Summations

1. $\sum_{i=j}^{k} c = (k - j + 1)c$
2. $\sum_{i=j}^{k}(cf(i)) = c\sum_{i=j}^{k} f(i)$
3. $\sum_{i=j}^{k}(f(i) + g(i)) = \left(\sum_{i=j}^{k} f(i)\right) + \left(\sum_{i=j}^{k} g(i)\right)$
4. $\sum_{i=j}^{k}(f(i)) = \left(\sum_{i=\ell}^{k}(f(i))\right) - \left(\sum_{i=\ell}^{j-1}(f(i))\right)$ **(for any $\ell < j$)**
5. $\sum_{i=j}^{k} f(i) = f(j) + f(j+1) + \ldots + f(k-1) + f(k)$
6. $\sum_{i=j}^{k} f(i) = f(j) + \ldots + f(\ell - 1) + \left(\sum_{i=\ell}^{k} f(i)\right)$ **(for any $j < \ell \leq k$)**
7. $\sum_{i=j}^{k} f(i) = \left(\sum_{i=j}^{\ell} f(i)\right) + f(\ell + 1) + \ldots + f(k)$ **(for any $j \leq \ell < k$)**
8. $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$
9. $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

   $n! \leq c_s n^n$

Inequalities

1. $f(n) \leq g(n)$ is true if you can find some $h(n)$ where $f(n) \leq h(n)$ and $h(n) \leq g(n)$
2. $f(n) \leq g(n)$ is true if you can find some $h(n)$ where $f(n) - h(n) \leq g(n) - h(n)$
3. $f(n) \leq g(n)$ is true if you can find some $h(n) \geq 0$ (for all n) such that $f(n) \cdot h(n) \leq g(n) \cdot h(n)$
4. Take it as a given that: $O(1) \leq O(\log(n)) \leq O(n) \leq O(n^2) \leq O(n^k)$ **(for $k > 2$)** $\leq 2^n$

Logarithms

1. $\log(n^a) = a\log(n)$
2. $\log(an) = \log(a) + \log(n)$
3. $\log(\frac{n}{a}) = \log(n) - \log(a)$
4. $\log_b(n) = \frac{\log_c(n)}{\log_c(b)}$
5. $\log(2^n) = 2^{\log(n)} = n$

Part 1: Data Structure Design (25 points total)

Several production implementations of hash maps use a variant of Hash Tables that we did not discuss in class. In this variant, instead of using a linked list (chaining) to store overflow, each bucket is the root of a self-balancing binary search tree (e.g., one of the balanced binary search tree variants discussed in class). In other words, a hash table with N buckets is composed of N BSTs. A simplified implementation of such a Hash Table is shown below.

```
1  class VariantHashMap[K,V] extends mutable.Map[K,V]
2  {
3    val buckets = Array[TreeMap[K,V]](INITIAL_SIZE)
4    val size = 0
5
6    def resizeIfNecessary(): Unit =
7      if(buckets.size * ALPHA_MAX < size)
8      {
9        /* double buckets.size and rehash */
10     }
11
12   def put(key: K, value: V): Unit =
13   {
14     size += 1
15     resizeIfNecessary()
16     buckets(key.hashCode).put(key, value)
17   }
18
19   def apply(key: K): V =
20     buckets(key.hashCode).apply(key)
21 }
```

You may assume that the `.hashCode` method produces a sufficiently (pseudo)random output and runs in $O(1)$ time.

Question 1 (8 points)

For the `apply` method, list each of the following asymptotic runtimes. Each answer should be in the form of a tight worst-case (i.e., Big-O) bound for each.

1. **The Expected Runtime:** $O(1)$ or $O(\alpha_{max})$ — Each bucket has at most α_{max} (a constant) elements, so the cost of finding an element in any Binary Search Tree is constant
2. **The Amortized Expected Runtime:** $O(1)$ or $O(\alpha_{max})$ — The are no periodic costs to amortize over, so this is just the expected runtime
3. **The Amortized Runtime:** $O(log(n))$ — The are no periodic costs to amortize over, so this is just the worst-case runtime
4. **The (unqualified) Runtime:** $O(\log(n))$ — If all elements are in a single hash bucket, the data structure reduces to being a balanced binary search tree

Answer:

- ☐ (2 pt) The expected runtime is $O(1)$, or $O(\alpha_{max})$, or 'constant'
- ☐ (2 pt) The amortized expected runtime is $O(1)$, or $O(\alpha_{max})$, or 'constant'
- ☐ (2 pt) The amortized runtime is $O(\log(n))$, or 'logarithmic'
- ☐ (2 pt) The unqualified runtime is $O(\log(n))$, or 'logarithmic'

Question 2 (8 points)

For the `put` method, list each of the following asymptotic runtimes. Each answer should be in the form of a tight worst-case (i.e., Big-O) bound for each.

