Analyzing Algorithmic Patterns Based on Real Coding Interview Questions

Ian Dempsey

I. General Patterns for Analysis

A. Palindrome

Palindromes are extremely useful for searching algorithms, as Palindromes are meant to be the same in both directions, one can easily discover if the input is an actual palindrome. This is helpful for searching because one can search and find the odd character out, or the unique piece of data in some text. This style of algorithm is also space efficient, they normally have a space analysis of O(n/2) as the algorithm works over two elements of the input at a time. The basic approach of a Palindrome algorithm is to work inwards with both pointers starting at either end of the input and constantly moving towards one another and comparing if the elements are the same.
The following description is a general algorithm for solving the palindrome problem which is a common problem in Java and other languages. This approach can be used to solve numerous other problems by altering the inside of the loop. Pseudocode:

```
Data: Given input of characters, S
Result: Boolean
initialization;
$leftIndex \longleftarrow$ S[0];
$rightIndex \longleftarrow$ S.length-1;
while leftIndex ¡ rightIndex do
    compare leftIndex with rightIndex;
    if leftIndex !=rightIndex then
        return false;
    else
        1
    end
    eftIndex++;
    rightIndex;
end
return true;
```

Algorithm 1: The Palindrome Algorithm

B. Merge Sort

Merge Sort is a very powerful algorithm. It is more efficient than most styles of insertion, with a time analysis of $O(n * log_n)$, whereas insertion is $O(n^2)$. The idea of merge sort is to divide an array or some input in half and then sort each half before joining it back together. They do not have to be the same size which is useful.
Merge Sort uses the idea of divide and conquer, this means the list to be sorted should be divided up into equal parts first, then these new smaller parts should be sorted individually first before recreating the full list. Pseudocode:

```
Data: List of unsorted data
Result: Sorted List
if length of A is 1 then
    return 1
else
    Split A into two halves , L and R. Repeat until
      size of part =1
    Sort each part individually
    Merge with another subdivided section into B,
      the sorted list
    Return B, the sorted structure
end
```

Algorithm 2: The Merge Sort Algorithm through Recursion

C. Trees

Trees are one of the most powerful styles of data structures for processing data, this is because they allow rapid searching and also fast insetion/deletion of a node. Trees are made up of Nodes, these are just basic Objects which hold some data and have a key . This key allows one to determine where this Node should be in the tree. The important distinction here with these Nodes in comparison to other Nodes used in various data structures, is that these Nodes contain references to children instead of just the next Link. Each Node has exactly one parent, but can many children of their own. There is a special style of Trees known as a Binary Tree. This is a Tree which has between 0 and 2 children. The first Node in a tree is the Root, and it is possible to traverse to any Node in the Tree from this Root Node. With Trees the main function one must take care of is how to traverse them. There are three basic styles of traversal: inorder, preorder and postorder. Inorder visits every Node in ascending order based on their key values. Preorder is where the root is visited first, followed by it's left subtree and then it's right subtree. Finally, postorder is where the left subtree is followed by the right subtree and then the root.
The following is an example of how one might search for a particular Node in a Tree. Pseudocode:

```
**Data:** Given a key to search for
**Result:** The desired Node, or null
initialization;
$Nodecurrent \longleftarrow$root;
while current.data is not key do
    if current is null then
        return null;
    end
    if current.data > key then
        move left on the tree;
    else
        move right on the tree;
    end
end
return current;
```

Algorithm 3: Finding a specific Node in a tree based on the key