CSCD320 Homework5, Spring 2014 Eastern Washington University. Cheney, Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, May 11, 2014 (Sunday)

Please follow these rules strictly:

1. Write your name and EWUID on **EVERY** page of your submission.

2. Verbal discussions with classmates are encouraged, but each student must independently write his/her own solutions, without referring to anybody else's solution.

3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Blackboard system). Send in whatever you have by the deadline.

4. Submission must be computer typeset in the **PDF** format and sent to the Blackboard system. I encourage you all to use the LATEX system for the typesetting, as what I am doing for this homework as well as the class slides. LATEX is a free software used by publishers for professional typesetting and are also used by nearly all the computer science and math professionals for paper writing.

5. Your submission PDF file must be named as:

firstname_lastname_EWUID_cscd320_hw5.pdf

(1) We use the underline ' ' not the dash '-'.
(2) All letters are in the lower case including your name and the filename's extend.
(3) If you have middle name(s), you don't have to put them into the submission's filename.

6. Sharing any content of this homework and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

CSCD320 Homework5, Spring 2014 Eastern Washington University. Cheney, Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, May 11, 2014 (Sunday)

Problem 1 (20 points)**.** *Let* A[1 . . .n] *be a max-heap. The operation* `max-heap-delete(A,i)` *deletes the heap element* A[i] *from the heap, so that the rest after the deletion is still a max-heap. Please gives the pseudocode of* `max-heap-delete(A,i)` *and its time complexity in big-oh notation and make your bound as tight as possible. Hint: it's a bit more complicated that inserting a new key, and you can use the subroutines in the books if they are useful.*

```
max-heap-delete(A,i)
{
    A[i] = A[n];
    A.heap_size--;

    if(A[i/2] < A[i])
    {
        while(A[i/2] < A[i] && i > 1)
        {
            temp = A[i/2];
            A[i/2] = A[i];
            A[i] = temp;
            i = i/2;
        }
    }
    else
    {
        Max_Heapify(A,i);
    }
}
```

In the worst case, this algorithm will have to traverse the entire height of the logically derived heap structure, and make the constant time cost at each step of swapping data values of two indexes in the array. It follows then, that the time complexity of this is O(log n).

CSCD320 Homework5, Spring 2014 Eastern Washington University. Cheney, Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, May 11, 2014 (Sunday)

Problem 2 (20 points)**.** *[Exercise 6.5-9 of CLRS, 3rd Ed.] Give an* O(n log k)*-time algorithm to merge* k *SORTED lists into one sorted list, where* n *is the total number of elements in all the input lists. Give your algorithmic idea and pseudocode and explain why your algorithm has a time cost of* O(n log k)*. Explain how you will use your algorithm for the* `mergesort` *if the* `mergesort` *has a setting of k-way splitting.*

Assuming that all k lists are sorted in ascending order, I would take the values found at the first index of each list and build a min heap of nodes. Each node would store a value, what list the value came from, and what index in the list. I would then use Min-Heap-Extract-Min to remove the smallest value of the heap and put it the first position of a new list; and also record what list and index the value originally came from. I would then take the value stored in the next index of that recorded list, and insert it into the min heap. Then I would use Min-Heap-Extract-Min to start the process again, and repeat it until the heap is empty; also making sure not to accidently exceed the maximum indexes of each list. When this algorithm is finished, the new list will contain all n elements sorted in ascending order.

```
Merge(lists[k][ ])          //k sorted lists
{
    heap[k]                 //Stores values with corresponding lists and indexes in a min heap of nodes
    cnt = 0                 //Index in the sorted result list
    resultList[n]           //Final sorted result list

    //Make heap from the first elements of each list
    for (i = 0; i < k; i++)
        heap.insert(lists,i, 0)             //Insert value at first index of each list, and store original location

    //Remove the minimum, and insert next value(if possible), until heap is empty
    while(!heap.empty())
    {
        k,i = Min-Heap-Extract-Min();  //Performs delete min operation, and return original location
        result[cnt] = lists[k][i]          //Add to result list
        cnt++ ;                            //Increase result list index
        i++;                               //Prep to get next value from list k
        if (i < lists[k].length)           //If not at end of list k
            heap.insert(lists,k,i)         //Insert next value
    }
    return resultList;                     //Done
}
```

Since insertion takes O(log k) time, the first for loop will take O(k log k) time. The while loop will run O(n) times, and inside insertion is called again, along with Min-Heap-Extract-Min which also takes O(log k) time; so the while loop altogether will take O(n 2log k) time. Both loops together take
O(k log k +n 2log k) = O(n log k). So, the algorithm takes O(n log k) time.

