CSCI 33500 - Spring 2016
Final Exam.

in class May 23rd

Instructions

- Write your full name on the front page.
- All sections are worth the same amount of points.
- Use the box under the questions to write clear and concise, but comprehensive, answers.
- No outside material or electronics are allowed.
- Use a black or blue pen.

1 Induction - Trees

A perfect binary tree is a binary tree in which all interior nodes have two children and all leaves have the same depth.

1.1 Each depth of a perfect binary tree has double the number of nodes than the depth above. Prove *by induction* that the total number of nodes is: $\sum_{i=0}^{D} 2^i = 2^{D+1} - 1$ where D is the depth of the leaves of the tree.

1.2 The root of a tree has depth zero. Prove *by induction* that the sum of the depths of all the nodes in a perfect tree is: $\sum_{i=0}^{D} 2^i i = 2^{D+1}D - 2^{D+1} + 2$ where D is the depth of the leaves of the tree.

For each proof, state clearly your base case, what is the next step of the hypothesis you are aiming to prove, and when you apply the induction hypothesis in the proof.

Answers:

1.1 See midterm.

1.2

$$\begin{aligned}\sum_{i=0}^{D+1} 2^i i &= 2^{D+1}(D+1) + \sum_{i=0}^{D} 2^i i \\ &= 2^{D+1}(D+1) + 2^{D+1}D - 2^{D+1} + 2 \text{ (by the induction hypothesis)} \\ &= 2 * 2^{D+1}D + 2^{D+1} - 2^{D+1} + 2 \\ &= 2^{D+2}(D+1) - 2^{D+2} + 2\end{aligned}$$

Q.E.D.

2 Min-Heap

2.1 What are the two properties that define a binary tree as a min-heap?

2.2 Consider the following min-heap, stored in array form: `[1,10,5,17,19,36,7,25,20]`. Draw the corresponding tree representation of this min-heap.

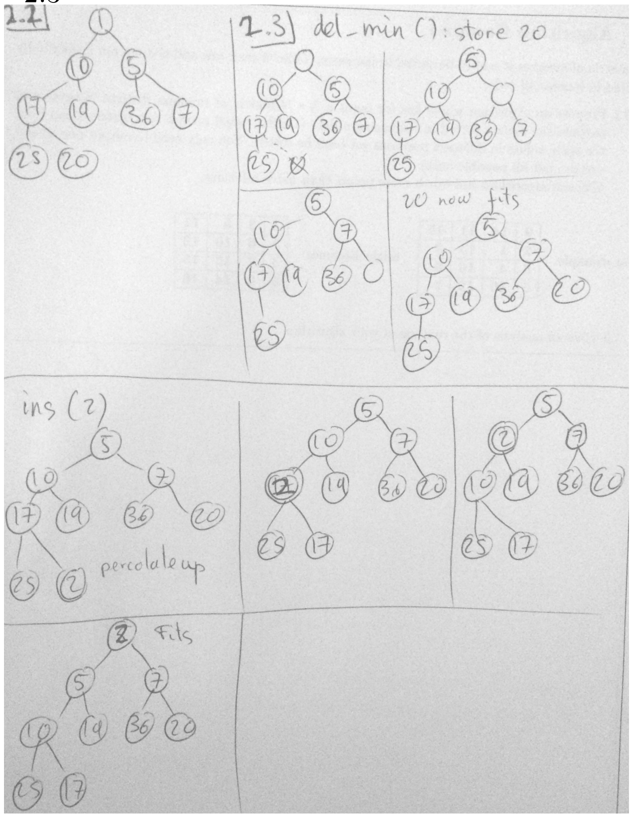
2.3 Show the steps and final result of performing the following operations: `delete_min(), ins(2)`

2.4 What is the worst case complexity of the `delete_min(), ins(x)` operations in a min-heap with N nodes?

Aswers:

2.1 properties: complete binary tree and heap-order (node value smaller than children).

2.2 - 2.3



2.4 `delete_min(), ins(x)` are both $O(log(N))$ operations.

3 Algorithm Creation

A matrix of integers is said to be sorted in increasing order if each row and column are individually sorted in increasing order.

3.1 Propose an algorithm which has for input a $N * N$ matrix of random, distinct integers and outputs the matrix sorted in increasing order. (Unlike sorted arrays, two matrices can hold the same values in different positions yet both be sorted. You only need to output one sorted matrix, not all possible ones)
Efficient algorithms are worth more points than naive solutions.

