Exam 2

Work alone. Do not use any notes or books. You have approximately 75 minutes to complete this exam.

Please write your answers on the exam. More paper is available if you need it. Please put your name at the top of the first page.

There are four questions on this exam, for a total of 80 points.

1 The answer is always hash tables (20 points)

Suppose you want to build a grade database that supports the operations INSERT and FIND-MEDIAN, where INSERT adds a new grade to the database and FIND-MEDIAN reports a median grade, defined as a grade x such that the number of grades less than x in the database and the number of grades greater than x in the database differ by at most one. Suppose also that we expect to call FIND-MEDIAN at least as often as we call INSERT.

1. Briefly explain why a hash table is a bad choice for this database.
2. Give an alternative data structure that supports these operations efficiently. You do not need to write any code for this. Describing what data structure you would use, how you would use it, and how much it would cost, is enough.

Solution

1. Because hash tables do not preserve the order of their contents, there is no way to compute the median other than looking at all of the elements, which will take at least $\Omega(n)$ time.
2. There are several options here. We were generally looking for a solution that runs in $O(\log n)$ time (or better) for both INSERT and FIND-MEDIAN.

   (a) An augmented balanced binary search tree, where each node includes a count of the number of nodes in its subtree. This allows ranking and unranking elements at cost $O(\log n)$, and maintains (at the root) a count of all elements, so we can just search for the element with rank $\lfloor n/2 \rfloor$. This gives a cost of $O(\log n)$ for both INSERT and FIND-MEDIAN.

   (b) Two heaps, with the invariant that every element in the left heap (a max-heap) is smaller than every element in the right heap (a min-heap), and that the sizes of the heaps differ by at most one. To perform an insert, put the new element in whichever heap will not violate this invariant. If one heap is too big, perform a DELETE-MIN or DELETE-MAX as appropriate on it and insert the element into the other heap (we will need to do this at most once for every INSERT). The cost of INSERT is thus $O(\log n)$.

   For FIND-MEDIAN, just return the top element in the bigger heap, or in either heap if they are the same size. This takes $O(1)$ time.

   A popular idea was to use an AVL tree without keeping around size fields. Unfortunately AVL trees are height-balanced but only very approximately size-balanced, so there is no way to find the median without extra information.

2 Transposing a two-dimensional array (20 points)

Write a function that takes an argument that is an n-long array of pointers to n-long arrays of `int`s, and transposes it, so that for each `i` and `j`, the value of `a[i][j]` after calling the function is the value of `a[j][i]` before calling the function. We have provided the function header for you:

```
void transpose(int n, int **a)
```

Solution

```
{
    int i;
    int j;
    int tmp;  /* for swapping */

    for(i = 0; i < n; i++) {
        /* only do j < i so we don't swap same pair twice */
        for(j = 0; j < i; j++) {
            tmp = a[i][j];
            a[i][j] = a[j][i];
            a[j][i] = tmp;
        }
    }
}
```

3 Summing a tree (20 points)

Suppose that you have a balanced binary search tree data structure declared as follows:

```
struct tree {
    int key;
    int value;
    struct tree *child[2];
};
```

Write a function `sumTree` that, given a pointer to the root of the tree, computes the sum of the `value` fields in all nodes in the tree (do not worry about overflow). We have provided the function header for you:

```
int sumTree(const struct tree *root)
```

Solution

```
int sumTree(const struct tree *root)
{
    int sum;
    int i;

    if(root) {
        sum = root->value;
        for(i = 0; i < 2; i++) { sum += sumTree(root->child[i]); }
    } else {
        sum = 0;
    }

    return sum;
}
```

4 A twisty program (20 points)

What output does the following program produce? Please draw a rectangle around your solution to distinguish it from any notes you used to produce it.

```
#include <stdio.h>
void f(int n, int ***a, int **b) {
    for(int i = 0; i < n; i++, a++) { *a   = &b[(i+1) % n]; }
}
void g(int n, int  **a, int  *b) {
    for(int i = 0; i < n; i++)      { a[i] = b + ((i+1) % n); }
}
int main(int argc, char **argv) {
    int **x[3]; int *y[3]; int z[3];
    f(3,x,y);   g(3,y,z);
    for(int i = 0; i < 3; i++) { z[i] = i; }
    for(int i = 0; i < 3; i++) { printf("%d ", **(x[i])); }
    for(int i = 0; i < 3; i++) { printf("%d ",  *(y[i])); }
    putchar('\n');
    return 0;
}
```

Solution

```
2 0 1 1 2 0
```