Midterm Exam (Solution Sketches)

(2:30 PM – 3:45 PM : 75 Minutes)

- This exam will account for either 15% or 30% of your overall grade depending on your relative performance in the midterm and the final. The higher of the two scores (midterm and final) will be worth 30% of your grade, and the lower one 15%.
- There are three (3) questions, worth 75 points in total. Please answer all of them in the spaces provided.
- There are 16 pages including four (4) blank pages and two (2) pages of appendices. Please use the blank pages if you need additional space for your answers.
- The exam is *open slides* and *open notes*. But *no books* and *no computers*.

GOOD LUCK!

Question	Pages	Score	Maximum
1. A Broken ATM	2–4		25
2. Hops	6–9		25
3. Recurrences with Triangular Numbers	11–12		25
Total			75

NAME: ______________________________

Question 1. [**25 Points**] **A Broken ATM.** This question is about an ATM (Automated Teller Machine) that can store dollar bills of exactly n different integral values, but when a customer tries to withdraw cash the machine fails unless it can output the amount using exactly k bills, where both n and k are positive integers. We assume that the value of the largest bill the machine stores is not more than cn for some constant $c \geq 1$. We also assume that before each transaction the machine will have at least k bills of each of the n different dollar values it stores (i.e., it will be refilled as soon as the number of bills of any value drops below k).

Now the question is: with any given n and k as above, how many distinct cash amount the ATM can successfully deliver?

1(a) [**5 Points**] Show that for any given k you can output all distinct withdrawal amounts the ATM can successfully deliver in $\mathcal{O}\left(n^2k^2\right)$ time. For example, if the ATM stores only \$5, \$10, \$20 and \$50 bills and $k = 2$, then it can fulfill the following 10 distinct withdrawal amounts:

1. \$10 (= \$5 + \$5)	**2.** \$15 (= \$5 + \$10)	**3.** \$20 (= \$10 + \$10)	**4.** \$25 (= \$5 + \$20)	**5.** \$30 (= \$10 + \$20)
6. \$40 (= \$20 + \$20)	**7.** \$55 (= \$5 + \$50)	**8.** \$60 (= \$10 + \$50)	**9.** \$70 (= \$20 + \$50)	**10.** \$100 (= \$50 + \$50)

[*Hint: Observe that since the value of the largest dollar bill is at most cn, the largest dollar amount one can withdraw with k bills is ckn. So, maybe you can compute a Boolean (*True/False*) array A of length ckn, where $A[i]$ ($1 \leq i \leq ckn$) will be* True *provided dollar amount i can be made using exactly k dollar bills, otherwise $A[i]$ will be* False.]

Solution Sketch:

```
ATM( c, k, n, DOLLAR BILLS )

for index ← 1 to ckn do A[index] ← FALSE
A[0] ← TRUE
for iteration ← 1 to k do                                   {k iterations}
    for index ← 1 to ckn do B[index] ← FALSE
    B[0] ← TRUE
    for each amount in DOLLAR BILLS do                      {n iterations}
        for index ← 1 to ckn do                             {ckn iterations}
            if A[index] = TRUE then B[index + amount] ← TRUE
    A ← B
S ← ∅
for index ← 1 to ckn do
    if A[index] = TRUE then S ← S ∪ {index}
return S
```

$$Complexity: \; \mathcal{O}\left(k \times n \times ckn\right) = \mathcal{O}\left(n^2k^2\right)$$

Grading Criteria:

- 4 marks - for Algorithm. (pseudo code or words.)
- 1 mark - for proof of Complexity.
- 2 marks - If shown algorithm only for $k = 2$

1(*b*) [**10 Points**] Explain how you will output all distinct withdrawal amounts in $\mathcal{O}\left(n^{1+\epsilon}\right)$ time when $k = 2$, where ϵ is any given positive constant which can be arbitrarily close to zero.

