EC 504 Midterm – Fall 2023

Name: _____

ID: _____

Instructions: This exam is closed book and no notes – except for 2 pages of personal notes to be passed in the end. No use of laptops or internet. You have one hour and fifty minutes. Do the easy ones first.

1. (20 pts) Answer True or False to each of the questions below. Each question is worth 2 points. Answer true only if it is true as stated, with no additional assumptions. No explanation is needed, but any explanation is likely to earn partial credit, and no explanation will not earn any credit if the answer is wrong.

   (a) $N^2(1+1/N)^{N^2} \in \Theta(N^2 e^N)$ HINT: $ln(1+x) \simeq x$ for small x
   Solution: TRUE: This is ok because $(1+1/N)^{N^2} = e^{N^2 ln(1+1/N)} \simeq e^N$.

   (b) Neither $N^2 \in O(N + N^3 + (-1)^N N^3)$ nor $N^2 \in \Omega(N + N^3 + (-1)^N N^3)$.
   Solution: TRUE: $N + N^3 + (-1)^N N^3$ oscilate beteen N and N^3 so N^2 is to always bigger or smaller than it.

   (c) The best median finding algorithm is on average $\Theta(N)$ but worst case $\Theta(N^2)$
   Solution: FALSE The trick to find recursely in 5 bunchs is always $\Theta(N)$

   (d) The sum total height $T_H(N)$ and total depth $T_D(N)$ of N nodes in a binary tree is HN only if the tree is perfect.
   Solution: FALSE: A simple counter example is

   ```
     3
    /
   2
  /
 1
   ```

   With $T_H(3) = 1 + 2$ and $T_D(3) = 1 + 2$ so $T_H(3) + T_D(3) = NH = 6$ for $N = 3$ and $H = 2$

   (e) A connected undirected acyclic graph is a binary tree.
   Solution: FALSE: This is any tree, not necessarily binary.

   (f) For a binary tree of (total) height H the number $(L(H))$ of leaves obeys $L(H) = O(2^H)$.

Solution: TRUE: A perfect tree has 2^H leaves, This is an upper bound since taking away leaves always decreases the number of leaves but not necessarily the height unless you remove the entire bottom row and you back to the perfect tree with $H-1$.

(g) Given an array of N integers, the best algorithm has a worst case time to build the heap $\Theta(Nlog(N))$.
Solution: FALSE: The bottom up approch of restoring heap order is $O(N)$.

(h) If doubling the size ($N \to 2N$) causes the execute time $T(N)$ of an algorithm to increase by a factor of 2, then $T(N) \in O(4N)$.
Solution: FALSE: Counter exmple suppose $T(N) = c\_0N^2$

(i) Any sorting algorithms for 32 bit positive integers that only uses the $\leq$ comparison test must have worst case performance $\Omega(N \log(N))$.
Solution: FALSE: As $N \to \infty$ you are forces to have more and more duplicates so that buket or histgrame methods are $O(N)$.

(j) If $f_i(N) \in O(N^2)$, then $\sum_{i=1}^{N} f_i(N) \in O(N^3)$
Solution: TRUE: If you have N terms bounded by N^2 the sum is bounded by $N \times N^2$.

2. (10 pts) Consider the following recursion relation.

$$T(N) = 3T(N/2) + N^k$$

(a) Substitute $T(N) = c_0 N^\gamma$ into the homogeneous equation (i.e. dropping N^k) and determine γ and for what values of k does the exact solution satisfy $T(N) = \Theta(N^\gamma)$.

Solution: The homogeneous equation (i.e. drop the last term has solution by substitution

$$c_0\, N^\gamma = c_0\, 3\, (N/2)^\gamma \implies 1 = 3/2^\gamma \quad \text{or} \quad \gamma = \log(3)/\log(2) \tag{1}$$

Base 2 is nice since $\log_2(2) = 1$ or $\gamma = \log_2(3)$

This is gives a leading solution $T(N) = \Theta(N^\gamma)$ if $k < \gamma$

(b) Now assume that $k = \gamma$ and find the exact solution in the form: $T(N) = c_0 N^\gamma + c_1 N^\gamma \log_2(N)$. (HINT: First show that $c_1 = 1$. Then Determine c_0 in terms of $T(1)$ by setting $N = 1$.)

