Advanced Data Structures
Homework 2

For this homework, you are welcome to use the internet and discuss the problems with others in the class - in fact, I highly encourage this! However, you must write up your own solutions, with NO verbatim copying of other's work. As a general rule, I'd suggest re-writing your solution without looking at any notes from your meetings or webpages, after you understand the solution. If you significantly used any online source, you must also include a reference to that website. Verbatim copying of any webpage will result in a 0 on the homework - use the internet to understand the solution, but ALWAYS rewrite in your own works while NOT looking at the page. (Note that I've googled all of these, so I have a pretty good idea of what's out there...)

I highly recommend using latex for typsetting your entire homework, but I'll accept it hand-written on a case by case basis.

Required Problems

1. Tarjan has shown that, given any splay tree containing the keys $1 \dots n$, if we search for the keys $1 \dots n$ in order then the total cost of all searches is $O(n)$. Show how this implies that, given any two binary trees T_1 and T_2 each on n nodes, the tree T_1 can be converted into the tree T_2 using $O(n)$ rotations.

2. Describe a priority queue that has the working set property. That is, insertions should take $O(1)$ amortized time and, if a call to DeleteMin() deletes the element x, then this should take $O(logt(x))$ time, where $t(x)$ is the number of elements in the priority queue that were inserted after x was inserted. Thus, if we use this priority queue like a stack, the insertions and deletions (pushes and pops) will take constant amortized time.

3. For problem 3, you have two options, so select **one** of the following to answer:

   (a) Short essay/lit search question: Investigate a useful implementation of some variant of B-trees or van Emde Boas trees, in a database system, a file system, or some other real world application, and tell me about it. Some questions to get you started: What is the exact variant of B- or VEB-trees used (B+, B*, etc), and how does it differ from the classical versions we discussed in class? How is it implemented (i. e. what language, what underlying hardware assumptions, or any other interesting aspects you find)? Why do they use that particular variant, and how does it affect the overall efficiency of that implementation? Write an essay summarizing your findings (again 500-ish words, with at least several reputable references).

   (b) More hands on: Go read the following webpages about binary search tradeoffs in memory layout: http://cglab.ca/~morin/misc/arraylayout/ and its followup, as well as the associated paper: https://arxiv.org/abs/1509.05053. Download the tests and run them on your machine, and summarize your own findings. Do they match his results, or agree with earlier work on this, i.e.https://dl.acm.org/doi/10.5555/545381.545386? Write up your results (again 500-ish words), and summarize why there are differences such as this in efficiency between studies. (You're welcome to find other studies, but these should get you started!)