[*Hint: Construct a polynomial for the dollar bills of different values the ATM stores, i.e., coefficient of* x^r *will be 1 if it stores* \$$r$ *bills and 0 otherwise.*]

Solution Sketch:

Constructing the Polynomial. $\longrightarrow 2marks$

$$P(x) = \sum a_r x^r, a_r = \begin{cases} 1 & \text{if ATM stores \$}r\text{ bills} \\ 0 & \text{otherwise,} \end{cases}$$

Multiplying the Polynomial with itself. $\longrightarrow 2marks$
For $k = 2$,

$$P_2(x) = P(x) * P(x)$$

Interpreting the output. $\longrightarrow 4marks$

The non zero coefficients of the polynomial are the distinct withdrawal amounts possible.

$$P_2(x) = \sum c_r x^r, c_r = \begin{cases} \geq 1 & \text{if ATM can deliver \$}r\text{ amount} \\ 0 & \text{otherwise,} \end{cases}$$

Complexity. $\longrightarrow 2marks$

Using the FFT, polynomial multiplication can be done in $O(n \log n)$ time and $\log n = \mathcal{O}\left(n^{\epsilon}\right)$, hence the complexity is $\mathcal{O}\left(n^{1+\epsilon}\right)$

Grading Criteria:

- 2 marks - Constructing polynomial.
- 2 mark - Identifying that can be solved using polynomial multiplication.
- 4 marks - Interpreting the output.
- 2 marks - showing $n \log n$ is $n^{1+\epsilon}$.

1(c) [**10 Points**] Explain how you will extend your algorithm from part 1(b) to output all distinct withdrawal amounts in $\mathcal{O}\left(nk(n^{\epsilon} + k^{\epsilon})\right)$ time for any given k, where ϵ is a given constant as in part 1(b).

[*Hint: Use repeated squaring. For example,* x^{25} *can be computed using only 6 multiplications (instead of 24) as follows:* $x^{25} = \left(x^{12}\right)^2 \cdot x$, $x^{12} = \left(x^{6}\right)^2$, $x^{6} = \left(x^{3}\right)^2$ *and* $x^3 = (x)^2 \cdot x$.]

Solution Sketch:

The polynomial is given in co-efficient form: $P(x) = \sum a_r x^r$

We will have to compute $(P(x))^k$. Since $P(x)$ has degree bound $cn + 1$, $(P(x))^k$ will have degree bound $ckn + 1$.

Convert $P(x)$ to point-value form which will take $\mathcal{O}\left(nk \log\left(nk\right)\right)$ time because we need to evaluate $P(x)$ at $\mathcal{O}\left(kn\right)$ (i.e., $ckn + 1$) distinct points.

We will compute $(P(x))^k$ using repeated squaring entirely in point-value form (please check solution to Question 2(c) to see what a repeated squaring algorithm looks like). To achieve this we will have to compute pairwise polynomial products in point-value form $\Theta\left(\log k\right)$ times. Each such product will require the computation of $\mathcal{O}\left(kn\right)$ value $\times$ value products, and thus $\mathcal{O}\left(kn\right)$ time. Thus the total time needed for the repeated squaring phase is this $\mathcal{O}\left(nk \log k\right)$.

Converting from point value form to co-efficient form requires $\mathcal{O}\left(nk \log(nk)\right)$ time.

Complexity.

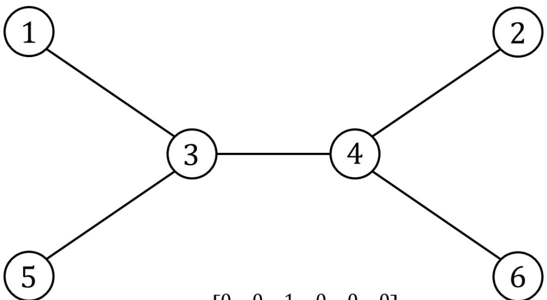
$\mathcal{O}\left(nk \log\left(nk\right)\right) + \mathcal{O}\left(nk \log k\right) + \mathcal{O}\left(nk \log\left(nk\right)\right) = \mathcal{O}\left(nk \log\left(nk\right)\right) = \mathcal{O}\left(nk(\log n + \log k)\right) = \mathcal{O}\left(nk(n^{\epsilon} + k^{\epsilon})\right)$

Grading Criteria:

- 6 marks - for Algorithm. (3 marks - for repeated squaring in coefficient form)
- 4 mark - for proof of Complexity. (2 marks - complexity proof for coefficient form).

