This homework must be completed and submitted electronically. Formatting standards, submission procedures, and (optional) document templates for homeworks may be found at

http://classes.engineering.wustl.edu/cse241/ehomework/ehomework-guide.html

Advice on how to compose homeworks electronically, with links to relevant documentation for several different composition tools, may be found at

http://classes.engineering.wustl.edu/cse241/ehomework/composing-tips.html

Please remember to

- typeset (do not hand-write) your homework’s text;
- create a separate PDF file for each problem;
- include a header with your name, WUSTL key, and the homework number at the top of each page of each solution;
- include any figures (typewritten or hand-drawn) inline or as floats;
- upload and submit your PDFs to Blackboard before class time on the due date.

Always show your work.
The approximate weights of each problem (out of 100%) are given in parentheses, but things that come up in the course of grading may cause these weights to be adjusted. Note that you need only do any four problems out of five to get full credit for this homework, though you may do all five for extra credit.

Unless otherwise noted, all occurrences of “log” refer to the base-2 logarithm.

1. (25%) (Note: this problem is adapted from CLR’s problem 8-6.)

Let $A$ and $B$ be two sorted arrays, each of length $n$. The operation $\text{Merge}(A, B)$ returns a new array of length $2n$ that contains all the elements of both $A$ and $B$ in sorted order. For example, the merge of $\{1, 3\}$ with $\{2, 4\}$ is $\{1, 2, 3, 4\}$. (It should be reasonably clear how to write a divide-and-conquer sorting algorithm around $\text{Merge}$.)

(a) Sketch the decision tree that implements a comparison-based $\text{Merge}$ algorithm on two arrays, each of length 2, using only the $<$ operator. (You don’t have to match any particular merge pseudocode found elsewhere.)

(b) Let $I(n)$ be the number of ways to interleave two arrays of $n$ elements (i.e. the number of possible ways to merge the two arrays without changing the order of either). Argue that

$$I(n) = \binom{2n}{n}.$$

(c) Argue that $\log_2(I(n)) = 2n - g(n)$, where $g(n)$ is some function whose growth is $o(n)$. Hint: the following approximation formula, due to Stirling, will help you take logs of factorials:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\alpha_n},$$

where $\alpha_n$ lies between $\frac{1}{12n+1}$ and $\frac{1}{12n}$.

(d) Suppose we wish to implement $\text{Merge}$ using only comparison operations; that is, the only test we are allowed to do is to compare the values of two elements. Use a decision-tree argument and the result of part (b) to show that any implementation of merging requires $\Omega(n)$ operations.

2. (25%) The basic lower-bound argument we studied in class shows only that there is some input that causes any algorithm for a problem to run slowly. Sometimes, we can prove a stronger result, namely that a problem is hard on average as well as having specific hard cases.

Let $\tau$ be a full binary tree (i.e. each non-leaf node has exactly two children) with $T$ total leaves. We will prove inductively on $T$ that the average depth of all leaves in $\tau$ is at least $\log(T)$.

(a) Give the base case of the proof.

(b) For the inductive step, suppose the two subtrees of the root have $t$ and $T - t$ leaves, respectively. By the inductive hypothesis, these two subtrees have average leaf depths at least $\log(t)$ and $\log(T - t)$, respectively.

Using the bounds on the subtree heights, give a lower bound for the average depth of all paths in the full tree, in terms of $T$ and $t$. (Don’t forget that the root adds one to the depths of all leaves in the subtrees!)

(c) What value of $t$ minimizes your lower bound expression in (b)? Prove it. (Hint: use calculus to find the critical value.)

(d) Plug in the minimum $t$ derived in (c) to show that the inductive hypothesis is satisfied for $T$. 
(e) Using the result proven in (a)–(d), what can you say about the average-case complexity of sorting using comparisons?

3. (25%)

(a) Show the skip list resulting from the following insertions:

- insert(1)
- insert(6)
- insert(2)
- insert(4)
- insert(13)
- insert(9)

Assume that the random number generator returns pillar heights 2, 1, 1, 3, 2, and 1 on successive calls. Assume further that the head and tail pillars are initially of height 1 and are doubled in height as often as needed every time we allocate a node whose height exceeds theirs.

(b) For the last two insertions above, give the complete sequence of keys in the list inspected by the `insert` method. List the keys for each level separately, i.e. “At level 3, we inspect 10, 20, 30, and +∞; at level 2 . . .”

(c) What is the sequence of keys in the list inspected by the operation `find(7)` on the skip list resulting from (a)?

4. (25%) Say that a link between nodes $x$ and $y$ at some level $\ell$ of a skip list “skips” a node $z$ at level $j < \ell$ if $z$ lies between $x$ and $y$ and has height $j + 1$ (hence, its pillar reaches but does not exceed level $j$).

Suppose I construct a skip list that satisfies the following two invariants at every level $\ell > 0$:

1. Every link at level $\ell$ skips at least one node at level $\ell - 1$.
2. No link at level $\ell$ skips more than three nodes at level $\ell - 1$.

(a) Prove that, in a skip list with $n$ elements that satisfies invariants (1) and (2), the worst-case height of the tallest pillar must be $O(\log n)$.

(b) Prove that, in a skip list with $n$ elements that satisfies invariants (1) and (2), a find operation takes worst-case $O(\log n)$ time. (Hint: imagine running the search backwards as we did in our analysis in class. Can use you the invariants to bound the number of horizontal steps at each level?)

(c) The following procedure attempts to dynamically maintain the two invariants above when inserting an item into a skip list. It always assigns a new node height 1, but it may dynamically “elevate” one or more existing nodes during insertion, growing the height of their pillars by 1.
Insert($z$)
allocate a pillar of height 1 for $z$
$\ell \leftarrow \text{head.height} - 1$
$x \leftarrow \text{head}$
while $\ell \geq 0$
do
  $y \leftarrow x[\ell].\text{next}$
  if $y.key < z.key$
    $x \leftarrow y$
  else
    if link $x \rightarrow y$ skips 3 nodes $a \rightarrow b \rightarrow c$ at level $\ell - 1$
      elevate node $b$ to level $\ell$
    $\ell \leftarrow -$ link $z$ between $x, y$ at level 0

Prove that, if a skip list satisfies the two invariants before inserting a node with this procedure, it still satisfies them afterwards. You may assume that the list is initially non-empty, and that the head and tail are initially exactly one level taller than the tallest node in the list.

(Hint: this strategy is similar to the preemptive splitting used to maintain the B-tree invariants on insertion. Formulate a similar inductive argument that shows that the invariant is maintained after processing each level of the list.)

5. (25%) You are designing a B-tree for a system with disk blocks of size 4096 bytes. Each disk block should be one tree node, which must store the following information:

- keys, at 3 bytes per key;
- child pointers, at 4 bytes per pointer;
- some auxiliary data (number of keys in the node, leaf bit, etc) totaling 4 bytes per node.

(a) What is the largest feasible minimum degree $t$ with these storage constraints?

(b) Given a B-tree with three levels (the root plus two more) and the minimum degree computed in (a), what is the largest number of keys that could be stored in this tree?

(c) What is the smallest number of keys that could be stored in the tree from (b)?

(d) Consider a B-tree of minimum degree $t = 2$. Initially, the tree contains the single key $N$. The following keys are then inserted, in the order given:


Show the state of the tree following each insertion. If you split one or more nodes in the course of an insertion, please indicate which node was split.