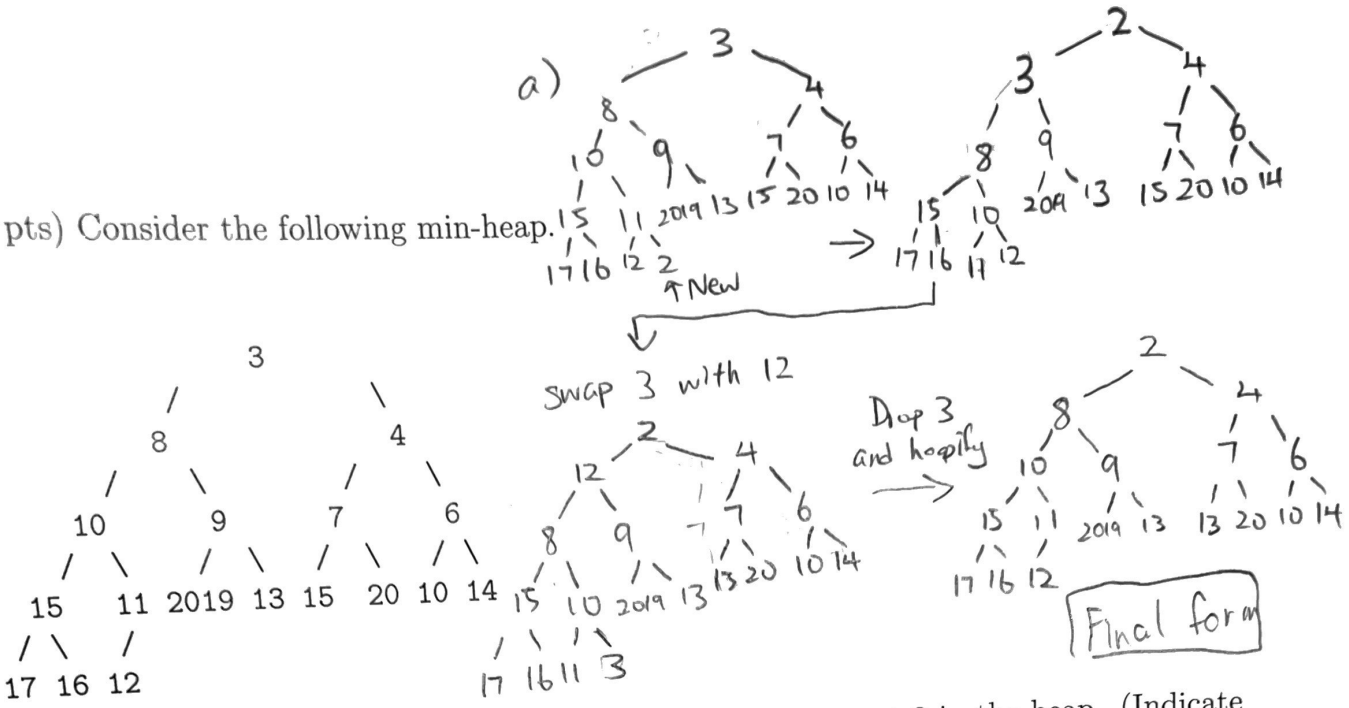
Solution: Note a homogeneous solution never determines the multiplicative const c_0. For this we must find a fulls solution. The substitution gives

$$\begin{aligned} c_0\, N^\gamma + c_1 N^\gamma \log_2(N) &= c_0\, 3\, (N/2)^\gamma + c_1 3 (N/2)^\gamma \log_2(N/2) + N^\gamma \\ &= c_0\, 3\, (N/2)^\gamma + c_1 (N)^\gamma [\log_2(N) - \log_2(2)] + N^\gamma \end{aligned} \tag{2}$$

Obviously the first (homogeneous term) still cancels. We could have ignore this piece for now.

