Homework 2: Due February 07 (11:55 a.m.)

Instructions

- **Answer each question on a separate page.**
- **Honors questions are optional. They will not count towards your grade in the course. However you are encouraged to submit your solutions on these problems to receive feedback on your attempts.**
- **You must enter the names of your collaborators or other sources as a response to Question 0. Do NOT leave this blank; if you worked on the homework entirely on your own, please write "None" here. Even though collaborations in groups of up to 3 people are encouraged, you are required to write your own solution.**

Question 0: List all your collaborators and sources: ($-\infty$ points if left blank)

Question 1: (1+4+5=10 points)

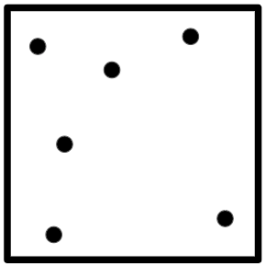

Figure 1: Sparse set of points in a unit square

1. State the Pigeon Hole Principle in your own words and give some intuition as to why it should be true.
2. Here is a more formal statement of the Pigeon Hole Principle: Let A and B be finite sets such that $|A| > |B|$. Then there does not exist any one to one function $f : A \to B$. Prove this using induction.
3. Consider a unit square (side length is 1 unit). A set of points S is called *sparse* if any two points in S are at least $\frac{1}{4}$ distance away from each other. Prove that there exists a constant C such that no sparse set of points within the square can contain more than C many points. (One approach to proving this is to try to apply the Pigeon Hole Principle)

Question 2: (6+6+6=18 points)

Solve each of the following recurrences by the Recursion Tree method. We have provided a complete explanation for this table at the end of this homework. **Please read the text before proceeding with this problem.** In order to do this, you will complete the following table and the blanks, for each recurrence relation:

Level	Size of Problem	Non-Recursive Cost of One Problem	No. of Problems	Total Cost
0	_____ (1)	_____ (2)	_____ (3)	_____ (4)
1	_____ (5)	_____ (6)	_____ (7)	_____ (8)
⋮	⋮	⋮	⋮	⋮
i	_____ (9)	_____ (10)	_____ (11)	_____ (12)
⋮	⋮	⋮	⋮	⋮
$d-1$	_____ (13)	_____ (14)	_____ (15)	_____ (16)
d	_____ (17)	_____ (18)	_____ (19)	_____ (20)

where d denotes the depth of the recursion tree and has value: $d =$ _____ (21).

Adding the last column, we get:

$$T(n) = \underset{(22)}{\_\_\_\_\_} + \sum_{i=0}^{d-1} \underset{(23)}{\_\_\_\_\_}$$

This solves to $T(n) =$ _____ (24)

1. $T(n) = 2 \cdot T(n/2) + n,\ T(1) = 1$
2. $T(n) = 2 \cdot T(n-1) + 1,\ T(0) = 1$
3. $T(n) = T(n/3) + n,\ T(1) = 1$

Question 3: (2+2+2+4=10 points)

In this problem, we will solve the following recurrence using the **substitution method** (i.e., induction). Assume $T(1) = 0,\ T(2) = 1$ and that the recurrences below define $T(n)$ for $n > 2$:

$$T(n) = T\Big(\frac{n}{3}\Big) + T\Big(\frac{2n}{3}\Big) + 1\ . \tag{1}$$

You may assume that n is of the form $n = 3^k$, for some natural number k. This means that the $n/3$ and $2n/3$ terms in the above recurrence can always be assumed to be natural numbers (rather than fractions).

1. Try $T(n) = \sqrt{n}$. Does the recurrence hold? Which side is bigger? Show your calculations.
2. Try $T(n) = n$. Does the recurrence hold? Which side is bigger? Show your calculations.
3. Try $T(n) = n^2$. Does the recurrence hold? Which side is bigger? Show your calculations.

Among parts 1 to 3, which choice of $T(n)$ "almost" satisfied recurrence (1)? That is, for which choice of $T(n)$ was the left-hand side *almost equal* to the right-hand side of (1)? Reflect on this question because it would help solve the next part.

4. Prove your result formally using the substitution method. (Hint: try $T(n)$ with a constant offset. That is, $T(n) = n^p + c$ for some $c \in \mathbb{R}$ and $p > 0$. Note that parts 1-3 correspond to $c = 0$ and $p = 1/2, p = 1, p = 2$, respectively. Now, solve for p and c.)

Question 4: (5+5=10 points)

Recall Merge Sort, in which a list is sorted by first sorting the left and right halves, and then merging the two lists. We define the *3-Merge Sort* algorithm, in which the input list is split into 3 equal length parts (or as equal as possible), each is sorted recursively, and then the three lists are merged to create a final sorted list.

1. Write a recurrence for $T(n)$, the worst-case run time for 3-Merge Sort on any input containing n elements.
2. Solve this recurrence for $T(n)$. Prove your result formally using the substitution method (i.e., induction). For full credit, you'll need to show both an upper and a lower bound.

Question 5: (1+1+6+2=10 points)

In class, we learned that $\log_2 n = O(n^p)$ for *any* $p > 0$. In the following exercises, we will prove this fact for the special case $p \geq 1$. That is, we will show that $\log_2 n = O(n^p)$ for any constant $p \geq 1$ using induction.

