State whether it is true or false.
- If it is false, clearly explain why. (at most 1-2 sentences)

(a) When we're finding the best-case big-$\mathcal{O}$ for a method that takes a list as input, we usually consider the case that the list has one element.

Solution: False. $\mathcal{O}, \Omega, \Theta$ are asymptotic terms – they only make sense as n gets arbitrarily large (larger than n_0), so we *never* consider $n = 1$ as the best-case.

(b) Because the running time of this function is "wiggly" it does not have a simple big-Theta. (Assume the function continues on the same growth rate for all n)



Solution: False. From the curve of the function, we can see that $n/3 \leq T(n) \leq 2n$, so we can say $T(n) = \Theta(n)$. Big-Theta ignores constant factors.

(c) If $f_1(n)$ is $\mathcal{O}(g(n))$ and $f_2(n)$ is $\mathcal{O}(g(n))$ then the function $f_1(n) + f_2(n)$ is $\mathcal{O}(g(n))$.

Solution: True.

Simplified Big-O

For each of the following functions, give the simplified, tight big-$\mathcal{O}$.

(a) $a(n) = n\log(n^3) + n^2$

(b) $b(n) = n^3 - 5n^2$

(c) $c(n) = 2^{\log_2(n)} + n^3$

(d) $d(n) = 2^{n/2} + n^3$

Solution:

- $a(n) = \mathcal{O}\left(n^2\right)$.
- $b(n) = \mathcal{O}\left(n^3\right)$.
- $c(n) = \mathcal{O}\left(n^3\right)$.
- $d(n) = \mathcal{O}\left(2^{n/2}\right)$.

O, Omega, Theta

For each of the following, circle T if the statement is true, F if the statement is false. No explanation required. The functions are the same as those defined in the previous part.

(a) $a(n) = n\log(n^3) + n^2$ is $\Omega(n\log(n))$ **T / F**

Solution: True

(b) $b(n) = n^3 - 5n^2$ has no simple-Θ because it has a negative term. **T / F**

Solution: False

(c) $c(n) = 2^{\log_2(n)} + n^3$ is $\mathcal{O}\left(n^2\log(n)\right)$ **T / F**

Solution: False

(d) $d(n) = 2^{n/2} + n^3$ is $\Theta(2^n)$ **T / F**

Solution: False

5. Code Modeling

In the following code, we will record late days for groups on a pair-programming project. Each of the data structures are the implementations discussed in class.

The function `updateLateDays` runs in constant time. `commits` is a (doubly) `LinkedList` of m commits. It contains the final commit made by each of the m groups on the last programming project. You should assume that every group has exactly one commit in the list. The `AVLDictionary` `groups` has n keys – one for each student in the course. The associated value is their partner's netid, or `null` if they are working alone.

Answer the following questions about code that might appear in our grading infrastructure.

```
1  public void RecordLateDays(LinkedList<Commit> commits, AVLDictionary<String, String> groups){
2          int m = commits.size();
3          for(int i = 0; i < m; i++){
4                  Commit c = commits.get(i);
5                  int days = daysBetween(c.getTime(), DUE_DATE); //runs in O(1) time
6                  updateLateDays(days, c.commiterID); //runs in O(1) time
7                  String partner2 = groups.get(c.committerID)
8                  updateLateDays(days, partner2); //runs in O(1) time
9          }
10 }
```

(a) What is the simplified big-Θ for the worst-case running time of line 4? Give your answer in terms of m.

Solution: $\Theta(m)$.

(b) What is the simplified big-Θ for the best-case running time of line 4? Give your answer in terms of m.

Solution: $\Theta(1)$.

(c) What is the simplified big-Θ for the worst-case running-time of line 7? Give your answer in terms of n.

Solution: $\Theta(\log n)$.

(d) What is the simplified big-Θ for the best-case running time of line 7? Give your answer in terms of n.

Solution: $\Theta(1)$.

(e) What is the simplified big-Θ for the worst-case running time of `RecordLateDays`? Give your answer in terms of n and/or m.