CSCD320 Homework5, Spring 2014 Eastern Washington University. Cheney, Washington.

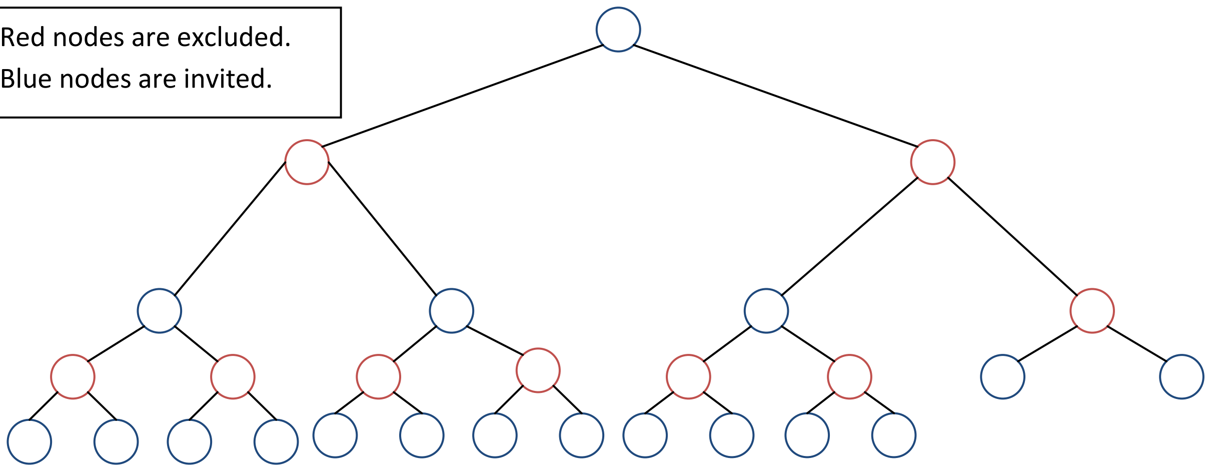
Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, May 11, 2014 (Sunday)

Problem 3 (20 points)**.** *Planning EWU's end-of-year party: Everyone (including all the faculty, staff, and students) at EWU has one and only one direct supervisor except the president, so the supervisorship among the people of EWU can be represented as a rooted tree where a node is the direct supervisor of its child nodes (if the node has child nodes) and the president is the root of this tree. Now the HR office of EWU's end-of-year party for the EWU community. In order to let everyone relax and have fun, all the people invited for the party should not have their direct supervisors invited. Now you, as the direct of EWU HR, are given the supervisorship tree and need to decide who should be invited. Your goal is to invite as many people as you can, as long as everyone can relax. Describe your strategy for picking the guests and explain why your strategy works. [Hint: It seems that you all should be invited and I should not, assuming I am your direct supervisor and you all do not have supervisees.]*

Given a tree structure where nodes represent people, and parent nodes are direct supervisors of their children nodes, we want to include as many nodes as possible without having any direct parent/child relationships between included nodes; that way as many people are invited to the party as is possible, without any direct supervisors of those people present. For my solution to this problem, I would start at the bottom of the tree and include every leaf node. Then, I would skip the next layer in the tree that is comprised of the parents of those leaf nodes, and move up to the layer beyond and include every node in that layer. I would continue this process of skipping a layer and taking the next, until there are no more layers in the tree. If the tree has an odd number of layers, then the root layer will be included; if the number of layers is even, then the root layer will be skipped. If we start with the leaf node layer(layer with the most nodes), then employ this method, it will insure that that maximum number of nodes is included, without including any direct parent nodes of the included nodes.

Example:

Red nodes are excluded.
Blue nodes are invited.



CSCD320 Homework5, Spring 2014 Eastern Washington University. Cheney, Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, May 11, 2014 (Sunday)

Problem 4 (20 points)**.** *You are given a set of* n **unit-length** *tasks. Each task* i, *for* $1 \leq i \leq n$, *can start at the earliest possible time* r_i *and must finish by its deadline* d_i. *You are asked to schedule a maximum number of tasks within the given constraint, so that all the picked tasks can be scheduled without conflicts. Describe your strategy on picking what tasks and how to schedule them. Explain why your strategy works.*

First, I would sort the list of tasks from the task with the earliest deadline to the task with the latest deadline. Then, I would start a count at zero to keep track of the total unit length of tasks I will be scheduling. Next, I would schedule the task with the earliest deadline first and add its unit length to my total unit length count. Then, I would move on to the task with the next earliest deadline, if its unit length added to my count is less than its deadline, then I would schedule it next and add its unit length to my total count. If its unit length added to my count is greater than its deadline, then the task will be skipped and its length will not be added to my count. Then, I will move onto the task with the next earliest deadline and perform the same check and procedure. I will do this for every task in the list. When I am finished, the tasks I have scheduled should be the maximum number of tasks that can be scheduled together, within the given constraint. Basically, if I keep track of the total unit length of scheduled tasks, while scheduling the remaining tasks in the order of earliest deadlines that can still make their deadlines, the resulting end schedule should be optimal for scheduling as many as possible tasks for completion that I can.

