PSU CS 350 Quiz Summer 2015 **Exam Key**

Before you start, ***write your name at the top of each page****. Enough space should be given for each solution, but if not then indicate this and continue on the back.*

I suggest that you read the entire exam before you start. If you find a problem with the exam, please note it in your answer and answer as best you can. Please show as much of your work as you reasonably can.

1. We are given a (one-based) input array a of integers. We want to search for a smallest k such that both k and $k+1$ are present in a. We would like to find the leftmost index of such a k if one exists, and fail otherwise. Formally:

   Leftmost Min Pair

   Instance: An array a of n integers.

   Solution: The minimum i such that k is the smallest integer such that

   $$a[i] = k \ \wedge \ \exists j \ . \ a[j] = k + 1$$

   or failure if no such i exists.

   For example, given an array

   $$[1, 9, 4, 8, 3, 5, 6, 11, 3]$$

   the smallest such k is 3 and its leftmost index position i is 5.

   Give pseudocode for a polytime algorithm for finding k (or failing if it does not exist), and analyze its big-O worst-case asymptotic running time.

```
To find a smallest leftmost k in an array a of size n:
    b ← sort a
    for j ← 0 .. n − 2
        if b[j+1] = b[j] + 1
            for k ← 1 .. n
                if a[k] = b[j]
                    return k
    fail
```

This algorithm is $O(n \log n)$ given a sort of this complexity; the search takes time $2n - 1 \in O(n)$. There's also the more brute-force $O(n^2)$ algorithm, which is fine.

2. Suppose that you want to extract the item at position i of a Heap of n items (the i^{th} element of the array) rather than extracting a best value?

   (a) Give pseudocode for the extraction in terms of UpHeap and DownHeap.

```
e ← H[i]
H[i] ← H[n]
shrink H by 1
if H[i] < H[parent(i)]
    upheap H[i]
else if H[i] > H[left(i)] or H[i] > H[right(i)]
    downheap H[i]
return e
```

   (b) What is the worst-case big-O asymptotic complexity of your algorithm? Justify your answer.

$O(\lg n)$ since in the worst case the heap is traversed from bottom-to-top or top-to-bottom.

3. Consider the following sorting algorithm:

Mystery Sort

To *merge* two arrays a and b:
```
c ← new array of |a| + |b| elements
i ← 1
j ← 1
k ← 1
while i ≤ |a| and j ≤ |b|
    if j > |b| or (i ≤ |a| and a[i] < b[j])
        c[k] ← a[i]
        i ← i + 1
    else
        c[k] ← a[j]
        j ← j + 1
    k ← k + 1
return c
```

To *msort* an array a:
```
if |a| ≤ 1
    return a
m ← (|a| + 2) div 3
b1 ← msort a[1..m]
b2 ← msort a[m+1..2m]
c1 ← merge b1 and b2
b3 ← msort a[2m+1..|a|]
c2 ← merge c1 and b3
return c2
```

Assuming constant-time exchanges and arithmetic, what is the asymptotic big-Θ worst-case complexity of this algorithm when calling *msort* on an array of n integers, where n is a power of 3? Justify your answer.

$O(n \lg n)$. This is Case 2 of the Master Theorem as handed out with the exam, since $a = 3$, $b = 3$ and $c = 1$.

4. Consider this pseudocode for Prim's Algorithm.

Prim's Algorithm

```
To compute a MWST for a graph (V,E):
    Q ← empty min-heap of edges
    T ← empty tree
    remove an arbitrary vertex v from V
    mark v
    insert edges neighboring v into Q
    n ← |V|
    while n > 0 and Q is not empty
        (w, v1, v2) ← extract-min Q
        if v1 is marked and v2 is marked
            continue
        insert (v1, v2) into T
        v3 ← whichever of v1 and v2 is unmarked
        mark v3
        for v in neighbors(v3)
            if v is unmarked
                insert the weighted edge from v3 to v into Q
        n ← n − 1
    return T
```

(a) For a connected graph with v vertices and e edges, exactly how many edges should the minimum-weight spanning tree produced by the algorithm have?

$v - 1$

(b) For a connected graph with v vertices and e edges, what is the worst-case big-O asymptotic running time for this algorithm?

$O(e \lg e)$, or since e is at most v^2, $O(e \lg v)$

5. The following proposed hashes are intended to be applied to an array w of five 8-bit values, producing a 16-bit result. They are all terrible. For each proposed hash, explain what is wrong with it.

   (a)

   $$h(w) = \text{concatenate } w[2] \text{ and } w[5]$$

   Doesn't even take three of the five characters into account, so distributes values horribly.

   (b)

   $$h(w) = \left(\sum_{i=1}^{100} (w[i \mod 5 + 1])^{17} \mod (2^{16} - 1) \right) \mod 2^{16}$$

   Quite slow to compute.

   (c)

   $$h(w) = \left(\sum_{i=1}^{5} w[i] \cdot \mathit{random}(1 \ldots 2^{16} - 1) \right) \mod 2^{16}$$

   Not actually a function. Returns a different answer on each call with the same input.

6. The Halfonacci Function is defined by

$$\begin{aligned} H(1) &= 1 \\ H(2) &= 1 \\ H(n) &= H(n \operatorname{div} 2) + H(n-1) \end{aligned}$$

Give a dynamic programming algorithm for computing $H(n)$ in time linear in n.

To compute the nth Halfonacci Number:
    $H \leftarrow$ new array of n elements
    $H[1] \leftarrow 1$
    $H[2] \leftarrow 1$
    for i **in** $3..n$
        $H[i] \leftarrow H$[i-1] + $H[i$ **div** 2]
    return $H[n]$