CISC621 Algorithm Design and Analysis, Fall 2015
Homework set 3, due Wednesday, October 28, 5:30pm

Check the "homework sheet" from the syllabus for general homework details. In particular, each homework solution is on an entirely separate (set of) sheet(s) of paper and is identified with your name(s). Do not staple solutions to two or more problems together. Submit in 201 Smith Hall directly to Chunbo Song or place on shelf marked CISC 621.

8. [Individual Problem] Binomial-HeapSort?

   (a) In CLRS section 6.3 the function `Build-Max-Heap` is shown to require at most 2n comparisons in the worst case. Explain why `Build-Max-Heap` requires about $3n/2$ comparisons (give or take a small constant) in the best case and describe, for each n, an input that requires only about $3n/2$ comparisons. For simplicity, you may give your argument and examples just for n's that are one less than a power of 2, so that the tree is a complete binary tree.

   (b) Give (and analyze) a function `Build-Max-Binomial-Heap` that builds a binomial heap from an array of n items using $n - h(n)$ comparisons, where $h(n)$ is the Hamming weight of n, namely the number of 1's in it's binary representation. For example, the Hamming weight of 11 is 3.

   [Food for thought (no written submission expected on this): Do you think a heap sort based on binomial heaps instead of binary heaps would be faster (albeit not in place)?]

9. [Group Problem] Union-Find programs

```
vector<int> wt;
vector<int> parent;

void init(n) { // initialize to n singleton sets
    wt.resize(n+1);
    parent.resize(n+1);
    for (int i = 1; i <= n; ++i) { wt[i] = 1; parent[i] = i; }
}
void union(int a, int b) { // union by weight
// Assume a and b are roots
    if (wt[a] < wt[b]) { parent[a] = b; wt[b] = wt[a] + wt[b]; }
    else { parent[b] = a; wt[a] = wt[a] + wt[b]; }
}
int find(int a) { // return the root of a's set
    if (a == parent[a]) return a;
    parent[a] = find(parent[a]); // path compression
    return parent[a];
}
```

   Above is C++ code for union-find using union by weight and path compression. A union-find program consists of an init(n) followed by m union and find operations. Let us count the number of parent accesses (read or write) made. Init(n) does n accesses. Union does one access. Find(a) does one read and one write per node in the path to the root and one read at the root itself, thus a total of $1 + 2 \times$ depth(a). Consider the class of all union-find programs in which init(n) is followed m operations in which, for some $1 \leq s \leq m$, the first s are unions and the remaining $m - s$ are find's. Show that these programs run in $O(n + m)$ parent accesses.