But to cancel the additive term we need to use both $3/2^\gamma = 1$ and again $\log_2(2) = 1$, to get $c_1 = 1$. The setting $N = 1$ we finally fix c_0 by $T(1) = c_0\, 1^\gamma + c_1 1^\gamma \log_2(1) = c_0$. Actually you can do this step first if you prefer.

3. (20 pts) Consider the following min-heap.

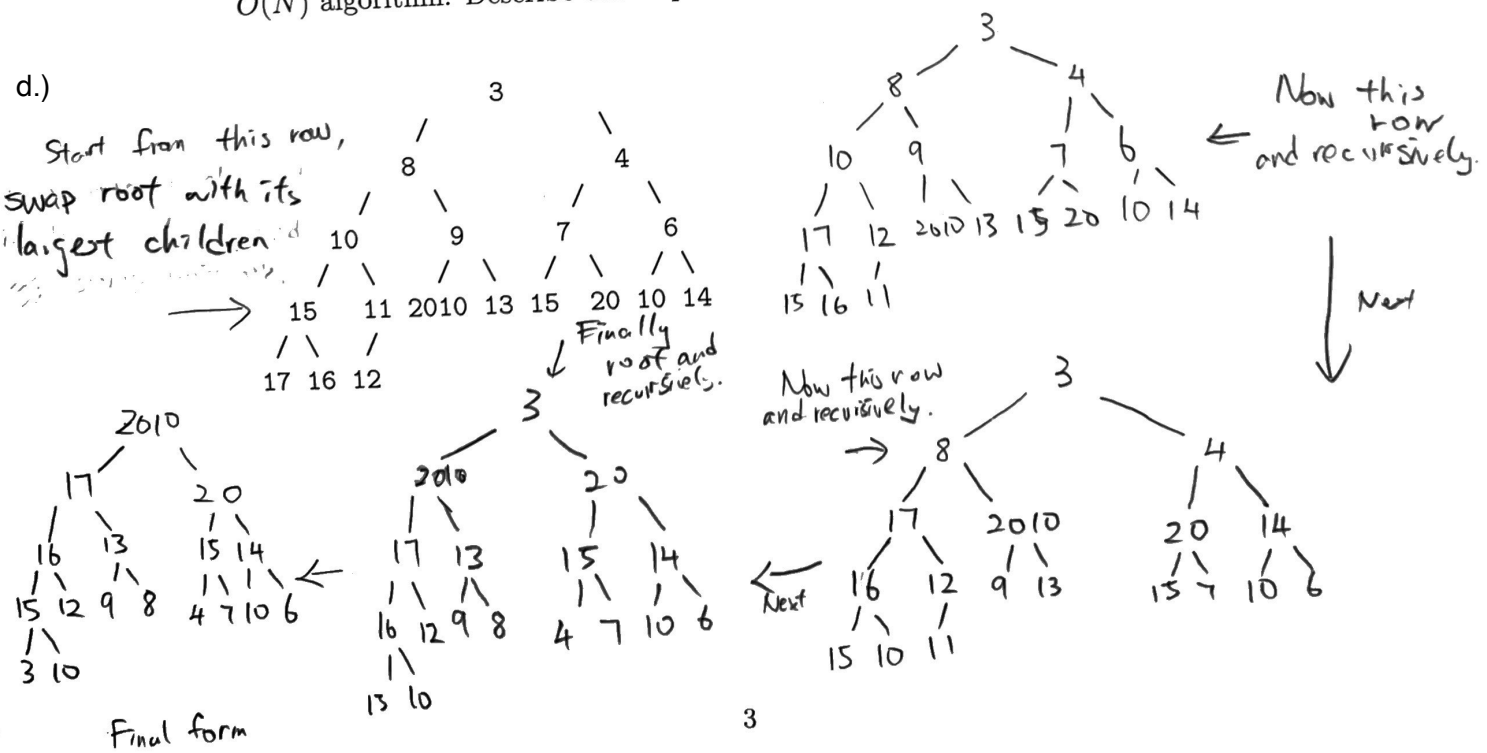


(a) Show the min-heap which results after inserting the element 2 in the heap. (Indicate the sequence of steps with arrows.) Then in this new heap show the steps required to delete element 3. Draw the final min-heap