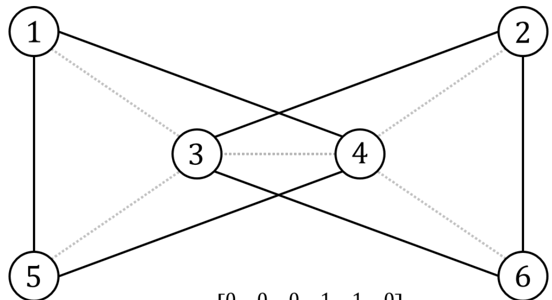
Use this page if you need additional space for your answers.

QUESTION 2. **[25 Points] Hops.** Suppose G is an undirected graph that has n vertices. Each vertex of G is identified by a unique integer in $[1, n]$. We say that two vertices u and v of G are adjacent provided they are connected by an edge. All edges of G are recorded in an $n \times n$ adjacency matrix A, where $A[u][v]$ is set to 1 provided vertices u and v are connected by an edge (i.e., provided edge (u, v) exists in G), otherwise $A[u][v]$ is set to 0. Since G is undirected $A[u][v] = A[v][u]$ always holds. We say that vertices u and v are connected by an h-hop path provided v can be reached from u following a path containing exactly h edges and vice versa. An $n \times n$ matrix $D^{(h)}$ which we call an h-hop matrix, records each pair of vertices that are connected by h-hop paths. Entry $D^{(h)}[u][v]$ is set to 1 provided u and v are connected by an h-hop path, and 0 otherwise. Again $D^{(h)}[u][v] = D^{(h)}[v][u]$ for all $u, v \in [1, n]$. Clearly, $D^{(1)} = A$.



$$D^{(1)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 1: An undirected graph whose edges (i.e., 1-hop paths) are captured by the matrix $D^{(1)}$ which is also the adjacency matrix of this graph.



$$D^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2: The solid edges show the vertices connected by 2-hop paths in the graph on the left. Matrix $D^{(2)}$ marks every pair of vertices connected by 2-hop paths in that graph.

Figure 1 shows an example undirected graph containing 6 vertices and its $D^{(1)}$ matrix which is the same as its adjacency matrix. Figure 2 shows the $D^{(2)}$ matrix for the graph in Figure 1.

```
Iter-Reach ( Z, X, Y )        { X, Y, Z are n × n matrices,
                                where n is a positive integer }
for i ← 1 to n do
    for j ← 1 to n do
        Z[ i ][ j ] ← 0
        for k ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] ⊕ X[ i ][ k ] ⊗ Y[ k ][ j ]
```

Figure 3: Combining an h_1-hop matrix $X = D^{(h_1)}$ and an h_2-hop matrix $Y = D^{(h_2)}$ to obtain an $(h_1 + h_2)$-hop matrix $Z = D^{(h_1+h_2)}$.

```
Iter-MM ( Z, X, Y )           { X, Y, Z are n × n matrices,
                                where n is a positive integer }
for i ← 1 to n do
    for j ← 1 to n do
        Z[ i ][ j ] ← 0
        for k ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] + X[ i ][ k ] · Y[ k ][ j ]
```

Figure 4: Multiplying two $n \times n$ matrices X and Y and putting the result in another $n \times n$ matrix Z.

Figure 3 shows an iterative algorithm ITER-REACH that uses bitwise OR ($\oplus$) and bitwise AND

($\otimes$) operators to obtain a new $(h_1 + h_2)$-hop matrix $Z = D^{(h_1+h_2)}$ by combining an h_1-hop matrix $X = D^{(h_1)}$ and an h_2-hop matrix $Y = D^{(h_2)}$.