For example

9	**14**	**11**	**15**
2	**1**	**13**	**3**
8	**4**	**16**	**6**
5	**10**	**12**	**7**

might becomes

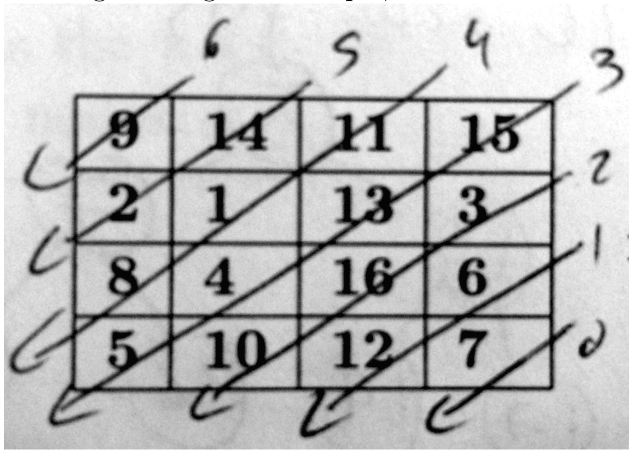
1	**3**	**9**	**11**
2	**6**	**10**	**12**
4	**7**	**13**	**15**
5	**8**	**14**	**16**

3.2 Give an analysis of the runtime of your algorithm.

Answers:

3.1 - 3.2
Naive solutions included:
- sorting sorting all the elements first, then inserting them into the matrix line by line. This is $O(N * N * Log(N * N)) = O(N^2 * Log(N))$.
- sorting each row, then each column, which requires a proof that it is correct. This also is $O(2 * N * N * Log(N)) = O(N^2 * Log(N))$.

An optimal solution is to apply the same logic as the `build_heap()` procedure.
Percolate values down+right, starting from the bottom right and looking at each diagonal.
Looking at the given example, elements would be traversed slice by slice this way:



For the same reason that `build_heap()` is linear, this algorithm would be linear in the size of the matrix, so $O(N * N)$.

In pseudocode:

```
sort_matrix(M)
{
   N = M.numrows()
   //now we need to traverse the matrix slice by slice, which is non-trivial
   for(int slice = 0; slice < 2 * N - 1; slice++)
   {
       int z = slice < N ? 0 :  slice - N + 1; //ternary operator
       for (int j = z; j <= slice - z; j++)
       {
           for (int j = (slice - z); j >= z; --j)
           {
               //we now almost have the row and column
               row = (n-1)-j
               col = (n-1)-(slice - j)
               //the down+right percolation is left as an exercise
               percolate_down_right(M, row, col)
           }
       }
   }
}
```

4 Divide and Conquer

Consider the following pseudocode for a mystery procedure, which takes for input a *sorted* matrix, as defined in section 3.

```
bool mystery_proc(matrix<int> M, int x)
{
    if M.numrows() == 1 and M.numcols() == 1
    {
        if x == M[0][0]
        {
            return true;
        }
        else
         {
             return false;
         }
     }
     else
     {
         int mid_row = M.numrows()/2-1;
         int mid_col = M.numcols()/2-1;
         if x == M[mid_row][mid_col]
         {
             return true;
         }
         else if x < M[mid_row][mid_col]
         {
             matrix<int> upper_left_quad = submatrix(M,0,0, mid_row,mid_col);
             return mystery_proc(upper_left_quad, x);
         }
         else
         {
             int last_row = M.numrows()-1;
             int last_col = M.numcols()-1
             matrix<int> upper_right_quad = submatrix(M,0,mid_col+1, mid_row,last_col);
             matrix<int> lower_left_quad  = submatrix(M,mid_row+1,0, last_row,mid_col);
             matrix<int> lower_right_quad = submatrix(M,mid_row+1,mid_col+1, last_row,last_col);
             bool result = mystery_proc(upper_right_quad, x)
                        or mystery_proc(lower_left_quad, x)
                        or mystery_proc(lower_right_quad, x);
             return result;
         }
```

Notes: the input matrix M is always square with $m = n * n = 2^k$ elements, all integers, all distinct., with k an positive integer
The procedure `submatrix(M,a,b,x,y)` returns a new matrix made from the elements of M in between rows `a` and `x`, and between columns `b` and `y`, inclusive.