(b) Consider min-heap as an array **int a[N+1]** with the size of heap stored in **a[0] = N**. (For example the one above has **a[0] = 18** and $a[1], a[2], \cdots, a[18]$. Describe carefully an $\Theta(N \log N)$ algorithm to sort `in place` the array elements $a[1], a[2], \cdots, a[N]$ in descending order. Use $a[0]$ the decreasing number remaining in the heap as you proceed with the sort. Next page

(c) Now use $a[0]$ as a temporary to re-arrange `in place` the sort into ascending order with $O(N)$ extra swaps. Next page.

(d) Re-arrange the original min-heap (repeated below) into a max-heap by a "bottom up" $O(N)$ algorithm. Describe the steps level by level.



b.) The min heap ican be use to sort in descending order: It is a iteration. In the min-heap with N values (e.g. N = 18 above) remove the min into temp (could be a[0]) and place at the empty site a[N]. Now you have min heap with N -1 values ending at a[N-1]. Now repeat this until the heap empty. The reverse sorted list is now in a[1] … a[N].

c.) To reverse the order of list do swap a[k] of wiht a[N +1 -k) for k = 1,…,N/2 using a[0] as a temp:
e.g the swap routine in pseudo code is

```
swap( a)
{
for(k =1; k< N/2,k++)  a[0] = a[k];  a[k] = a[N +1-k]; a[k] = a[0;
}
```

4. (20 pts) This problem is to construct step by step BST and AVL trees given the following list of $N = 10$ elements.

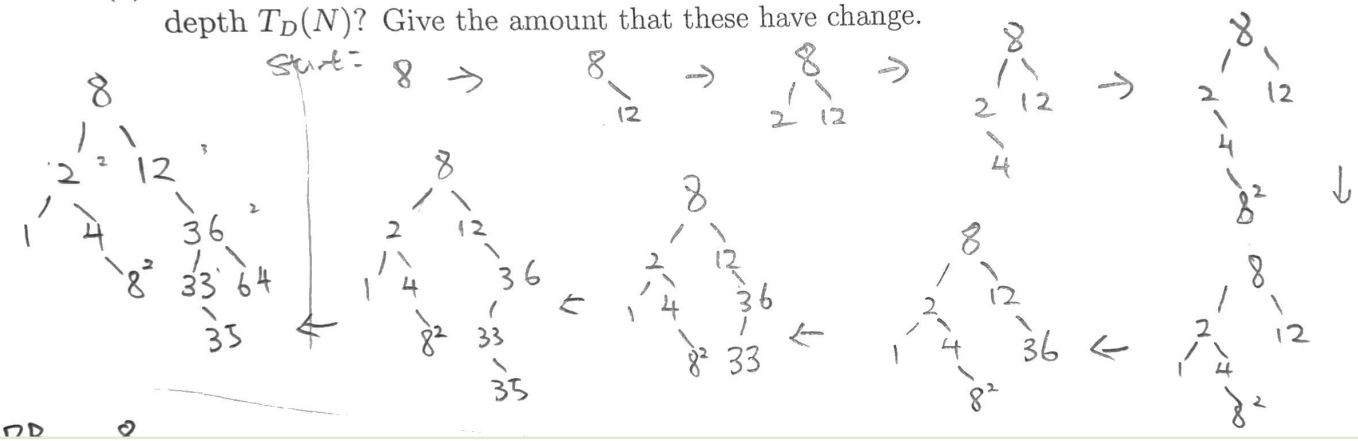
8 12 2 4 8 1 36 33 35 64

(a) Insert them sequentially into a BST (Binary Search Tree).

(b) Insert them sequentially into an empty AVL tree, restoring the AVL property after each insertion. Show the AVL tree which results after each insertion and name the type of rotation (RR or LL zig-zig or versus RL or LR zig-zag).

(c) Relative to the BST has AVL tree decreased the total height $T_H(N)$ and the total depth $T_D(N)$? Give the amount that these have change.

