Assignment 4: Sample solutions and comments

1. (a) Consider the following algorithm:

Algorithm 1 Incremental Un-Dominated(S)

```
1: while |S| > 0 do
2:    find the point p ∈ S with maximum x-coordinate (breaking a tie by
      choosing the point with maximum y-coordinate)
3:    output p
4:    remove from S all points dominated by p
5: end while
```

Each iteration of the while-loop outputs another un-dominated point in S, with successively smaller x-coordinates. The body of the loop has cost $O(|S|)$ since each point of S needs to be considered at most once in each of steps (2) and (4).

(b) Certainly every un-donimated point in S is either an un-dominated point of S_R or an un-dominated point of S_L. Thus $u_R + u_L \geq u$. To show $u_R + u_L = u$ it suffices to note that (i) no point of S_R is dominated by a point of S_L, and hence all un-dominated points of S_R are un-dominated points of S, and (ii) any point of S_L that is not dominated by a point of S_R is either (a) dominated by a point of S_L or (b) un-dominated in S.

(c) Since in the worst case no points are removed in step (iii), we can express this as
$T(n, u) = O(n) + \max_{u_R \leq u}\{T(n/2, u_R) + T(n/2, u - u_R)\}$, provided $n \geq 2$. When $n = 1$, $T(nu) = O(1)$.

(d) Actually, this only makes sense when u (and hence n) is at least 2. Choose c so that $T(n, u) \leq cn + \max_{u_R}\{T(n/2, u_R) + T(n/2, u - u_R)\}$. We can prove that $T(n) \leq cn \lg u$, by induction on n. Suppose the claim holds for $n < n_0$. Then

$$\begin{aligned}
T(n,u) &\le cn + \max_{u_R \le u}\{T(n/2, u_R) + T(n/2, u - u_R)\} \\
&= cn + \max_{u_R \le u}\{c(n/2)\lg u_R + c(n/2)\lg(u - u_R)\} \\
&= cn + \max_{u_R \le u}\{c(n/2)\lg(u_R \cdot (u - u_R))\} \\
&= cn + c(n/2)\lg((u/2)^2)(*) \\
&= cn + cn\lg(u/2) \\
&= cn\lg u
\end{aligned}$$

where the step (*) follows by elementary calculus.

2. (a) Let p_i denote the relative access frequency of key x_i in some long access sequence. We assume that $p_i > \sum_{j=i+1}^{n} p_j$, for $i \le i < n$.

   We claim that the (unique) tree T that minimizes the expected access cost for successful searches with this set of keys is just a chain where x_1 is at the root and, for $1 < i \le n$, x_i is the right child of x_{i-1}. It is easy to confirm that this tree has expected access cost $\sum_{j=1}^{n} j \cdot p_j$, since the access path to key p_j consists of j nodes.

   We prove the optimality of T by induction on n. If $n = 1$ there is nothing to prove. Suppose that the hypothesis is true for $n < k$. Let T be any binary search tree for $\{x_1, \ldots, x_k\}$ with key x_r at the root. Denote by T_L and T_R the left and right subtrees of T.

   Since T_L is a binary search tree in the keys $\{x_1, \ldots, x_{r-1}\}$ and T_R is a binary search tree in the keys $\{x_{r+1}, \ldots, x_k\}$, it follows from the induction hypothesis that $\mathrm{cost}(T_L)$, the expected search cost within T_L is at least $\sum_{j=1}^{r-1} j \cdot p_j$. Similarly $\mathrm{cost}(T_R)$, the expected search cost within T_R is at least $\sum_{j=r+1}^{k} (j - r) \cdot p_j$. Thus, $\mathrm{cost}(T)$, the expected search cost of T is at least $1 + \sum_{j=1}^{r-1} j \cdot p_j + \sum_{j=r+1}^{k} j \cdot p_j$. But $1 = p_1 + \ldots + p_k \ge r \sum_{j=r}^{k} p_j$, since $p_j > \sum_{i=r}^{k} p_i$, for $j < r$. Thus, $\mathrm{cost}(T) \ge \sum_{j=1}^{k} j \cdot p_j$, with equality just when $r = 1$ and T_R is the optimal binary search tree on keys $\{x_2, \ldots, x_k\}$. Thus, the hypothesis holds when $n = k$.

   (b) By the analysis in part (a), it suffices to observe that $\sum_{j=1}^{n} j \cdot p_j$ is maximized, subject to the constraint that $p_i \ge \sum_{j=i+1}^{n} p_j$, for $1 \le i < n$, when $p_i = \sum_{j=i+1}^{n} p_j$, for $1 \le i < n$, which holds just when $p_i = 2^{-i}$, for $1 \le i < n$, and $p_n = 2^{-(n-1)}$.

   Using the fact that $\sum_{j=1}^{n} j \cdot 2^{-j} = 2 - 2^{-(n-1)} - n2^{-n}$, it follows that for this choice of p_i-values, $\sum_{j=1}^{n} j \cdot p_j = 2 - 2^{-(n-1)}$.