4.1 Give a description of what the goal of the mystery procedure is, and how it achieves this goal.

4.2 Justify that the *worst case* runtime T of the mystery procedure, as a function of the size m of the input matrix M, is defined by the following recurrence relation: $T(m) = 3T(m/4) + O(1)$.

4.3 State the Master Theorem, use it to solve the above recurrence relation, and finally give the worst case runtime T of the mystery procedure as a function of n.

4.4 What is the *average case* runtime recurrence relation, as a function of m (justify you answer)? Solve it and finally give the average case runtime of the mystery procedure as function of n.

Answers:

4.1 The procedure aims to check if integer `x` is present in matrix `M`. It uses a divide and conquer approach, which works because the matrix is sorted. Line 18 compares `x` to the central element of the matrix. If the center is larger, then we know that if `x` is present, it will have to be in the upper left quadrant of the matrix. If the center is smaller, then `x` must be in one of the remaining three quadrants (upper right, lower left, lower right).

4.2 In the worst case, each call to the procedure will make 3 recursive calls, the case where the center element is smaller than `x`. Each call only makes 3 total comparisons, which is $O(1)$ in total. Each recursive call is 1/4 the size of the original call.
Thus the recurrence relation is $T(m) = 3T(m/4) + O(1)$.

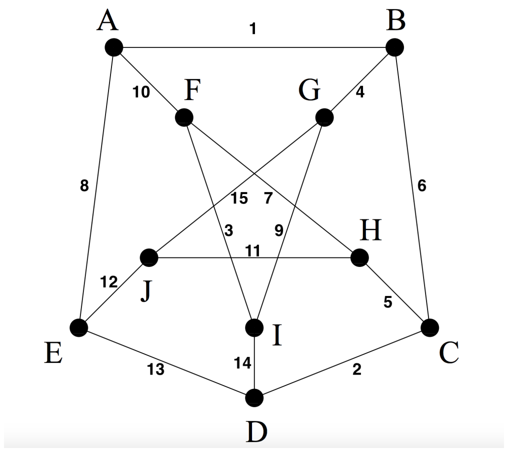
4.3 See book for definition of Master Theorem.
This is a case where $a = 3, b = 4, k = 0$ so $a > b^k$, leading to $T(m) = O(m^{log_4(3)})$.
Since $m = n^2$ we have $T(m) = O(n^{2*log_4(3)}) \approx O(n^{1.585})$.

4.4 In the average case, assuming that `x` being greater than the central element is as likely as being smaller, their will be $(3+1)/2 = 2$ recursive calls. This leads to a $O(n^{2*log_4(2)}) = O(n)$ average runtime.

Note: there is another, simpler algorithm that runs in $O(n)$ *worst case* runtime.

5 Graph Algorithm

Consider the following graph, commonly called the Petersen graph. In this case it has been randomly weighted.



5.1 Give the adjacency matrix and adjacency list representations of this Petersen graph.

5.2 Give the definition of a minimum spanning tree in a connected, undirected, weighted graph.

5.3 Apply Prim's algorithm to find a minimum spanning tree in the given Petersen graph.

5.4 Given an arbitrary connected, undirected, weighted graph $G = (V, E)$, with $|V|$ vertices and $|E|$ edges, what is the runtime of Prim's algorithm:

- if you use an adjacency matrix representation for G?
- if you use an adjacency list representation for G?

Answers:

5.1 Adjacency matrix:

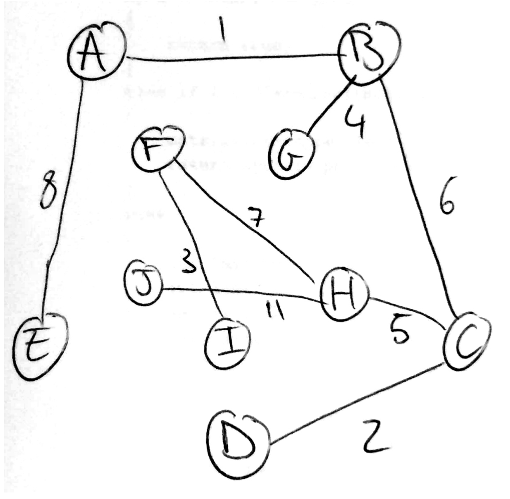
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A	X	1	X	X	8	10	X	X	X	X
B	1	X	6	X	X	X	4	X	X	X
C	X	6	X	2	X	X	X	5	X	X
D	X	X	2	X	13	X	X	X	14	X
E	8	X	X	13	X	X	X	X	X	12
F	10	X	X	X	X	X	X	7	3	X
G	X	4	X	X	X	X	X	X	9	15
H	X	X	5	X	X	7	X	X	X	11
I	X	X	X	14	X	3	9	X	X	X
J	X	X	X	X	12	X	15	11	X	X