Observe that ITER-REACH can be obtained from the standard iterative matrix multiplication algorithm ITER-MM shown in Figure 4 simply by replacing the standard addition ($+$) and multiplication ($\times$) operators with the bitwise OR ($\oplus$) and bitwise AND ($\otimes$) operators, respectively. Both algorithms run in $\Theta\left(n^3\right)$ time.

Now answer the following questions.

2(a) [**8 Points**] Argue that you cannot obtain a $\Theta\left(n^{\log_2 7}\right)$ time algorithm for computing $D^{(h_1+h_2)}$ from $D^{(h_1)}$ and $D^{(h_2)}$ by simply replacing the $+$ and $\times$ operators with $\oplus$ and $\otimes$ operators, respectively, in Strassen's matrix multiplication algorithm given in the Appendix.

Solution Sketch:

(See Strassen's algorithm in the Appendix) Strassen's algorithm uses subtractions, which cannot be simply performed by $\oplus$ and $\otimes$ operators. Subtraction is the inverse of addition ($+$), but the bitwise OR operator ($oplus$) does not have an inverse.

Grading Criteria:

If you point out subtractions can't be done by $\oplus$ and $\otimes$, or $-$ can't be replaced by $\oplus$ and $\otimes$, you get 8 points. Mention only subtractions without further explanation will lose 1 point.

If you tried to explain something, you get 2 points.

2(b) [**10 Points**] Give an $\Theta\left(n^{\log_2 7}\right)$ time algorithm for correctly computing $D^{(h_1+h_2)}$ from $D^{(h_1)}$ and $D^{(h_2)}$ based on Strassen's matrix multiplication algorithm.

Solution Sketch:

First observe that while the algorithm in Figure 3 sets each $Z[i][j]$ to either 1 or 0 depending on if there is a path between i and j or not, respectively, the matrix multiplication algorithm in Figure 4 sets $Z[i][j]$ to the number of distinct paths between i and j.

So, we will use Strassen's algorithm with $+$ and $\times$ to calculate $D^{(h_1+h_2)}$. Suppose the result is E. Since there might be entries in E greater than 1, which is not exactly what we want in $D^{(h_1+h_2)}$, we will replace every entry in E to obtain $D^{(h_1+h_2)}$ by the following criteria: set $D_{ij}^{(h_1+h_2)} = 0$ if $E_{ij} = 0$, otherwise set $D_{ij}^{(h_1+h_2)} = 1$. This leads to the correct result of $D^{(h_1+h_2)}$.

The complexity of Strassen's algorithm (using $+$ and $\times$) is $\Theta(n^{\log_2 7})$. Replacing each entry in E is $\Theta(n^2)$. So the overall complexity is $\Theta(n^{\log_2 7}) + \Theta(n^2)$, where $n^{\log_2 7}$ dominates.

Hence, we can compute $D^{(h_1+h_2)}$ in $\Theta(n^{\log_2 7})$ time.

Grading Criteria:

Using Strassen's algorithm (using $+$ and $\times$) gains 3 points.

Getting the correct output (E), which means that you are aware of there are entries greater than 1, gains 2 points.

Being able to transform E back to correct $D^{(h_1+h_2)}$ gains 3 points.

Correct analysis of Strassen's complexity $\Theta(n^{\log_2 7})$ gains 1 point. You have to analyze it again to get this point.

Correct analysis of overall complexity, which means that you are aware of $\Theta(n^2)$ cost brought by replacing, gains 1 point.

2(c) [**7 Points**] For any positive integer n, explain how you will compute $D^{(n)}$ in $\Theta\left(n^{\log_2 7} \log n\right)$ time.

[*Hint: Use your result from part* 2(b).]

Solution Sketch:

The following algorithm computes $D^{(n)}$ from $D^{(1)}$ using Strassen's algorithm and repeated squaring.

