1 Problem 1

(1) Prove that the solution to the Josephus problem J(k) = 1 (that is, the last person standing is number 1) whenever k = 2^n for any positive integer n.

$k = 2^n$
n = 1:
1 2
1

n = 2:
1 2 3 4
1 3
1

n = 3:
1 2 3 4 5 6 7 8
1 3 5 7
1 5
1

n = 4:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 3 5 7 9 11 13 15
1 5 9 13
1 9
1

We can get the recurrence relation: $J(2^n) = 2 * J(2^{n-1}) - 1$
$J(2^1) = 1$
$J(2^2) = 1$
$J(2^3) = 1$
Assume: $J(2^{n-1}) = 1$
$J(2^n) = 2 * 1 - 1 = 1$

Every round removes all even numbers. Powers of 2 keep the structure symmetrically. So number 1 is never eliminated.

2 Problem 2

(2) Imagine a pile of n large stone discs of distinct sizes as in Towers of Hanoi, but these discs are in a random order, with larger ones possibly on top of smaller ones. (Don't think too hard about stability. :P) You have a machine that can grab a disc and every disc above it in the pile and flip the entire grabbed stack.

(For example, given 3 6 1 2 4 5, with 3 on top, a call to grab(4) would grab 3 6 1 2 and flip that group, producing 2 1 6 3 4 5.)

Design an algorithm that uses calls to grab and results in the pile being in sorted order with the smallest disc on top. What is your algorithm's runtime, assuming grab is the critical operation? (i.e. How many calls to grab do you need, asymptotically?)

This algorithm would follow the same cadence of insertion sort, but with a few changes. Since we grab elements from the top down and reverse them, sorting bottom up makes the most sense to not over complicate the algorithm. The insertion would also differ as we reverse the disc element in each grab call but this would not change how our algorithm runs.

Each iteration we would put the largest element at the top part of the discs, then reverse them into the correct place. We ignore what is unsorted list as we would deal with them later. This requires calling grab(k), where k is how many discs down the largest unsorted element is. Then calling grab(l), where l the stack of discs have been sorted up to plus one. This puts the disc into the correct sorted index, we repeat this process until the whole stack of discs are sorted.

This can be boiled down to; find the largest unsorted element, flip it to be at the top of the disc stack. Then flip again into its destination.

We would call the grab algorithm, $2*n$ times, so the runtime would be $\Theta(n)$, since 2 is a constant

3 Problem 3

(3) Demonstrate the execution of quicksort on the letters of the word PROSECUTIONS, using Hoare's partition method. You should show the new order after each partition. Also, use subscripts to distinguish multiple copies of the same letter. (i.e. Your initial positions should be written as $PRO_1S_1ECUTIO_2NS_2$.) Note that your sort might not be stable.

let p = partition variable
Quick Sort:
$PRO_1S_1ECUTIO_2NS_2$

p = P $PRO_1S_1EC|UTIO_2NS_2$

First Scan:
$IO_2O_1NEC|UTPRS_1S_2$

LHS:
NIO_2O_1EC
p = N $CEI|O_1O_2N$

CEI
p = C
No swaps so we can assume p is in the right space.

EI
p = E
No swaps again, this time and since the list size is two we can assume that both are in the correct space.

O_1O_2N
p = O_1
NO_2O_1
For simplicity we will assume that this is now sorted correctly.

LHS: $CEINO_2O_1$

RHS:
$UTPRS_1S_2$
p = U
$S_2TPRS_1S_2U$
We can assume that U is in the correct space as the list size is 1.

S_2TPRS_1
p = S_2
$S_2S_1PR|T$
Since the right side has a list size of 1 we can assume that it is the right place.
S_2S_1PR
p = S_2
RPS_1S_2

RP
p = P
PR
List size is 2, so we can assume they are in the right place.
S_1S_2
No swaps are needed we can assume they are in the right place.
RHS: PRS_1S_2TU

Final $CEINO_2O_1PRS_1S_2TU$

4 Problem 4

(4) Find the order of growth for solutions to these recurrences. Show your work.
(a) $T(n) = 4T(n/2) + n, T(1) = 1$
a = 4, b = 2, f(n) = n; d = 1
$a > b^d$
$n^{\log_b a} = n^{\log_2 4} = n^2$

$\Theta(n) = n^2$

(b) $T(n) = 4T(n/2) + n^2, T(1) = 1$
a = 4, b = 2, f(n) = n; d = 2
$a = b^d$
$n^{\log_2 4} \log n = n^2 \log n$
$\Theta(n^2 \log n)$
(c) $T(n) = 4T(n/2) + n^3, T(1) = 1$
a = 4, b = 2, f(n) = n; d = 3
$a < b^d$
$\Theta(n^3)$

5 Problem 5

(5) You are given a jumbled-up collection of n bolts of different widths and n corresponding nuts. You can try a nut/bolt combination and determine if the nut is smaller than the bolt, larger than the bolt, or matches precisely, but you may not compare two nuts, nor two bolts. Design an algorithm to match all bolts with their corresponding nuts that has average-case efficiency of O(n log n).

We would do this very similarly to quick sort, but considering we cannot compare like objects we would tweak it a little bit.

Choose a random bolt and preform the partition, nuts smaller go in one pile and nuts bigger go in another. One nut will match exactly. Use the matching nut to partition the bolts. We would continue this just like quick sort, repeat until sorted.

6 Problem 6

(a) How many (recursive) matrix multiplications are needed for this technique?

4 quadrants * 2 multiplications per quadrant = 8.
(b) How many matrix additions are needed for this technique? What is the asymptotic runtime of performing these matrix additions?

Each addition involves matrices $(n/2)^2$. Adding two matrices requires exactly $n^2/4$ additions. But since there are four quadrants when split up the final result would just be n^2

(c) Use your answers from (a) and (b) to write the recurrence relation governing this recursive matrix multiplication technique. Then use the Master Theorem to find its closed-form asymptotic runtime.

Each split of the box creates another 4 boxes. Each of these 4 boxes get multiplied twice, so 8 total calls. The amount of work per layer is $(n/2)^2$, but since there are 4 quadrants we can simplify this to n^2 work per call.

$T(n) = 8T(n/2) + n^2$
Using the master theorem:
a = 8, b = 2, d = 2
$a > b^d$
$\Theta(n^{\log_2 8}) = \Theta(n^3)$

(d) It is possible to accomplish this task with less multiplication and more addition. Specifically, seven mutliplications and 18 additions/subtractions are needed. Write the recurrence relation for this new technique, and use the Master Theorem to find its closed-form asymptotic runtime.

This would make the recurrence relation:
$T(n) = 7T(n/2) + O(n^2)$ a = 7, b = 2, d = 2;
$a > b^d$
$n^{\log_2 7} = 2^{2.81}$
$\Theta(n^{2.81})$