In-Class Midterm (Solution Ideas)

(2:35 PM – 3:50 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 four (4) 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*.

GOOD LUCK!

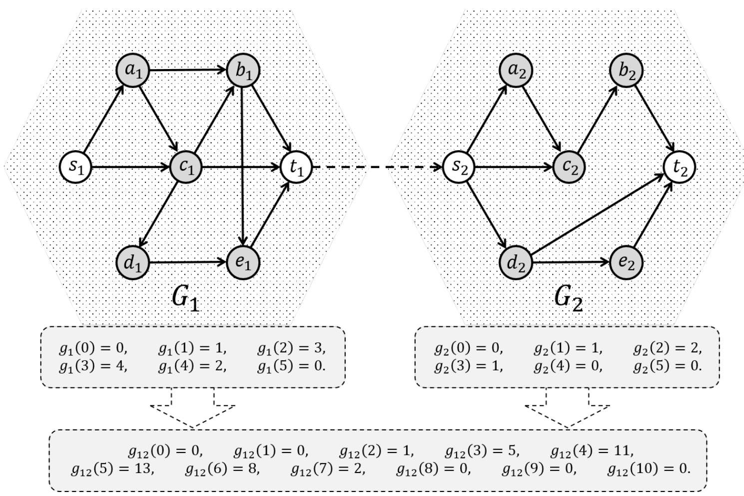
Question	Pages	Score	Maximum
1. Counting Paths	2–4		20
2. A Schönhage-Strassen-like Recurrence	6–8		25
3. Closest Pair of Points	10–11		20
4. An Impossible Priority Queue	13		10
Total			75

NAME: ____

Question 1. [**20 Points**] **Counting Paths.** Suppose you are given two directed graphs[1] G_1 and G_2 containing $n+2$ nodes each for some $n \geq 0$. For $i \in \{1, 2\}$, G_i includes two special nodes — a *source node* s_i with no incoming edges[2] and a *target node* t_i with no outgoing edges[3]. These two nodes are called *external nodes* while the rest are called *internal nodes*. The figure below shows an example with $n = 5$ in which the internal nodes are colored grey and the external nodes are white. Let $g_i(k)$ denote the number of paths in G_i that go from s_i to t_i and pass through exactly k internal (i.e., grey) nodes. For example, in the figure below $g_1(3) = 4$ which represents the following 4 paths:

$$
\begin{aligned}
&s_1 \to a_1 \to b_1 \to e_1 \to t_1, \\
&s_1 \to a_1 \to c_1 \to b_1 \to t_1, \\
&s_1 \to c_1 \to b_1 \to e_1 \to t_1 \\
\text{and } &s_1 \to c_1 \to d_1 \to e_1 \to t_1.
\end{aligned}
$$

Suppose for $0 \leq k \leq n$, all $g_1(k)$ and $g_2(k)$ values are known to you.



Now suppose you connect G_1 and G_2 by putting an edge directed from t_1 to s_2. For $0 \leq k \leq 2n$, let $g_{12}(k)$ denote the number of paths from s_1 to t_2 that pass through exactly k internal (i.e., grey) nodes. The figure above shows an example in which $g_{12}(3) = 5$ representing the following 5 paths:

$$
\begin{aligned}
&(s_1 \to c_1 \to t_1) \to (s_2 \to c_2 \to b_2 \to t_2), \\
&(s_1 \to c_1 \to t_1) \to (s_2 \to d_2 \to e_2 \to t_2), \\
&(s_1 \to a_1 \to b_1 \to t_1) \to (s_2 \to d_2 \to t_2), \\
&(s_1 \to a_1 \to c_1 \to t_1) \to (s_2 \to d_2 \to t_2) \\
\text{and } &(s_1 \to c_1 \to b_1 \to t_1) \to (s_2 \to d_2 \to t_2).
\end{aligned}
$$

[1] e.g., road networks with one-way roads
[2] e.g., incoming roads
[3] e.g., outgoing roads

1(a) [**5 Points**] For any given integer $k \in [0, 2n]$, show that $g_{12}(k)$ can be computed from g_1's and g_2's in $\mathcal{O}(n)$ time.

Solution. Observe that for any $k \in [0, 2n]$,

$$g_{12}(k) = \sum_{i \,=\, \max\{\, 0,\ k-n \,\}}^{\min\{\, k,\ n \,\}} g_1(i) \times g_2(k-i)$$

Then clearly, for any given $k \in [0, 2n]$, the following code fragment computes $g_{12}(k)$.