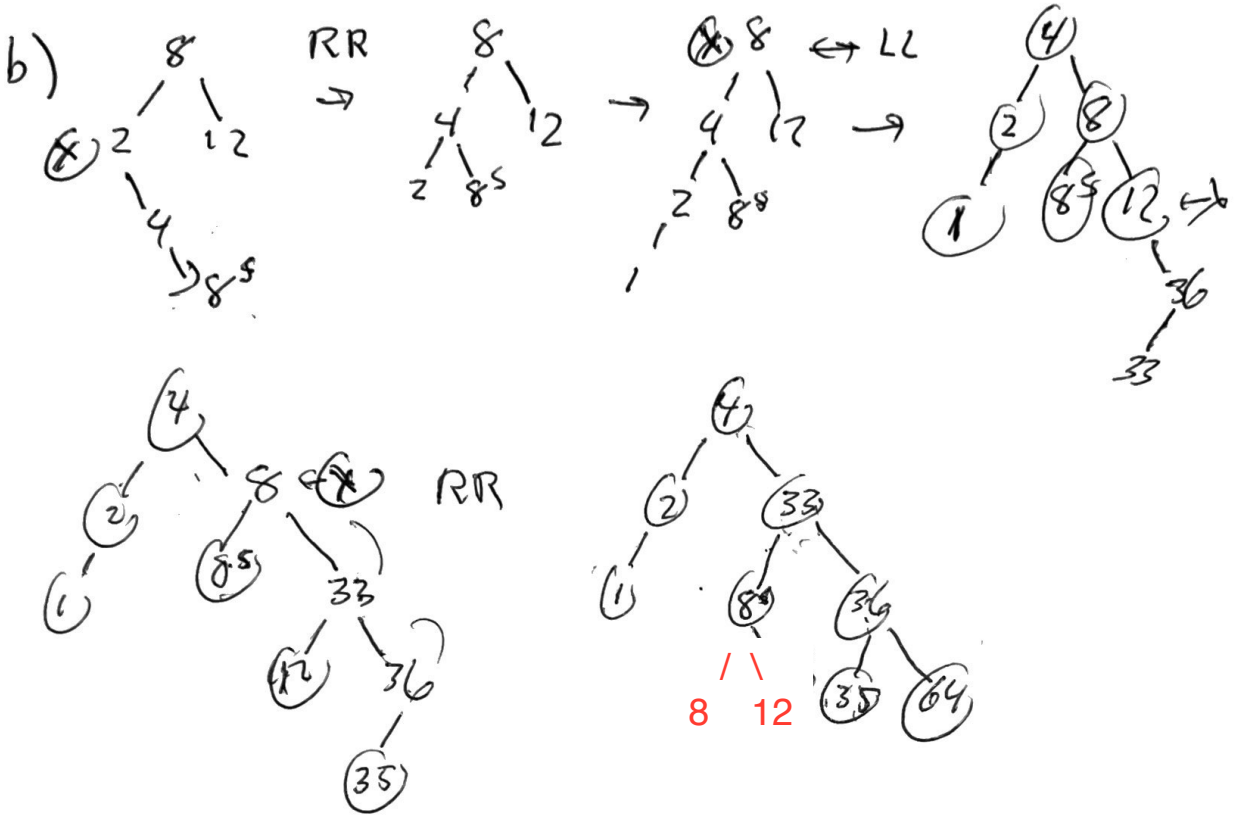
a)



b) On next page

Corrected in Red

b)



---

c) BST $T_D(N) = 1 \times 2 + 2 \cdot 3 + 3 \cdot 4 = 21$

$T_H(N) = 1 + 1 + 2 + 2 + 3 + 4 = 13$) 34

AVL $= T_D(N) = 1 \times 2 + 3 \cdot 2 + 3 * 4 = 20$ · 28

$T_H(N) = 2 \cdot 1 + 1 + 2 + 3 = 8$

5. (15 pts) You are interested in compression. Given a file with characters, you want to find the binary code which satisfies the prefix property (no conflicts) and which minimizes the number of bits required. As an example, consider an alphabet with 8 symbols, with relative weights (frequency) of appearance in an average text file give below:

alphabet:	G	R	E	A	T	F	U	L
weights:	8	6	60	30	4	12	16	15

(a) Determine the Huffman code by constructing a tree with **minimum external path length**: $\sum_{i=1}^{8} w_i d_i$. (Arrange tree with smaller weights to the left.)

