



## 2. SPARC to Python

Consider the following SPARC code of the Fibonacci sequence, which is the series of numbers where each number is the sum of the two preceding numbers. For example, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 ...

```

foo x =
    if x ≤ 1 then
        x
    else
        let (ra,rb) = (foo (x - 1)) , (foo (x - 2)) in
            ra + rb
    end.

```

- 2a. (6 pts) Translate this to Python code – fill in the `def foo` method in `main.py`
- 2b. (6 pts) What does this function do, in your own words?

The function recursively implements the Fibonacci sequence. The base case is when  $x \leq 1$ , as this should result in 1 being returned. The recursive case adds the previous two Fibonacci numbers in the sequence, represented by `ra` and `rb`, and the sum of these is returned by `ra+rb`, and `ra+rb` is the sequence's next term. Running the function `foo(x)` will return the  $x+1^{\text{th}}$  term in the Fibonacci sequence.

### 3. Parallelism and recursion

Consider the following function:

```

def longest_run(myarray, key)
    """
    Input:
        `myarray`: a list of ints
        `key`: an int
    Return:
        the longest continuous sequence of `key` in `myarray`
    """

```

E.g., `longest_run([2,12,12,8,12,12,12,0,12,1], 12) == 3`

- 3a. (7 pts) First, implement an iterative, sequential version of `longest_run` in `main.py`.
- 3b. (4 pts) What is the Work and Span of this implementation?

$$W(n) = O(n)$$

$$S(n) = O(n)$$

Both are just  $O(n)$  complexity because we loop through mylist  $n$  times and perform all  $O(1)$  operations inside the loop. The span is also  $O(n)$  since the algorithm is sequential.

- 3c. (7 pts) Next, implement a `longest_run_recursive`, a recursive, divide and conquer implementation. This is analogous to our implementation of `sum_list_recursive`. To do so, you will need to think about how to combine partial solutions from each recursive call. Make use of the provided class `Result`.
- 3d. (4 pts) What is the Work and Span of this sequential algorithm?

Tree Method

$$\log_2 n \left\{ \begin{array}{l} \text{if } n=1 \\ \frac{n}{2} \quad \frac{n}{2} \\ \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4} \\ |cn(\text{mylist})| = 1 \end{array} \right.$$

$$W(n) = \begin{cases} O(1) & \text{if } n=1 \\ 2W\left(\frac{n}{2}\right) + O(n) \leq 2W\left(\frac{n}{2}\right) + C, & \text{if } n>1 \end{cases}$$

$$\text{Total Cost } \sum_{i=0}^{\log_2 n} 2^i c_i = c \sum_{i=0}^{\log_2 n} 2^i = 2c, n \in O(n)$$

$$W(n) \in O(n)$$

$$S(n) \in O(n)$$

because the algorithm performs constant time comparisons when checking the list for the key, but merging takes  $O(n)$  time. The span of the algorithm is also  $O(n)$  since we are still merging sequentially.

- 3e. (4 pts) Assume that we parallelize in a similar way we did with `sum_list_recursive`. That is, each recursive call spawns a new thread. What is the Work and Span of this algorithm?

The work would stay the same since it is still the same total cost.

$$S(n) = \begin{cases} c_2 & \text{if } n=1 \\ S\left(\frac{n}{2}\right) + c_3 & \text{otherwise} \end{cases} \quad S(n) \in O(n)$$

The span would become  $O(\log n)$  as it is essentially the depth of recursion, and height of tree is  $\log n$  which is the longest dependency.