Pow$\left(\ D^{(1)},\ n\ \right)$

1. ***if*** $n = 1$ ***then*** $Q \leftarrow D^{(1)}$
2. ***else***
3. $\quad Q \leftarrow$ Pow($D^{(1)}$, $\lfloor \frac{n}{2} \rfloor$)
4. $\quad Q \leftarrow Q \times Q$ $\qquad\qquad\qquad$ {*use Strassen's algorithm*}
5. $\quad$ ***if*** n is an odd number ***then***
6. $\qquad Q \leftarrow Q \times D^{(1)}$ $\qquad\qquad$ {*use Strassen's algorithm again*}
7. ***return*** Q

The algorithm uses Strassen's algorithm at most $2 \log n$ times, and thus has a overall cost of $\Theta\left(n^{\log_2 7} \log n\right)$.

Grading Criteria:

Presenting the algorithm correctly gains 5 points. If you only analysis the case that n is power of 2, you lose 2 points.

2 points for correct analysis of complexity.

Use this page if you need additional space for your answers.

Question 3. [**25 Points**] **Recurrences with Triangular Numbers.** The k-th *triangular number* $\triangle k$ is defined as follows: $\triangle k = 1 + 2 + \ldots + k$, where k is a natural number. The first few triangular numbers ($\triangle 1$, $\triangle 2$, $\triangle 3$, $\triangle 4$, $\triangle 5$ and $\triangle 6$) are shown in Figure 5 below.

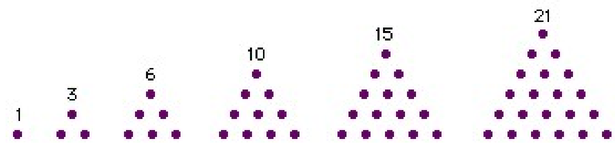


Figure 5: The first 6 triangular numbers.

3(a) [**10 Points**] The time $T(n)$ needed to query a widely used data structure of size n can be described by the following recurrence relation involving triangular numbers:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 6, \\ \sum_{k=2}^{5} \frac{1}{\triangle k} T\left(\frac{kn}{k+1}\right) + \frac{1}{3}T(n) + \Theta(1) & \text{otherwise.} \end{cases}$$

Solve the recurrence for finding an asymptotic tight bound for $T(n)$.

[*Hint: Frame it as an Akra-Bazzi recurrence.*]

Solution Sketch:

The values of $\triangle k$ are :
$\triangle 2 = 3$, $\triangle 3 = 6$, $\triangle 4 = 10$, $\triangle 5 = 15$,
Substituting them in $T(n)$ and expanding the $\sum$ we get the following equation:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 6, \\ \frac{1}{3}T\left(\frac{2n}{3}\right) + \frac{1}{6}T\left(\frac{3n}{4}\right) + \frac{1}{10}T\left(\frac{4n}{5}\right) + \frac{1}{15}T\left(\frac{5n}{6}\right) + \frac{1}{3}T(n) + \Theta(1) & \text{otherwise.} \end{cases},$$

Now bringing $\frac{1}{3}T(n)$ on the right side of the equation to left.

$$T(n) - \frac{1}{3}T(n) = \frac{1}{3}T\left(\frac{2n}{3}\right) + \frac{1}{6}T\left(\frac{3n}{4}\right) + \frac{1}{10}T\left(\frac{4n}{5}\right) + \frac{1}{15}T\left(\frac{5n}{6}\right) + \Theta(1)$$

Solving this we get

$$\frac{2}{3}T(n) = \frac{1}{3}T\left(\frac{2n}{3}\right) + \frac{1}{6}T\left(\frac{3n}{4}\right) + \frac{1}{10}T\left(\frac{4n}{5}\right) + \frac{1}{15}T\left(\frac{5n}{6}\right) + \Theta(1)$$

Multiplying $\frac{3}{2}$ on both sides to the above equation gives,

$$T(n) = \frac{1}{2}T\left(\frac{2n}{3}\right) + \frac{1}{4}T\left(\frac{3n}{4}\right) + \frac{3}{20}T\left(\frac{4n}{5}\right) + \frac{1}{10}T\left(\frac{5n}{6}\right) + \Theta(1)$$

Comparing with the Akra-Bazzi form, we have:

$$a_1 = \frac{1}{2}, b_1 = \frac{2}{3}$$
$$a_2 = \frac{1}{4}, b_2 = \frac{3}{4}$$
$$a_3 = \frac{3}{20}, b_3 = \frac{4}{5}$$
$$a_3 = \frac{1}{10}, b_4 = \frac{5}{6}$$

Above recurrence satisfies Akra-Bazzi conditions,

$$x_0 \text{ is a constant and } \geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\} for 1 \leq i \leq k$$

$g(u) = \Theta(1)$, which is of the form $g(u) = u^{\alpha} \log^{\beta} u$ with $\alpha = \beta = 0$ and thus satisfies the polynomial growth condition.

