When discussing Hash Maps, we briefly discussed the Linked Hash Map structure, which combines a Linked List with a Hash Map to provide linear-time (in \( n \), rather than \( N \)) iteration of the map’s contents. The Linked Map structure has a few other useful tricks as well.

Consider the analogous Linked Set structure, summarized below:

```scala
class LinkedSet[A] extends mutable.Set[A]
{
  val insertOrder = DoublyLinkedList[A]
  val contents = mutable.Map[A, DoublyLinkedListNode[A]]()

  def add(a: A): Unit =
  {
    if(contents.contains(a)){ return }
    val node = insertOrder.prepend(a)
    contents.put(a, node)
  }

  def apply(x: A): Boolean =
  {
    contents.contains(x)
  }

  def remove(x: A): Unit =
  {
    contents.remove(x) match {
      case Some(node) => node.removeSelfFromList () /* Remove by position */
      case None => /* no-op */
    }
  }

  def removeOldest = ???
}
```

**Question A.1 (4 points)**

Provide a pseudocode implementation of the `removeOldest(x)` function that removes the oldest item still in the set. This is the item that was least recently added (ignoring duplicate additions). Your implementation should have the same complexity as `contents.remove()`. 

---

**PART A: LINKED SET (10 points total)**

---
**Question A.2 (6 points)**

Assume the variable contents is implemented as a TreeMap or a HashMap (with chaining and a constant $\alpha_{max}$), respectively. Provide tight worst-case, amortized, and expected runtime bounds for the add method for each variant in terms of $n$ (i.e., in terms of contents.size).

<table>
<thead>
<tr>
<th></th>
<th>TreeMap</th>
<th>HashMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst-Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amortized</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expected</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider a Log-Structured Merge Index data structure (i.e., as in PA2) with a buffer size of 2 (i.e., a promote happens every 2 insertions, \(O(2^\ell)\) records at level \(\ell\)). The following keys are inserted into the data structure:

15, 24, 10, 4, 25, 2, 12

Draw the state of the data structure (i) after every promote step, and (ii) its final state.
Question B.2 (10 points)

Campaign contributions are public data. Suppose each record in this data set corresponds to a single donation, and includes at least the following three attributes: (i) the name of a politician, (ii) the amount of money that was donated, and (iii) the name of the donating entity. Write pseudocode for an algorithm to compute the total amount donated to each politician. Assume that each politician has a unique name, and that the name is used consistently throughout the data set. Your algorithm must have a runtime that is linear ($O(n)$) in the number of records. Be sure to note which specific data structure(s) your algorithm uses.
The following questions pertain to the following graph

**Question C.1 (5 points)**

List the nodes visited by Depth First Search in the order in which they are visited, starting from vertex B. Assume that edges are explored in alphabetical order of the opposite vertex. For example, from vertex D, edge DB would be explored before edge DE.

**Question C.2 (5 points)**

Draw the spanning tree produced by DFS on the graph above.
Question C.3 (5 points)

Assume that the graph above is stored using an adjacency list data structure. Draw the vertex list from this data structure. Your illustration does not need to precisely capture the pointer structure of the list contained at each vertex, but should at least state which edges appear in it.
PART D: SETS AND MAPS (10 POINTS TOTAL)

The following items are inserted into a set (in the order given):

30, 20, 48, 36, 98, 68, 85

Question D.1 (3 points)
Show the final structure resulting after the items above are inserted if the set is stored as a general binary search tree.

Question D.2 (3 points)
Show the final structure resulting after the items above are inserted if the set is stored as an AVL tree. Show the tree after each insertion.
Question D.3 (4 points)

Show the final structure resulting after the items above are inserted if the set is stored as a hash table with chaining. Show the structure just prior to every rehash operation. Use an initial capacity of 5 and $\alpha_{\text{max}} = 0.5$, along with the following hash function:

- $hash(20)=69$ ; $hash(30)=22$ ; $hash(36)=39$ ; $hash(48)=27$ ; $hash(68)=72$ ; $hash(85)=49$ ; $hash(98)=61$
PART E: SHORT ANSWER (30 points total)

**Question E.1 (5 points)**

Explain how to create an immutable queue with amortized constant ($O(1)$) time enqueue and dequeue operations. Recall that changes to immutable structures require constructing a new object that encodes the modified version of the data structure. The new object may re-use components of the original version.

**Question E.2 (5 points)**

Show an example of a function $f(n)$ for which both $f(n) \in \Omega(n \log(n))$ and $f(n) \cdot n \in \Omega(n \log(n))$.

**Question E.3 (5 points)**

Consider an implementation of QuickSort that consistently uses the first item in the range being sorted as a pivot. For example, when partitioning the range $[4, 8)$, the value at index 4 would be used as a pivot. Explain how to construct an input that, when used with this implementation of QuickSort, will consistently trigger QuickSort’s worst case runtime of $O(n^2)$. Be sure to explain why the resulting input would result in the worst-case runtime.
**Question E.4 (5 points)**

Say you have an unsorted sequence. You could (i) use a linear scan to find a value, or you could (ii) sort it and then use binary search to find the value. For each of these two approaches, for what sorts of use cases is the approach better than the other. Your response should contrast of the runtime complexity of each approach.

---

**Question E.5 (5 points)**

Assume you have a B+Tree with \( n \) records. Assume that \( C \) is the capacity of both data and directory pages (i.e., the number of key/pointer pairs and the number of records that fit on one data page). Recall that the `texttttrange(low, high)` method on an ordered map returns the records with keys in the range \([low, high]\). Assume that there are \( k \) such keys. What is the worst-case (i.e., big-O) IO complexity, in terms of \( n \) and \( k \), of this method.

---

**Question E.6 (5 points)**

Nearly every set of slides for Dr. Kennedy’s CSE-250 lectures features a picture on the title page that is not directly related to CSE-250 itself. What are the pictures of?
**Question F.1 (15 points)**

For each of the following functions, provide a tight Big-$O$, Big-$\Omega$, and Big-$\Theta$ bound, or indicate that the bound does not exist.

\[
f(n) = 2^{\log_2(4n)} + 19n \log(n) + 12n^2
\]

\[
g(n) = \sum_{i=0}^{n^2} i
\]

\[
h(n) = \begin{cases} 2^n & \text{if } n \text{ is odd} \\ n^2 & \text{if } n \text{ is even} \end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>Big-$O$</th>
<th>Big-$\Omega$</th>
<th>Big-$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(n)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$g(n)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h(n)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>