5.1 Adjacency list:

A	{(B,1), (E,8), (F,10)}
B	{(A,1), (C,6), (G,4)}
C	{(B,6), (D,2), (H,5)}
D	{(C,2), (E,13), (I,14)}
E	{(A,8), (D,13), (J,12)}
F	{(A,10), (H,7), (I,3)}
G	{(B,4), (I,9), (J,15)}
H	{(C,5), (F,7), (J,11)}
I	{(D,14), (F,3), (G,9)}
J	{(E,12), (G,15), (H,11)}

5.2 See book.

5.3 Prim's algorithm, starting with A (chosen arbitrarily, final result will be identical because all the edge weights are distinct):
Final result:



Inserted in the following order:
(A-B), (B-G), (B-C), (C-D), (C-H), (F-H), (F-I), (A-E), (H-J).

5.4 Prim's algorithm runtime:
- using an adjacency matrix is $O(|V|^2)$.
- using an adjacency list is $O(|E| * Log(|V|)$ (using heaps to find minimum weighted edge).

6 Quicksort

Consider the following array `A = [7, 10, 9, 8, 11, 14, 12, 6, 3, 1, 4, 15, 5, 13, 2]`

6.1 Use the Quicksort algorithm to sort the array A. Use the first element of the array as your pivot. Show each step of the algorithm in detail.

6.2 When using the first element as the pivot, what type of input produces the worst case runtime? Justify that this leads to a runtime recurrence relation of $T(N) = T(N-1) + cN$, with N the size the input array, c a constant, and $T(0) = T(1) = 1$.
Solve this recurrence *exactly*, i.e. do not use big-O notation (hint: use a telescoping sum).

6.3 Ideally, what property should the pivot have to minimize the runtime, and which value of an array has this property? Justify that this leads runtime recurrence relation of
$T(N) = 2T(N/2) + cN$. Use the master theorem to solve it.

Answers:

6.1
```
pivot: 7          [6,3,1,4,5,2] 7 [10,9,8,11,14,12,15,13]
pivots: 6,10      [3,1,4,5,2] 6 7 [9,8] 10 [11,14,12,15,13]
pivots: 3,9,11    [1,2] 3 [4,5] 6 7 [8] 9 10 11 [14,12,15,13]
pivots: 1,4,14     1 [2] 3 4 [5] 6 7 8 9 10 11 [12,13] 14 [15] (8 is alone thus sorted)
pivot: 12          1 2 3 4 5 6 7 8 9 10 12 [13] 14 15
```
A final call sees that `[13]` has one element and the sorting is finished.

6.2 An already sorted array is the worst type of input for a Quicksort routine that uses the first element as the pivot. This leads to a recursive call that is only one smaller (the pivot removed) than the input. This leads to a runtime recurrence relation of $T(N) = T(N-1) + cN$. cN is the work done when selecting the pivot and by the $N-1$ comparisons to the pivot.
See page 319 of the book for a proof that this leads to
$T(N) = T(1) + c\sum_{i=2}^{N} i = c((N+1)N/2 - 1) + 1$.

6.3 Ideally the pivot should split the array into two sub-arrays of the same size. The median of the array is the value that has this property.
This leads to 2 calls both half the size of the original call, thus $T(N) = 2T(N/2) + cN$.
This is a Master Theorem case with $a = 2, b = 2, k = 1$ and $a = b^k$ so $T(N) = O(N^1 * Log(N))$

7 Extra Credit

- "A quine is a non-empty computer program which takes no input and produces a copy of its own source code as its only output." [Wikipedia: Quine (Computing)].
  Write a `c++` quine.

The most basic quine apply the following idea:

```
print the following text, the second time in quotes:
   "print the following text, the second time in quotes"
```