   (c) The tree T in part (a) is arguably the most *unbalanced* tree possible on n nodes. However, we can transform it into an *almost balanced* binary search tree T' on the same set of nodes, without increasing the depth of any node by more than one. One way of doing so is to (i) put node $x_{\lceil \lg n \rceil}$ at the root of T', (ii) make the optimal binary search tree on the keys $\{x_1, \ldots, x_{\lceil \lg n \rceil - 1}\}$, the chain described in part (a), the left subtree, and (iii) make any height-balanced binary search tree on the keys $\{x_{\lceil \lg n \rceil + 1}, \ldots x_n\}$, the right subtree.

By this construction, it should be clear that the maximum depth of any node in T' is at most $1 + \lg n$. Furthermore, (i) for $1 \le i < \lceil \lg n \rceil$, $\mathrm{depth}_{T'}(x_i) = \mathrm{depth}_T(x_i) + 1$, (ii) $\mathrm{depth}_{T'}(x_{\lceil \lg n \rceil}) = 0 < \mathrm{depth}_T(x_{\lceil \lg n \rceil})$, and (iii) for $\lceil \lg n \rceil < i \le n$, $\mathrm{depth}_{T'}(x_i) \le \lceil \lg n \rceil + 1 \le i = \mathrm{depth}_T(x_i) + 1$.

So the expected access cost in T' is no more than one greater than the expected access cost in T, and the worst-case access cost is at most $1 + \lg n$.

3. We are considering the following algorithm that outputs a binary sequence that we will interpret as the *encoding* of a positive integer a.

```
                        ▷ phase 1
r ← 1
while a ≥ 2r do            ▷ invariant: a ≥ r
    output 1
    r ← r + r
output 0
                           ▷ assertion: r = 2^⌊lg a⌋

                        ▷ phase 2
low ← r;  high ← 2r;  gap ← r
while gap > 1 do           ▷ invariant: low ≤ a < high = low + gap
    mid ← low + gap/2
    if a < mid
        then do
            output 0
            high ← mid
        else do
            output 1
            low ← mid
    gap ← gap/2
```

(a) Suppose that the input a has the $\lfloor \lg a \rfloor + 1$-bit binary representation: $b_{\lfloor \lg a \rfloor} b_{\lfloor \lg a \rfloor - 1} \ldots b_0$. Note that $b_{\lfloor \lg a \rfloor} = 1$. It suffices to show that the code produced by the algorithm on input a is exactly described as: a sequence of $\lfloor \lg a \rfloor$ 1's, followed by a 0, followed by the $\lfloor \lg a \rfloor$-bit sequence $b_{\lfloor \lg a \rfloor - 1} \ldots b_0$, since either two distinct numbers a and a' have $\lfloor \lg a \rfloor \neq \lfloor \lg a' \rfloor$, in which case their codes differ in their initial sequence of 1's, or their $\lfloor \lg a \rfloor + 1$-bit binary representations differ, in which case their codes differ in their last $\lfloor \lg a \rfloor$ bits.

To prove the above characterization of the code of a, it suffices to argue that the last $\lfloor \lg a \rfloor$ bits are $b_{\lfloor \lg a \rfloor - 1} \ldots b_0$ (by part (a) on the midterm, or the assertion). But this follows from the assertion that after i iterations of the phase 2 while loop, $gap = 2^{\lfloor \lg a \rfloor - i}$, $low = (1b_{\lfloor \lg a \rfloor - 1} \cdots b_{\lfloor \lg a \rfloor - i})_2 \cdot gap$, and $high = [(1b_{\lfloor \lg a \rfloor - 1} \cdots b_{\lfloor \lg a \rfloor - i})_2 + 1] \cdot gap$.

(b) The *decoding* procedure (taking the encoding of a and returning a) should be clear from the characterization in part (a): simply prepend the last $\lfloor \lg a \rfloor$ bits of the code with a 1 (the value $\lfloor \lg a \rfloor$, of course, is given by the length of the initial string of 1's).

(c) Since the phase 1 of the standard encoding algorithm outputs a sequence of $\lfloor \lg a \rfloor$ 1's, followed by a zero, we could view this as the unary encoding of the number $length = \lfloor \lg a \rfloor$. So we create a more compact encoding of *length* using the standard encoding and follow this with phase 2.

```
                      ▷ phase 0
r ← 1
length ← 0
while a ≥ 2r do            ▷ invariant: a ≥ r
    length ← length + 1
    r ← r + r
                           ▷ assertion: length = ⌊lg a⌋ & r = 2^⌊lg a⌋

                      ▷ phase 1
encode length  using the standard encoding
                      ▷ phase 2
low ← r;  high ← 2r;  gap ← r
while gap > 1 do           ▷ invariant: low ≤ a < high = low + gap
    mid ← low + gap/2
    if a < mid
        then do
            output 0
            high ← mid
        else do
            output 1
            low ← mid
    gap ← gap/2
```

The resulting encoding of a has length $1 + 2\lfloor \lg \lfloor \lg a \rfloor \rfloor + \lfloor \lg a \rfloor$.