Solution: Normally we would get $m^2 + m \log n$, but the setup implies that $m \leq n \leq 2m$, so $m \log n$ is actually always dominated. The simplified version is: $\Theta(m^2)$.

(f) What is the simplified big-Θ for the best-case running time of `RecordLateDays`? Give your answer in terms of n and/or m.

Solution: $\Theta(m^2)$. `RecordLateDays` can only run in one way – even though lines 4 and 7 can be quick, over all the iterations of the loop, you're still going to have a $\sum_{i=0}^{n} i$ from all the `get` calls on `commits`.

It's still dominated, but the contribution of `get` calls on `groups` still totals $\Theta(m \log n)$ (we're `get`ting half of the nodes in the AVL tree; since most nodes are at the "bottom" of a balanced tree, the total contribution is still $\Omega(m \log n)$).

(g) Let m and n be defined as in the problem. Is every function that is $\Theta(m)$ also $\Theta(n)$? Briefly explain (1-2 sentences)

Solution: Yes. Because $\frac{n}{2} \leq m \leq n$. So if $c_1 m \leq f(m,n) \leq c_2 m$ when $m \geq m_0$, we also have $\frac{c_1}{2} n \leq f(m,n) \leq 2c_2 n$.

(h) We haven't written our code very well. Explain how we could make our code more efficient (1-2 sentences)

Solution: In the for loop, instead of using index i, we should use an `iterator` over the `LinkedList`. Instead of `get` calls, which total $\Theta\left(m^2\right)$, we'll total only $\Theta\left(m\right)$ time for those calls (and the final running time would be $\Theta\left(m \log n\right) = \Theta\left(m \log m\right)$, since the other term now dominates).

6. Design Decisions

For each of the following scenarios, select one of the following ADTs that would best fit the situation, and briefly (2-3 sentences) justify your choice.

ADTs: List, Stack, Queue, Dictionary

You've decided that GMail's app isn't good enough – you're going to customize your own email manager for your own particular needs.

(a) As part of the overhaul, you've decided to change the way you read email. To ensure you actually respond to everyone, you will answer the email that has been sitting in your inbox the longest (regardless of who it came from).

Solution: Queue. Because queue is first-in-first-out, we can make sure that oldest email gets replied first.

(b) You respond to people slowly. Sometimes people text you telling you to check you email. When you get such a text, you want to quickly find the most recent email you received from that person. You only care about finding the one most recent email per-person.

Solution: Dictionary. We can maintain a dictionary where each key is a person and the value is the last email we received from that person. It's now easy to maintain this information when we get new emails, and easy to lookup the most recent email when we get a text (just look up that key).

Despite the mention of "most recent" a stack isn't a great solution here – if we have many people emailing us, it could take a long time to find the most recent email from a given person.

(c) You'd like to have another screen that shows all the emails where you've replied and are waiting for a response. You should be able to view those emails ordered based on how long you've been waiting, and remove an email from the screen when you get a response.

Solution: List. We want to be able to remove from anywhere (whenever we get a response) while still having items ordered. A list lets us maintain an ordering, while still quickly ($\Theta(1)$) removing anything (not just the front or back, like a stack or queue).

For the following scenario, select one of the following data structures to implement the Dictionary ADT most effectively and briefly (2-3 sentences) justify your choice.

Data structures: hash table with separate chaining, hash table with linear probing, AVL tree

You are writing a dictionary that will be shared by many users (i.e. multiple users will be able to insert their own key,value pairs). Your manager firmly believes in posting your code publicly, despite knowing that your company's competitors like to create dummy accounts and try to interfere with your code's performance.

Solution: AVL tree. With your code public, and enemies with the ability to insert keys, you should be very concerned about an artificially high number of collisions, and hitting $\Theta(n)$ behavior for hash tables. For "adversarial" input like this (input where someone is trying to bring about the worst-case) AVL trees are a better option, because they have worst-case $\Theta(\log n)$ for dictionary operations.

7. Extra Credit

Communicate your current feelings about data structures in a small piece of art. Suggested media include memes, haikus, or drawings.