(b) Identity the code for each letter and list the number of bits for each letter and compute the average number of bits per symbol in this code. Is it less than 3? (You can leave the answer as a fraction since getting the decimal value is difficult without a calculator.)

(c) Give an example of weights for these 8 symbols that would saturate 3 bits per letter. What would the Huffman tree look like? Is a Huffman code tree always a full tree?

Solution: We build a tree bottom up starting least frequent (**smallest weights** w_i) so nice to sort the letters first in that order. The resultant graph is in Fig.1

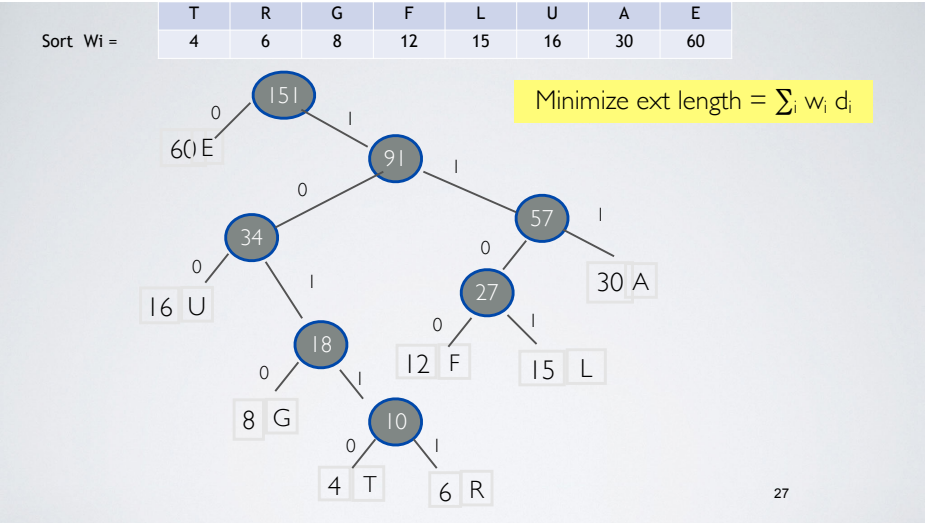


Figure 1: Rearrange Letter sorting the weight and start at the bottom, adding to get nodes (circle) for combined latter below and join at each level the minimum of letters and/or nodes. The coding convention at each level is to put the smaller number (in the node and/or leaf to the left.

Then we identify the code for each letter and build at table to compute the final total external length if Fig. 2

If all the letters have equal weight the Huffman code will be a perfect binary tree of height 3 with all the letter in the 8 leaves at the bottom. Any order give a 3 bit code for each letter. Actually even with unequal weight is all weight are less than the sum of any two it still would be still be a perfect tree.