Problem 5 (20 points)**.** *Suppose you are given a text of* 375 *characters drawn from the alphabet* {`a, b, c, d, e, f, g, h`} *and the frequency of each letter is:*

a	b	c	d	e	f	g	h
40	30	90	20	60	120	10	5

1. Create a Huffman tree for this text (you may have multiple different Huffman tree for this text, but anyone is fine).

2. Show the Huffman code of each letter.

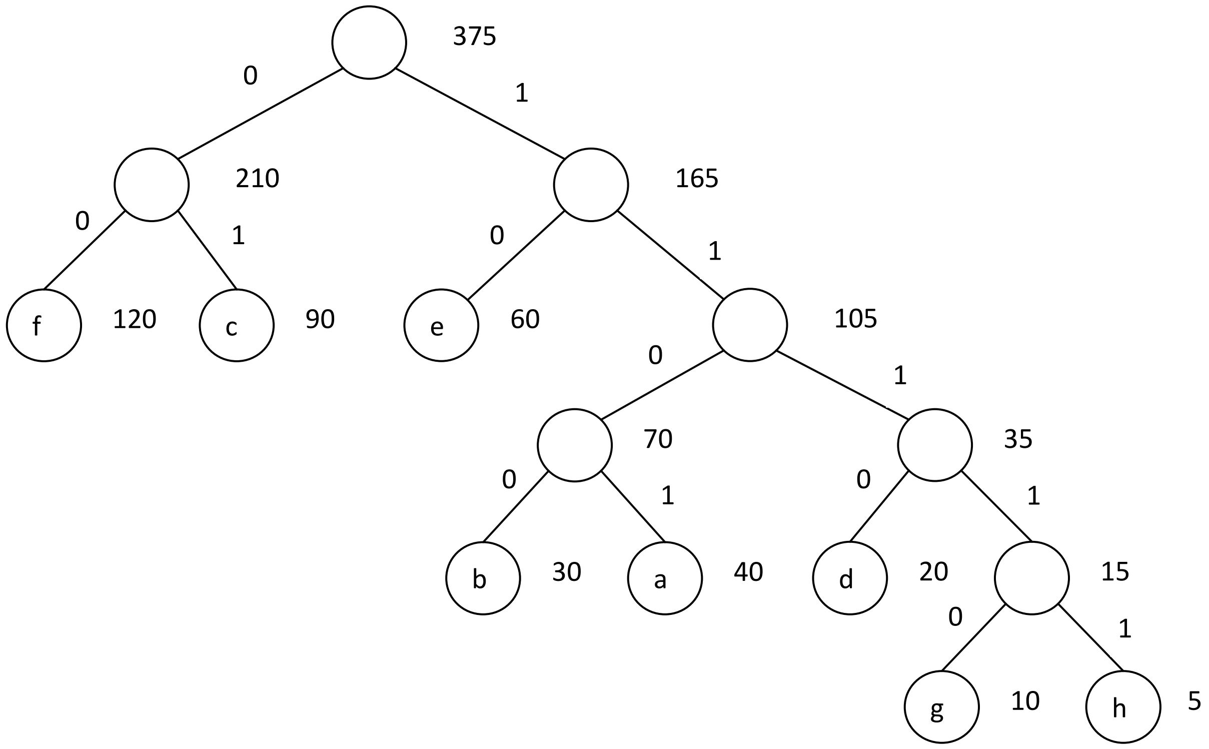
3. Computer the size of the Huffman code compressed version of this text in bits.

4. Calculate the compression ratio: compressed text size / raw text size, if you use 8-bit ASCII code for storing the raw text.

CSCD320 Homework5, Spring 2014 Eastern Washington University. Cheney, Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, May 11, 2014 (Sunday)

Huffman Tree:



Huffman Codes:

a: 1101	b: 1100	c: 01	d: 1110
e: 10	f: 00	g: 11110	h: 11111

The size of the Huffman code compressed version of this text is (270*2 + 90*3 + 15*5)-bits, which calculates to 855-bits.

The original size of the text with 8-bit ASCII code is (375*8)-bits, which calculates to 3000-bits. So the compression ratio is $\frac{855}{3000}$, which reduces to $\frac{57}{200}$.