1. Show that to prove $\log_2 n = O(n^p)$ for $p \geq 1$, it suffices to show that $\log_2 n \leq n$ for all $n \geq 1$. (Hint: you may use the fact that if $p \geq 1$, then $n \leq n^p$ for any $n \geq 1$).
2. Now we proceed to showing $\log_2 n \leq n$ for all $n \in \mathbb{N}$ by induction. Show the base case for $n = 1$.
3. Prove the inductive step. That is, show that if $\log_2 n \leq n$ holds for $1 \leq n \leq k$, then $\log_2(k+1) \leq k+1$. (Hint: for $k \geq 1$, compare $\log_2(k+1)$ and $\log_2(2k)$).

Together, parts 1-3 complete the proof that $\log_2 n = O(n^p)$ for any constant $p \geq 1$.

4. Now prove that for any base $b > 1$ and any $p \geq 1$, $\log_b n = O(n^p)$. (Hint: prove that $\log_b n = O(\log_2 n)$).

Honors Question 1:

Solve the following recurrences (assume $T(1) = T(2) = 1$ and that the recurrences below define $T(n)$ for $n > 2$):

1. $T(n) = 2T(\lceil \sqrt{n} \rceil) + \log_2 n$
2. $T(n) = \lceil \sqrt{n} \rceil \, T(\lceil \sqrt{n} \rceil) + n$

In all cases you must prove the correctness of your solution. It is good enough to get an answer $f(n)$ such that $T(n) = \Theta(f(n))$.

Honors Question 2:

Can you improve the bound C in Question 1, part 3? What's the smallest upper bound you can prove?

Honors Question 3:

Think about applying Recursion Tree to solve the recurrence relation seen in Question 3. How does that work?

Recursion Tree, an Explanation

Read the following text before solving Problem 2. In class, we looked at solving recurrence relation using a recursion tree. Rather than drawing the tree, one can simplify the process of solving by this method with the use of a simple table which can be filled up almost mechanically. The table is reproduced below for your convenience:

Level	Size of Problem	Non-Recursive Cost of One Problem	No. of Problems	Total Cost
0	_____ (1)	_____ (2)	_____ (3)	_____ (4)
1	_____ (5)	_____ (6)	_____ (7)	_____ (8)
⋮	⋮	⋮	⋮	⋮
i	_____ (9)	_____ (10)	_____ (11)	_____ (12)
⋮	⋮	⋮	⋮	⋮
$d-1$	_____ (13)	_____ (14)	_____ (15)	_____ (16)
d	_____ (17)	_____ (18)	_____ (19)	_____ (20)

where d denotes the depth of the recursion tree and has value: $d =$ _____ (21).

Adding the last column, we get:

$$T(n) = \underset{(22)}{\_\_\_\_\_} + \sum_{i=0}^{d-1} \underset{(23)}{\_\_\_\_\_}$$

This solves to $T(n) =$ _____ (24)

Now, we explain the table.

- The level column starts from level 0 which is the root of the tree, and ends at level d which is where the leaf nodes can be found.
- **Size of Problem**: This begins with n in Blank (1) typically. Every level after is influenced by the term inside the recurrence call.
  - For recurrence relations of the form: $T(n) = aT(n/b) + f(n)$, this goes as n in blank 1, then blank 5 will have $\frac{n}{b}$, blank 9 will have $\frac{n}{b^i}$
  - For recurrence relations of the form: $T(n) = aT(n-b) + f(n)$, this goes as n in blank 1, then blank 5 will have $n-b$, blank 9 will have $n-ib$
- **Depth** d: To compute the depth d, you will have to use the base case and the size of the problem at level d, to compute d.
  - For recurrence relations of the form: $T(n) = aT(n/b) + f(n)$, we know that at level d, the size of the problem is $\frac{n}{b^d}$. If we have base case as $T(1)$, then we find d such that $\frac{n}{b^d} = 1$ or $d = \log_b n$. If the base case was some $T(x)$ then we have to find d such that $\frac{n}{b^d} = x$ or $d = \frac{1}{x}\log_b n$.

  - For recurrence relations of the form: $T(n) = aT(n-b) + f(n)$, we know that at level d, the size of the problem is $n - bd$. If we have base case as $T(0)$, then we find d such that $n - bd = 0$ or $d = n/b$. If the base case was some $T(x)$ then we have to find d such that $n - bd = x$ or $d = (n - x)/b$.

- **Non-Recursive Cost of One Problem**: This is obtained by merely substituting the size of problem in function $f(n)$.
  - For recurrence relations of the form: $T(n) = aT(n/b) + f(n)$, this goes as $f(n)$ in blank 2, then blank 6 will have $f(\frac{n}{b})$, blank 13 will have $f(\frac{n}{b^i})$
  - For recurrence relations of the form: $T(n) = aT(n-b) + f(n)$, this goes as $f(n)$ in blank 2, then blank 6 will have $f(n-b)$, blank 13 will have $f(n-ib)$

- **No. of Problems**: This corresponds to the number of nodes in each level of the tree, starting from 1 typically.
  - For recurrence relations of the form: $T(n) = aT(n/b) + f(n)$, this goes as 1 in blank 3, then blank 7 will have a, blank 11 will have a^i
  - For recurrence relations of the form: $T(n) = aT(n-b) + f(n)$, this goes as 1 in blank 3, then blank 7 will have a, blank 11 will have a^i

- **Total Cost**: Simply multiply third and fourth column for every row. So, blank 4 will be product of blanks 2 and 3, and so on.

- Blanks 22 and 20 are the same, and Blank 23 is obtained by summing up the last column.

- To get 24, you may use standard simplifications for summations:
  - $\sum_{i=1}^{n} 1/i = O(\log n)$
  - $\sum_{i=1}^{n} 1 = O(n)$
  - $\sum_{i=1}^{n} i = O(n^2)$
  - $\sum_{i=1}^{n} i^2 = O(n^3)$