Sample Midterm

(projected time: 120 minutes)

Problem 1:

You are taking pictures of teams at an indoors soccer tournament. The organizer wants the teams to stand in two rows, with one team in front of the other. The organizer also insists that the player standing behind the player is taller than the player in front of them. Before the game, you are given the heights of each team (in cm) as an array $[a_1, a_2, \ldots, a_n]$ and $[b_1, b_2, \ldots, b_n]$. You have a valid placement of both teams if after reordering $a_1 < b_1$, $a_2 < b_2$, …, $a_n < b_n$.

(a) How many possible ways are there to order the two teams (with n players each)?

(b) Show that if the photo is possible, then there is one where the players in the back row are ordered by height, from left to right.

(c) Show that is the photo is possible, then we can have a photo where the players in the front are also ordered.

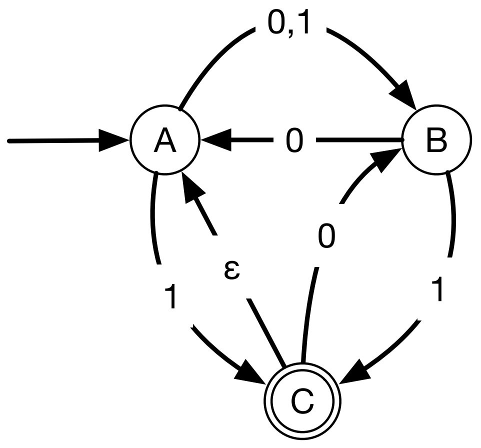
Problem 2:

(a) What is the exact number of comparisons needed to find the simultaneously the maximum and the minimum of four elements?

(b) We want to find simultaneously the maximum and minimum of an array with n elements. We divide the array in groups of four elements plus possibly a group with one, two, or three elements. We determine the maxima and the minima of all groups. Then we use the naive method to find the maximum of the group maxima and the minimum of the group minima. These are the maximum and minimum of the array respectively. What is the exact number of comparisons needed depending on $n \pmod 4$.

Problem 3:

Given the following NFA, create an equivalent DFA. Give the transition diagram of the DFA. (Don't forget the empty set, should it arise). Use double stroke to indicate an accepting state. The initial state is given by the arrow pointing from nowhere.



Problem 4:

What is the recurrence for the run time of the following algorithm in terms of hi-lo (presented in Pseudo-Python)?

```
def alg(array, lo, hi):
     """ array is an array of integers, lo and hi are indices """
     if lo == hi:  #one element in the array
          return array[lo]
     if lo > hi:
          return None
     else:
          suma = 0
          for j in range(lo, hi):
                suma += alg(array(lo, hi-1))
          return suma // (hi-lo)
```

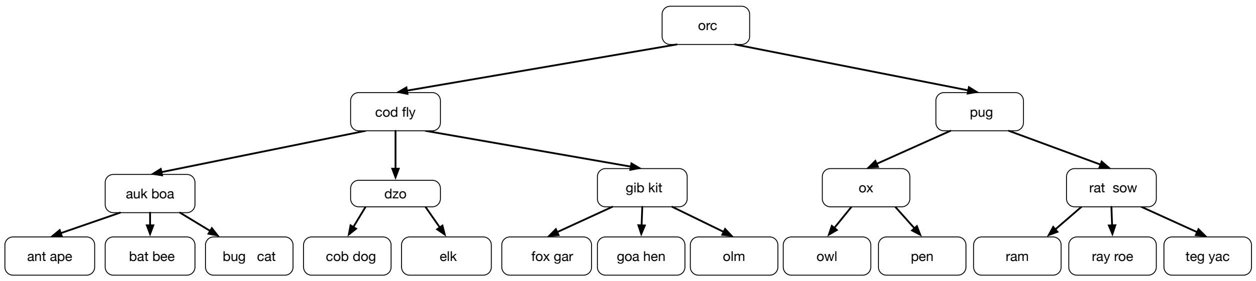
Problem 5:

Apply the Master Theorem on the following recurrences, if possible. Indicate when and why the MT does not apply.

(a) $T(n) = 27T(n/3) + n^2 \log(n)$
(b) $T(n) = 2T(n-1) + 1$
(c) $T(n) = 2T(n/4) + \sqrt{n}$

Problem 6:

From the following B-tree, delete dzo. (For deletes, always choose the predecessor and prefer left rotate over right rotate).



1. First delete 'orc'.
2. Then delete ‘roe' from the resulting 2-3 tree.
3. Then delete ‘rat' from the resulting 2-3 tree.

Explain all steps and show the tree after each deletion.

Problem 7:

Calculate the values for the split pointer and the level of an LH hash table with 10 buckets. Then calculate the buckets were records with hash of key 5, 6, 7, 8, 9, and 10 are inserted.

Extra Problems:

Problem 1:

Assume that a machine has a hardware sorting network — a hardware device built into the CPU just like a floating point adder— that determines simultaneously the maximum and minimum of an array of length up to 8.

1. What is the number of comparisons needed to determine the maximum of *n* elements?
2. How would you use recursion in order to determine the maximum using the sorting network?
3. Give a recursion for the runtime of the maximum finding algorithm.
4. Determine its asymptotic run-time using the Master Theorem.

Problem 2:

A recursive algorithm on an array of n elements is given by the following Python pseudo-code. Use the Master Theorem in order to determine its asymptotic runtime. The brackets are slices and the return value is the concatenation of arrays which takes linear time.

```
def algo(array):
    n = len(array)
```

```
if n<10:
    return sorted(array)
ar1 = algo(ar[0 : 2*n//6])
ar2 = algo(ar[1*n//6 : 3*n//6])
ar3 = algo(ar[2*n//6 : 4*n//6])
ar4 = algo(ar[3*n//6 : 5*n//6])
ar5 = algo(ar[4*n//6 : 6*n//6])
return ( ar1[1:n//6] + ar2[1:n//6] + ar3[1:n//6]+
         ar4[1:n//6]+ar5[1:n//6])
```