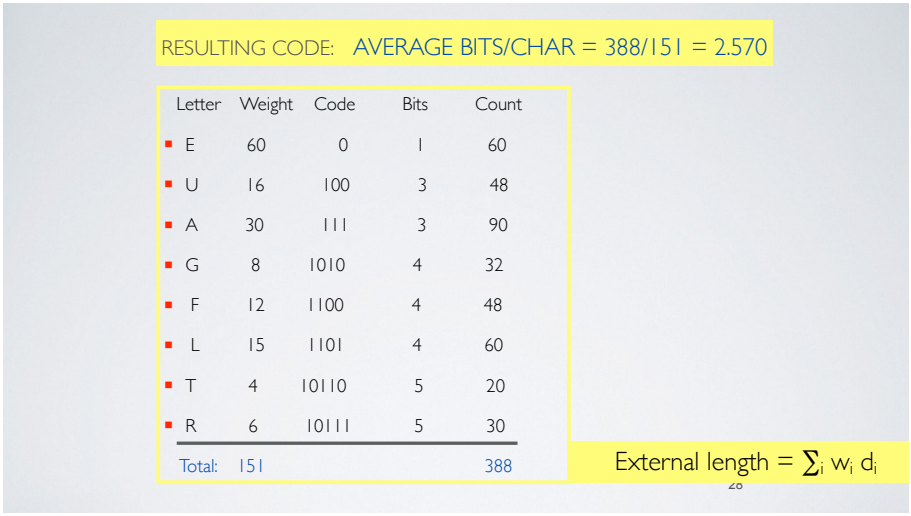


Figure 2: Table of letter with their weight w_i, the code, number of bitsd_i and the contribution $w_i d_i$ to the total external length. The ratio of the total external length to the total weight it the mean number of bits per letter.

A Huffman code is always a full tree. First note that all the letter reside in leaves or it doesn't have unique code. Suppose it wasn't a full tree. Any node with no children would have be a leaf (can't add no weight together to get a node!). Any node with one child would be a letter that would then be reabsorbed into the node as leaf.

6. (15 pts) For each of these recursions, please give the tightest upper bound for the recursion. You can write your answer as $T(n) \in O(g(n))$ for your best choice of function g(n).

(a) $T(n) = 8T(n/2) + n^2$

Solution: $\log_2(8) = 3$, so by the Master theorem, $T(n) \in \Theta(n^3) \in O(n^3)$.

(b) $T(n) = 5T(n/2) + (n \log n)^2$

Solution: $(n \log n)^2 \in O(n^{\log_2(5)})$ for some $ > 0$, so $T(n) \in \Theta(n \log_2(5))$.

(c) $T(n) = 2T(n/2) + n \log n$

Solution: This is similar to what we saw in one of the homework exercises. It does not satisfy the Master theorem, but by counting, we get $T(n) \in \Theta(n(logn)^2)$.

(d) $T(n) = T(n/2) + n$

Solution: This is a simple case of the Master theorem. the we get $T(n) \in \Theta(n)$.

(e) $T(n) = 3T(n-1) - T(n-1)$

Solution: Try solution $T(n) = x^n$, this leads to the quadratic equation $x^2 - 3x + x = 0$ with roots $x = (1 \pm \sqrt{9-4})/2 = 1/2 \pm \sqrt{5/4}$ so $T(n) \in \Theta((1/2 + \sqrt{5/4})^n)$.

7. (10 pts) Consider the definition $N! = 1 * 2 * 3 * 4 * \cdots * (N-1) * N$ Prove the statements below for $T(N) = \log(N!)$ [1]

   (a) Prove $\log(N!) \in O(N \log N)$.

   Solution:

   Take N times the last term:

$$\begin{aligned} T(N) &= \log(1) + \log(2) + \cdots + \log(N) < Nlog(N) \\ &\Longrightarrow T(N) = \log(N!) = O(N \log N) \end{aligned}$$

   (b) Prove $\log(N!) \in \Omega(N \log N)$.

   Solution: Consider the terms that at $N/2$ and larger

$$\begin{aligned} T(N) &> \log(N/2) + \log((N+1)/2) + \cdots + \log(N) \\ &> (N/2)\log(N/2) = (N/2)(\log(N) - \log 2) \\ &\Longrightarrow T(N) = \log(N!) = O(N \log N) \end{aligned}$$

   (c) Do the above two statements (a) and (b) imply $\log(N!) \in \Theta(N \log N)$? Do they imply $N! \in \Theta(N^N)$? (Explain)

   Solution: Yes $O(..)$ and $\Omega(..)$ is the same $\Theta(..)$ But for we have the actual form is $N! \sim N^N e^{-N}$ so $N! \in \Omega(N^N)$ fails but taking the log give $log(N!) \sim Nlog(N) - N$ which is ok.

[1] Hint: Just look a the sum.

$$T[N] = \log(1) + \log(2) + \log(3) + \cdots + \log(N)$$

and look at part of this series to give under and over estimates that still are proportional to $NlogN$.
Do not use Stirling's approximation: $N! \simeq \sqrt{2\pi N} e^{-N} N^N$ The point of this problem is to get the leading term for $\log(N!)$ without it!