1. **The Expected Runtime:** $O(1)$ — The worst-case behavior is when put needs to rehash the array, which takes linear time; However amortized runtimes average out over expected runtimes.
2. **The Amortized Expected Runtime:** $O(1)$, or $O(\alpha_{max})$ — Rehashings amortize over insertions, so we get constant expected for the fast-path insert, and O(1) amortized for the rehash.
3. **The Amortized Runtime:** $O(\log(n))$ — Rehashings amortize over insertions, so we get amortized constant for the rehash, but without the expected qualifier, we have $O(\log(n))$ to insert into a BST.
4. **The (unqualified) Runtime:** $O(n)$ — Again, without the amortized qualifier, we could need to rehash the array.

Answer:

- ☐ (2 pt) The expected runtime is $O(1)$, or 'constant'; However, since this aspect was not presented clearly in class, $O(n)$ or 'linear' is also accepted.
- ☐ (2 pt) The amortized expected runtime is $O(1)$, or $O(\alpha_{max})$, or 'constant'
- ☐ (2 pt) The amortized runtime is $O(\log(n))$, or 'logarithmic'
- ☐ (2 pt) The unqualified runtime is $O(n)$, or 'linear'

Question 3 (5 points)

Asymptotically (i.e., ignoring constant factors), would the amount of memory used by this table be more than, less than, or about the same as the chaining hash table discussed in class? Why?

Answer:
The same. The number of nodes in a Binary Search Tree is exactly the number of records it stores. Each node in a BST is constant-size.

- ☐ (4 pt) The answer includes a statement that the space consumption is asymptotically the same.
- ☐ (1 pt) The answer justifies this by communicating that the space consumption of a binary search tree is proportional to the number of records, or indicates that the space consumption is the same as the hash table with chaining.
- ☐ (2 pt partial credit) If the answer is given in terms of exact (i.e., non-asymptotic) space complexity.

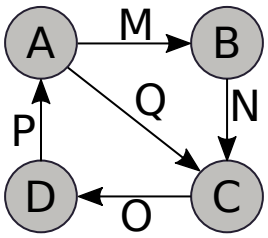
Question 4 (4 points)

Name a specific data structure that implements a Balanced Binary Search Tree (i.e., a data structure that guarantees that its tree will be balanced).

Answer:
An AVL or a red-black tree.

☐ (4 pt) Answer references an AVL or Red-Black Tree

Part 2: Project Review (20 points total)

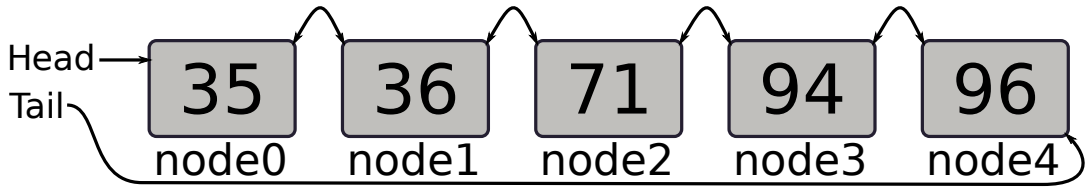


Question 5 (10 points)

PA3 asked you to implement three methods: Two graph traversals, and a method to organize the contents of StreetGraph into a `Map[String,Seq[String]]` for easy access by the other two methods. Illustrate the `Map` that would be generated for the StreetGraph shown above with intersections A, B, C, and D and streets M, N, O, P, and Q

Answer:
$\{A \to [M, Q], B \to [N], C \to [O], D \to [P]\}$

- ☐ (4pt) The answer shows a recognition of the fact that the data structure should allow efficient access to some subset of the edges incident on each vertex.
- ☐ (3pt) The above is correct, and specifically demonstrates a recognition of the fact that the data structure should only include outgoing edges for each vertex.
- ☐ (3pt) The answer is fully correct.



Question 6 (10 points)

Consider the variant linked list you implemented in PA2, an instance of which is illustrated above. List the sequence of value comparisons a correct implementation would perform in response to the method call `update(node2, 20)`

Answer:
71 vs 20, 36 vs 20, 35 vs 20

- ☐ (7pt) The answer shows a recognition of the fact that the list is expected to be in sorted order.
- ☐ (3pt) The answer is exactly correct, or at most off-by-one.