Substituting the above a_i, b_i in $\sum a_i b_i^p = 1$, we get

$$p = 0$$
$$T(x) = x^p(1 + \int_1^x \frac{g(u)}{u^{p+1}} du) = x^0(1 + \int_1^x \frac{1}{u^{0+1}} du) = 1 + \int_1^x \frac{1}{u} du = 1 + \log(x) \Rightarrow T(n) = \Theta(\log(n))$$

Grading Criteria:

- 1 mark - Stating x_0 is a constant and $\geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\} for 1 \leq i \leq k$ condition
- 1 mark - Stating $g(x)$ is a non negative function that satisfies a polynomial growth condition
- 1 mark - Stating $\sum a_i b_i^p = 1$
- 1 mark - Solving for p ($p = 0$).
- 2 marks - Taking $T(n)$ on right side to left side.
- 2 marks - Solving the integration.
- 2 marks - all steps written correctly and arriving at answer.

3(b) [**15 Points**] The expected running time $T(n)$ of a randomized algorithm on an input of size n can be described by the following recurrence relation involving triangular numbers $\triangle 2 = 3$, $\triangle 3 = 6$ and $\triangle 4 = 10$:

$$T(n) = \begin{cases} \Theta(n) & \text{if } n \leq 1024, \\ \frac{1}{3}n^{\frac{2}{3}}T\left(n^{\frac{1}{3}}\right) + \frac{1}{6}n^{\frac{5}{6}}T\left(n^{\frac{1}{6}}\right) + \frac{1}{10}n^{\frac{9}{10}}T\left(n^{\frac{1}{10}}\right) + \frac{2}{5}T(n) + \Theta(n \log\log n) & \text{otherwise.} \end{cases}$$

Solve the recurrence for finding an asymptotic tight bound for $T(n)$.

[*Hint: You may find it useful to divide both sides of the recurrence by n. Then reduce it to an Akra-Bazzi recurrence.*]

Solution Sketch:

Divide both sides of the recurrence by n,

$$\frac{T(n)}{n} = \begin{cases} \Theta(1) & \text{if } n \leq 1024, \\ \frac{1}{3}n^{\frac{2}{3}}\frac{T(n^{\frac{1}{3}})}{n} + \frac{1}{6}n^{\frac{5}{6}}\frac{T(n^{\frac{1}{6}})}{n} + \frac{1}{10}n^{\frac{9}{10}}\frac{T(n^{\frac{1}{10}})}{n} + \frac{2}{5}\frac{T(n)}{n} + \frac{\Theta(n\log\log n)}{n} & \text{otherwise.} \end{cases}$$

For $n > 1024$,

$$\frac{T(n)}{n} = \frac{1}{3}\frac{T(n^{\frac{1}{3}})}{n^{\frac{1}{3}}} + \frac{1}{6}\frac{T(n^{\frac{1}{6}})}{n^{\frac{1}{6}}} + \frac{1}{10}\frac{T(n^{\frac{1}{10}})}{n^{\frac{1}{10}}} + \frac{2}{5}\frac{T(n)}{n} + \Theta(\log\log n)$$

Now bringing $\frac{2}{5}T(n)$ on the right side of the equation to left.

