Data Structures and Algorithms

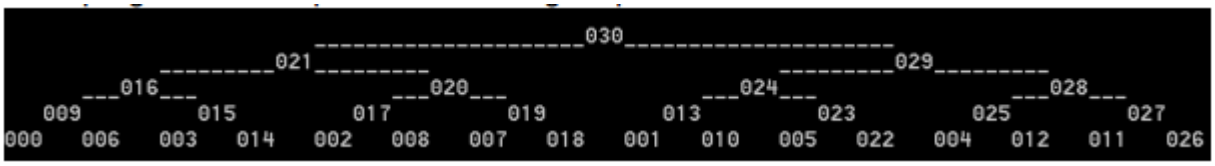
Lab 7 – Heap & Hash

Question 1

Printing a heap as a normal tree:
Write a program that prints a heap as a normal tree. For example, with the heap:

```
int size = 31;
int *maxHeap = new int[size];
for (int i = 0; i < size; i++) {
    maxHeap[i] = i;
}
buildHeap(maxHeap, size);
```

Your program should print the following output



Here is another example

Note: for printing purpose, we assume that our heap only consists of integers ranged from 0 to 999 or else there will be displacement problem. If a node is less than 100, you will have to pad zeros in front of it so that it has exactly tree characters

Question 2

Binary search tree to heap: Write a function to convert a binary search tree to a max heap.

```
BST insersion:
Insersion #0: 473
Insersion #1: 309
Insersion #2: 286
Insersion #3: 598
Insersion #4: 438
Insersion #5: 663
Insersion #6: 948
Insersion #7: 424

Our tree after the insersion process (BST pritning code is resued from lab 4):
((286)309((424)438))473(598(663(948)))
Number of nodes: 8

Your goal is to construct the following heap:
                    _________948_________
          ___663___                   ___598___
   473           438           424           309
286
```

Question 3

Implement the following hash functions:

a. $f_1(key) = (7 \times key[0]^0 + 7 \times key[1]^1 + ... + 7 \times key[end]^{end}) \% 7719$
b. $f_2(key) = (7 \times key[0]^{key[0]} + 7 \times key[1]^{key[1]} + ... + 7 \times key[end]^{key[end]}) \% 7719$

Question 4

a. Find errors in the following functions and fix them:

```
void reheapUp(int *maxHeap, int position){
    if (position > 0) {
        int parent = (position - 1) / 2;
        if (maxHeap[position] < maxHeap[parent]){
            reheapUp(maxHeap, position);
            swap(maxHeap[position - 1], maxHeap[parent]);
            position = parent;
        }
    }
}

void reheapDown(int *maxHeap , int position, int size) {
    int leftChild = position * 2 + 1;
    int rightChild = position * 2 + 2;
    int largeChild = -1;
    if (leftChild >= size - 1) {
        if (rightChild >= size-1 && maxHeap[rightChild] >
```

```
maxHeap[leftChild]){
            largeChild = rightChild;
        }
        else {
            largeChild = leftChild;
        }
        if (maxHeap[largeChild] < maxHeap[position]) {
            swap(maxHeap[largeChild], maxHeap[position]);
            reheapDown(maxHeap, largeChild, size);
        }
    }
}
```

b. Write a function to convert a max heap tree to a min heap tree.