Part 3: Binary Search Trees (20 points total)

Question 7 (5 points)

Draw the Binary Search Tree (not AVL or RB-Tree) that would result after inserting the following elements in the order in which they are given: 81, 19, 91, 71, and 4

Answer:

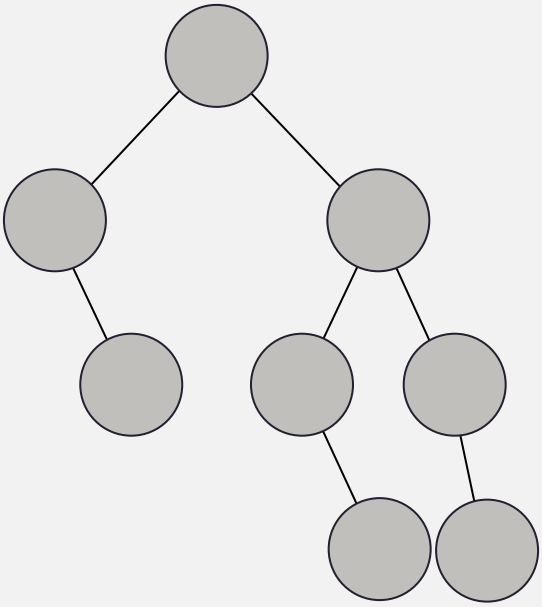
1. 81
    (a) 19
        i. 4
        ii. 71
    (b) 91

☐ (3 pt) An in-order traversal of the tree produces the elements in order. (3 pt partial credit for a minor typo)

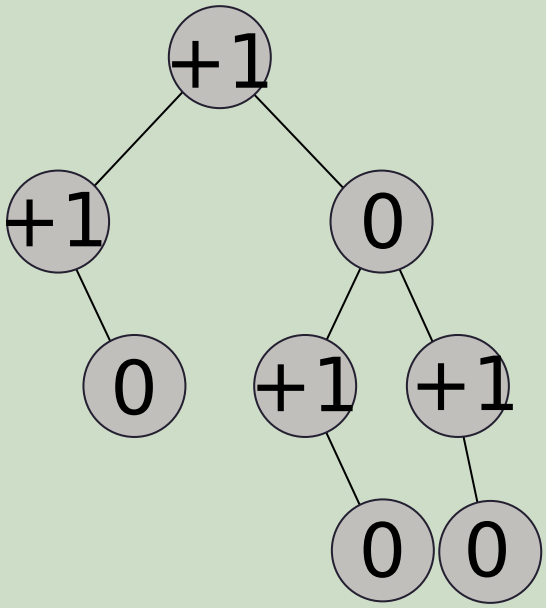
☐ (2 pt) The data structure follows the insertion order.

Question 8 (10 points)

Label every node in the following tree with its balance factor and indicate which nodes (if any) violate the AVL tree property.



Answer:



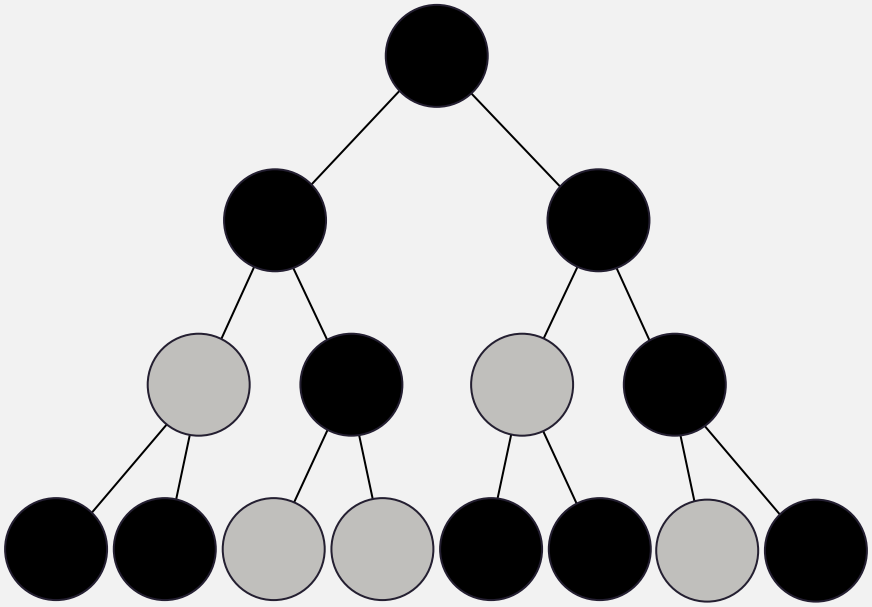
☐ (7 pt) Answer has the correct balance factors (modulo sign)

☐ (2 pt) Answer correctly identifies violations (or the lack thereof)

☐ (1 pt) Answer's balance factors have the correct sign

Question 9 (5 points)

Consider the following tree, with its nodes colored Red (grey) and Black (black). Mark and identify each way in which the tree below violates the properties of a Red-Black tree.



