Part II Solution

1. Recursive selection sort:

Here, the idea is to use the divide and conquer strategy to express the problem in terms of a problem of smaller size. For selection sort on an array of size n, we can place the smallest element in the beginning of the array and then sort the remaining array of size $n-1$. This leads to the following recursive algorithm.

```
public static void selectionSort( int[] arr, int low ) {
    if ( low >= arr.length )
      return;

    // determine smallest element between indices low
    // and arr.length-1 and place it at the low index
    int min = low;
    for(int i = low; i < arr.length; i++ )
      if ( arr[i] < arr[min] )
        min = i;
    int temp = arr[low];
    arr[low] = arr[min];
    arr[min] = temp;

    // Recursive call to sort the remainder of the array
    selectionSort( arr, low+1 );
}
```

Let $T(n)$ be the number of comparisons performed for an array of size n. From the recursive algorithm above, we see that $T(n)$ satisfies the following recurrence relation:

$$T(n) = T(n-1) + n$$

Using the substitution method, we obtain:

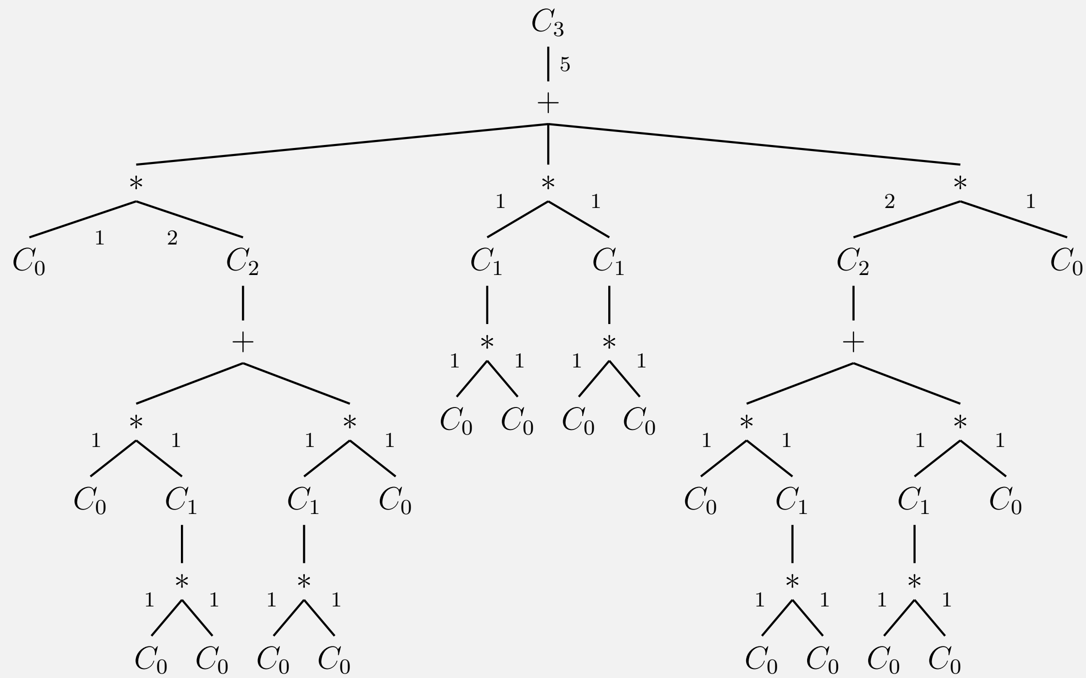
$$T(n) = T(n-2) + (n-1) + n = T(n-k) + \sum_{i=0}^{k-1} n - i$$

For the base case, we have $T(0) = 0$ and is reached when $k = n$, Therefore:

$$T(n) = \sum_{i=0}^{n-1} n - i = \tfrac{1}{2}n(n+1) = \Theta(n^2).$$

2. Catalan number C_3:

A naïve recursive implementation does not store any results and performs redundant computations. From the definition provided, we will obtain the following call tree for C_3.



3. Iterative Substitution:: $T(n) = 2T(n-1) + 1$

This recurrence relation is easily solved by unrolling the recurrence. We have:

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2[2T(n-2) + 1] + 1 = 2^2T(n-2) + 2 + 1 \\ &= 2^2[2T(n-3) + 1] + 2 + 1 = 2^3T(n-3) + 2^2 + 2 + 1 \\ &\vdots \\ &= 2^kT(n-k) + \sum_{i=0}^{k-1} 2^i \end{aligned}$$

The cost of the base case be a, i.e. $T(1) = a$. The base case is reached when $k = n-1$. Thus, we have:

$$T(n) = a2^{n-1} + 2^{n-1} - 1.$$

Therefore, for $a \geq 0$, we conclude that $T(n) = \Theta(2^n)$.

4. Naïve Fibonacci recursive computation:

The Fibonacci sequence F_n for $n \geq 0$ is defined recursively as:

$$F_0 = 0, \quad F_1 = 1;$$
$$F_n = F_{n-1} + F_{n-2}.$$

Let the number of additions be $T(n)$. Using the recursive definition of F_n,

$$\begin{aligned} &T(n) = T(n-1) + T(n-2) + 1, \\ &T(0) = 0, \\ &T(1) = 0. \end{aligned}$$

We need to check that $T(n) = F_{n+1} - 1$ satisfies this recurrence. For the base cases:

$$\begin{aligned} T(0) &= F_{0+1} - 1 = 1 - 1 = 0, \\ T(1) &= F_{1+1} - 1 = 1 - 1 = 0, \end{aligned}$$

and for the recursive case:

$$\begin{aligned} F_{n+1} - 1 &= (F_n - 1) + (F_{n-1} - 1) + 1, \quad \text{(from the recurrence)} \\ F_{n+1} &= F_n + F_{n-1}, \end{aligned}$$

which we know is true from the definition of Fibonacci numbers.