WEEK 06

1. Preparation for Assignment

If, and *only if* you can truthfully assert the truthfulness of each statement below are you ready to start the exercises.

1.1. Reading Comprehension Self-Check.

- I know why it is **false** to say that *transform-and-conquer* is a group of general problem-solving techniques based on the idea of transformation to a problem that is harder to solve.
- We know why it is **false** to say that there are two principal variations of the transform-and-conquer strategy: *instance simplification* and *representation change*.
- I know that list presorting and Gaussian elimination are examples of instance simplification.
- I know that heaps are most important for the efficient implementation of *priority queues*.
- I know in what sense Horner's rule for polynomial evaluation has useful by-products.
- I know why it is **not quite true** to say that the *fast Fourier transform (FFT)* algorithm is one of the most important algorithmic discoveries of all time.
- I know in what sense Professor X solved his problem by *reduction*. ;)

1.2. Memory Self-Check.

1.2.1. *Applying Representation Change.* Ether 12:27 & 28 teaches us much about Christ, faith, humility, and our weaknesses. It states that Christ makes our weakness into strengths, not us. How might understanding this scripture and The Lord's admonition "In your patience possess ye your souls" change your view of who you are, your relationship with Christ, and how you treat yourself when you succumb to weakness? (FYI, it does not justify the "eat, drink, and be merry" approach to life.)

2. Week 04 Exercises

2.1. Exercise 2 on page 205.

2.2. Exercise 1 on page 216.

Date: October 19, 2018.

2.3. **Exercise 2 on page 225.**

2.4. **Exercise 1 on page 233.**

2.5. **Exercise 5 on page 239.**

2.6. **Exercise 1 on page 248.**

3. Week 06 Problems

Here is a straightforward implementation of a min-max algorithm in a familiar language:

```
1 void sMinMax(int* a, int n, int& min, int& max)
2   {
3       min = max = a[0];
4       for (int i = 1; i < n; i++)
5       {
6           if (a[i] < min) min = a[i];
7           if (a[i] > max) max = a[i];
8       }
9   }
```

What is its efficiency?

The divide-and-conquer recurrence relation for the number of comparisons in the min-max algorithm (shown below in elisp) is:

$T(n) = 2T(n/2) + 2, T(2) = 1.$

Solve this recurrence relation using telescoping (like Bro. Neff did) or backward substitution (like Bro. Barney did). Then answer this question:

Do each of the following two implementations work the same? Compare, through evidence, if any have a slightly-more-efficient-than-straightforward efficiency.

```
1 void rMinMax(int* a, int i, int j, int& min, int& max)
2   {
3       if (i == j)
4           min = max = a[i];
5       else
6       {
7           int mid = (i + j) / 2;
8           int min1, max1;
9
10          rMinMax(a, i, mid, min, max);
11          rMinMax(a, mid + 1, j, min1, max1);
12
13          if (min < min1) min = min1;
14          if (max > max1) max = max1;
15      }
16  }
```

```
(defun compare-min (p1 p2)
    (if (< (car p1) (car p2))
        (car p1)
      (car p2)))

  (defun compare-max (p1 p2)
    (if (> (cdr p1) (cdr p2))
        (cdr p1)
      (cdr p2)))

  (defun r-min-max (array i j)
    (if (= i (1- j)) ;; base case
        (let ((e1 (aref array i))
              (e2 (aref array j)))
          (if (< e1 e2)
              (cons e1 e2) (cons e2 e1)))
      ;; else recurse
      (let* ((mid (/ (+ i j) 2))
             (mm1 (r-min-max array i mid))
             (mm2 (r-min-max array (1+ mid) j)))
        (cons (compare-min mm1 mm2) (compare-max mm1 mm2)))))
```

3.1. **Exercise 1 on page 205.**

3.2. **Exercise 3 on page 216.**

3.3. **Exercise 9 on page 249.**