Answer:
A single violation is present: There are 4 black nodes on a path from the root to the rightmost leaf, and only 3 on every other path.

- ☐ (4pt) Answer correctly identifies the violation
- ☐ (1pt) Answer does not identify any incorrect violations

Part 4: Asymptotic Analysis (15 points total)

Consider the following three functions:

1. $f(n) = n^3 + n^2 + n$
2. $g(n) = \sum_{i=0}^{\log(n)} i$
3. $h(n) = \begin{cases} n & \textbf{if } n \mod 5 = 1 \\ n^2 & \textbf{otherwise} \end{cases}$

Question 10 (15 points)

For each function, give (i) a tight lower asymptotic (i.e., big-Ω) bound, (ii) a tight upper asymptotic (i.e., big-O) bound, and (iii) a tight asymptotic (i.e., big-Θ) bound or state that no tight bound exists. Your answers should be a concise description of the applicable complexity class: a single term with no constants.

Answer:
Variant 1

	Big-O	**Big-Ω**	**Big-θ**
$f(n)$	n^3	n^3	n^3
$g(n)$	$\log^2(n)$	$\log^2(n)$	$\log^2(n)$
$h(n)$	n^2	n	n/a

- ☐ (3 pt) At least one of Big-O, Big-Θ, or Big-Ω includes the correct answer for $f(n)$
- ☐ (1 pt each) The remaining two answers for $f(n)$ are correct.
- ☐ (3 pt) At least one of Big-O, Big-Θ, or Big-Ω includes the correct answer for the summation question (either $g(n)$ or $h(n)$).
- ☐ (1 pt each) The remaining two answers for the summation are correct.
- ☐ (2 pt each) Big-O and Big-Ω are correct for the cases-block question (either $g(n)$ or $h(n)$).
- ☐ (1 pt) Big-Θ is correctly marked as n/a, or left blank.

Part 5: Short Answer (20 points total)

Question 11 (5 points)

Recall the hash join algorithm discussed in class, where we combine elements from two datasets on a "join key." The algorithm achieves an expected linear runtime (in the size of both datasets and the result) with a two-phased approach. In the first phase, it builds a hash map over the records of the first dataset according to their join keys. In the second phase, it iterates over the records of the second dataset and looks up the matching records in the hash map. **Propose a way to modify this algorithm so that it has an *unqualified* log-linear runtime (i.e., $O(n \log(n))$).** You may assume that the two input tables have size n and each record in dataset 1 matches with at most one record of dataset 2.

Answer:
Replace the hash map with a map implemented using a balanced binary search tree (e.g., an AVL or Red-Black tree).

- ☐ (5pt) The answer is correct (balanced binary search tree, AVL, Red-Black Tree, or similar).
- ☐ (3pt partial credit) The answer makes some reference to the connection between hash tables / hash maps and expected-constant runtimes.

Question 12 (5 points)

Write down a function $f(n)$ such that $f \in O(n \log(n))$ and $f \in \Omega(\log(n))$

Answer:
Any function in a complexity class between $\log(n)$ and $n \log(n)$ (inclusive) meets the criteria. Standard complexity classes in this range are: $\log(n)$, n, $n \log(n)$, although the way the question is phrased allows for the function to be prefixed by any constant.

- ☐ (5 pt) The answer is any function in a complexity class between $\log(n)$ and $n \log(n)$ (inclusive). Answers given in the form of code or an algorithm are acceptable if the code/algorithm has a runtime in the required bounds.

Question 13 (5 points)

Write down the highest possible number of IO operations (disk reads) needed to find a record in a B+Tree. Your answer should be given as a function of the number of keys per page (k) and the number of records stored in the tree (n).

Answer:
$\log_{k/2}(n)$ ($\log_k(n)$ is also acceptable)

- ☐ (3 pt) The answer is $\Theta(\log(n))$
- ☐ (2 pt) The answer is fully correct (i.e., $\log_{k/2}(n)$ or $\log_k(n)$)

Question 14 (5 points)

Write down the name of one of the course TAs (e.g., one of your recitation instructors).

Answer:
Amelia (Amm), Andrew, Anton, David, Dikshit (DK), Heba, Hope, Jacky, Joey, Kartike, Kyle, Nawar, Riad, Sean, Thinh, Tirth, or Vrund

☐ (5 pt) The answer correctly names a TA.