$$\frac{3}{5}\frac{T(n)}{n} = \frac{1}{3}\frac{T(n^{\frac{1}{3}})}{n^{\frac{1}{3}}} + \frac{1}{6}\frac{T(n^{\frac{1}{6}})}{n^{\frac{1}{6}}} + \frac{1}{10}\frac{T(n^{\frac{1}{10}})}{n^{\frac{1}{10}}} + \Theta(\log\log n)$$

$$\Rightarrow \frac{T(n)}{n} = \frac{5}{9}\frac{T(n^{\frac{1}{3}})}{n^{\frac{1}{3}}} + \frac{5}{18}\frac{T(n^{\frac{1}{6}})}{n^{\frac{1}{6}}} + \frac{1}{6}\frac{T(n^{\frac{1}{10}})}{n^{\frac{1}{10}}} + \Theta(\log\log n)$$

Substitute $R(n)$ for $\frac{T(n)}{n}$,

$$R(n) = \frac{1}{3}R(n^{\frac{1}{3}}) + \frac{1}{6}R(n^{\frac{1}{6}}) + \frac{1}{10}R(n^{\frac{1}{10}}) + \Theta(\log\log n)$$

Substitute $n = 2^m \Rightarrow \log n = m$, where m is a real number,

$$R(2^m) = \begin{cases} \Theta(1) & \text{if } m \leq 10, \\ \frac{5}{9}R(2^{\frac{m}{3}}) + \frac{5}{18}R(2^{\frac{m}{6}}) + \frac{1}{6}R(2^{\frac{m}{10}}) + \Theta(\log m) & \text{otherwise.} \end{cases}$$

Substitute $S(m) = R(2^m)$,

$$S(m) = \frac{5}{9}S(\frac{m}{3}) + \frac{5}{18}S(\frac{m}{6}) + \frac{1}{5}S(\frac{m}{10}) + \Theta(\log m)$$

Comparing with the Akra-Bazzi form, we have:

$$a_1 = \frac{5}{9}, b_1 = \frac{1}{3}$$
$$a_2 = \frac{5}{18}, b_2 = \frac{1}{6}$$
$$a_3 = \frac{1}{6}, b_3 = \frac{1}{10}$$

Above recurrence satisfies Akra-Bazzi conditions,

$$x_0 \text{ is a constant and} \geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\} for 1 \leq i \leq k$$

$g(u) = \Theta(\log u)$, which is of the form $g(u) = u^\alpha \log^\beta u$ with $\alpha = 0$ and $\beta = 1$ and thus satisfies the polynomial growth condition.
Substituting the above a_i, b_i in $\sum a_i b_i^p = 1$, we get

$$p = 0$$

Then

$$S(m) = m^p(1 + \int_1^m \frac{g(u)}{u^{p+1}} du)$$
$$S(m) = m^0(1 + \int_1^m \frac{\log u}{u^{0+1}} du)$$
$$S(m) = 1 + \int_1^m \frac{\log m}{u} du$$
$$S(m) = 1 + \frac{(\log m)^2}{2}$$
$$S(m) = (\log m)^2$$

Substitute $S(m) = R(2^m)$,

$$R(m) = (\log \log m)^2$$

Substitute $R(m) = T(n)/n$,

$$T(n) = n(\log \log n)^2$$

Grading Criteria:

- 1 mark - Stating x_0 is a constant and $\geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\} for 1 \leq i \leq k$ condition
- 1 mark - Stating g(x) is a non negative function that satisfies a polynomial growth condition
- 1 mark - Stating $\sum a_i b_i^p = 1 = 1$

- 1 mark - Solving for p (p = 0).
- 1 marks - Taking T(n) on right side to left side.
- 1 mark - Substituting T(n)/n.

- 1 mark - n = 2^m.
- 2 marks - $\Theta\left(n \log \log n\right)$ to $\Theta\left(\log n\right)$ and $n < 1024$ to $n < 10$.
- 2 marks - transform back - 1 + 1 mark each.
- 2 marks - Solving the integration.
- 2 marks - all steps written correctly and arriving at answer.

Appendix: Recurrences

Master Theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & \text{otherwise,} \end{cases}$$

where, $\frac{n}{b}$ is interpreted to mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$. Then $T(n)$ has the following bounds:

Case 1: If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

Case 2: If $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$ for some constant $k \geq 0$, then $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$.

Case 3: If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Akra-Bazzi Recurrences. Consider the following recurrence:

$$T(x) = \begin{cases} \Theta(1), & \text{if } 1 \leq x \leq x_0, \\ \sum_{i=1}^{k} a_i T(b_i x) + g(x), & \text{otherwise,} \end{cases}$$

where,

1. $k \geq 1$ is an integer constant,
2. $a_i > 0$ is a constant for $1 \leq i \leq k$,
3. $b_i \in (0, 1)$ is a constant for $1 \leq i \leq k$,
4. $x \geq 1$ is a real number,
5. x_0 is a constant and $\geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\}$ for $1 \leq i \leq k$, and
6. $g(x)$ is a nonnegative function that satisfies a polynomial growth condition (e.g., $g(x) = x^\alpha \log^\beta x$ satisfies the polynomial growth condition for any constants $\alpha, \beta \in \Re$).

Let p be the unique real number for which $\sum_{i=1}^{k} a_i b_i^p = 1$. Then

$$T(x) = \Theta\left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right).$$

Appendix: Computing Products

Integer Multiplication. Karatsuba's algorithm can multiply two n-bit integers in $\Theta\left(n^{\log_2 3}\right) = \mathcal{O}\left(n^{1.6}\right)$ time (improving over the standard $\Theta\left(n^2\right)$ time algorithm).

Matrix Multiplication. Strassen's algorithm can multiply two $n \times n$ matrices in $\Theta\left(n^{\log_2 7}\right) = \mathcal{O}\left(n^{2.81}\right)$ time (improving over the standard $\Theta\left(n^3\right)$ time algorithm).

Polynomial Multiplication. One can multiply two n-degree polynomials in $\Theta\left(n \log n\right)$ time using the FFT (Fast Fourier Transform) algorithm (improving over the standard $\Theta\left(n^2\right)$ time algorithm).

Appendix: Strassen's Matrix Multiplication Algorithm

$$\begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \times \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{bmatrix}$$

$$= \begin{bmatrix} -P_{r1} \quad -P_{22} \quad +P_{d1} \quad +P_{c2} & +P_{r1} \quad +P_{11} \\ +P_{r2} \quad -P_{22} & -P_{r2} \quad +P_{11} \quad +P_{d1} \quad -P_{c1} \end{bmatrix}$$

Sums:

$$\begin{aligned} X_{r1} &= X_{11} + X_{12} & Y_{r1} &= Y_{11} + Y_{12} \\ X_{r2} &= X_{21} + X_{22} & Y_{r2} &= Y_{21} + Y_{22} \\ X_{c1} &= X_{11} - X_{21} & Y_{c1} &= Y_{11} - Y_{21} \\ X_{c2} &= X_{12} - X_{22} & Y_{c2} &= Y_{12} - Y_{22} \\ X_{d1} &= X_{11} + X_{22} & Y_{d1} &= Y_{11} + Y_{22} \end{aligned}$$

Products:

$$\begin{aligned} P_{11} &= X_{11} \cdot Y_{c2} & P_{c1} &= X_{c1} \cdot Y_{r1} \\ P_{22} &= X_{22} \cdot Y_{c1} & P_{c2} &= X_{c2} \cdot Y_{r2} \\ P_{r1} &= X_{r1} \cdot Y_{22} & P_{d1} &= X_{d1} \cdot Y_{d1} \\ P_{r2} &= X_{r2} \cdot Y_{11} \end{aligned}$$

Sums:

$$\begin{aligned} Z_{11} &= -P_{r1} - P_{22} + P_{d1} + P_{c2} \\ Z_{12} &= +P_{r1} + P_{11} \\ Z_{21} &= +P_{r2} - P_{22} \\ Z_{22} &= -P_{r2} + P_{11} + P_{d1} - P_{c1} \end{aligned}$$

Running Time:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2), & \text{otherwise.} \end{cases}$$

$$= \Theta\left(n^{\log_2 7}\right)$$

$$= \mathrm{O}\left(n^{2.81}\right)$$