1. $c \leftarrow 0$
2. ***for*** $i \leftarrow \max\{\ 0,\ k-n\ \}$ ***to*** $\min\{\ k,\ n\ \}$ ***do***
3. $\quad c \leftarrow c + g_1(i) \times g_2(k-i)$
4. $g_{12}(k) \leftarrow c$

Observe that the ***for*** loop above iterates $t = \min\{\ k,\ n\ \} - \max\{\ 0,\ k-n\ \} + 1$ times with each iteration taking $\Theta(1)$ time, and so the code fragment runs in $\mathcal{O}(t)$ time. But $t = n - (k-n) + 1 = n + 1 = \mathcal{O}(n)$ when $k > n$, and since $k \leq 2n$, $t = k - 0 + 1 = k + 1 = \mathcal{O}(n)$ otherwise.

Thus the computation of $g_{12}(k)$ requires $\mathcal{O}(n)$ time.

1(b) [**15 Points**] Show that for $0 \leq k \leq 2n$, one can compute all $g_{12}(k)$ values simultaneously in $\mathcal{O}(n \log n)$ time.

Solution. In $\mathcal{O}(n)$ time we construct the following two polynomials of degree at most n each.

$$\mathcal{G}_1(z) = g_1(0) + g_1(1)z + g_1(2)z^2 + \ldots + g_1(n)z^n = \sum_{0 \leq i \leq n} g_1(i)z^i$$

$$\mathcal{G}_2(z) = g_2(0) + g_2(1)z + g_2(2)z^2 + \ldots + g_2(n)z^n = \sum_{0 \leq j \leq n} g_2(j)z^j$$

Let $\mathcal{G}_{12}(z) = \mathcal{G}_1(z)\mathcal{G}_2(z)$. Then clearly $\mathcal{G}_{12}(z)$ is of degree at most $2n$. Let

$$\mathcal{G}_{12}(z) = c_0 + c_1 z + c_2 z^2 + \ldots + c_{2n} z^{2n} = \sum_{0 \leq k \leq 2n} c_k z^k.$$

Since $\mathcal{G}_{12}(z)$ is the product of $\mathcal{G}_1(z)$ and $\mathcal{G}_2(z)$, clearly, for $k \in [0, 2n]$,

$$c_k = \sum_{i \,=\, \max\{\, 0,\; k-n \,\}}^{\min\{\, k,\; n \,\}} g_1(i) \times g_2(k-i) = g_{12}(k).$$

So if we compute the product $\mathcal{G}_1(z)\mathcal{G}_2(z)$, for each $k \in [0, 2n]$, the coefficient c_k of z^k in the product will give us the value of $g_{12}(k)$.

We know that two polynomials of degree at most n can be multiplied in $\mathcal{O}(n \log n)$ time using FFT. Hence, all coefficients of $\mathcal{G}_{12}(z)$, and thus $g_{12}(k)(= c_k)$ for all $k \in [0, 2n]$ can be computed in $\mathcal{O}(n \log n)$ time.

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

Question 2. [**25 Points**] **A Schönhage-Strassen-like Recurrence.** Consider the following recurrence (for $n \geq 2$) which is similar to the recurrence that arises during the analysis of the *Schönhage-Strassen algorithm* for multiplying large integers.

$$T(n) = \begin{cases} \Theta(1) & \text{if } 2 \leq n \leq 8, \\ n^{\frac{2}{3}} T\left(n^{\frac{1}{3}}\right) + n^{\frac{1}{3}} T\left(n^{\frac{2}{3}}\right) + \Theta(n \log n) & \text{otherwise.} \end{cases}$$

$2(a)$ [**4 Points**] Show that the recurrence above can be rewritten as follows, where $T(n) = nS(n)$.

$$S(n) = \begin{cases} \Theta(1) & \text{if } 2 \leq n \leq 8, \\ S\left(n^{\frac{1}{3}}\right) + S\left(n^{\frac{2}{3}}\right) + \Theta(\log n) & \text{otherwise.} \end{cases}$$

Solution. Dividing both sides of the given recurrence for $T(n)$ by n,

$$\frac{T(n)}{n} = \begin{cases} \frac{\Theta(1)}{n} & \text{if } 2 \leq n \leq 8, \\ \frac{T\left(n^{\frac{1}{3}}\right)}{n^{\frac{1}{3}}} + \frac{T\left(n^{\frac{2}{3}}\right)}{n^{\frac{2}{3}}} + \Theta(\log n) & \text{otherwise.} \end{cases}$$

But $\frac{T(n)}{n} = S(n)$, $\frac{T\left(n^{\frac{1}{3}}\right)}{n^{\frac{1}{3}}} = S\left(n^{\frac{1}{3}}\right)$, $\frac{T\left(n^{\frac{2}{3}}\right)}{n^{\frac{2}{3}}} = S\left(n^{\frac{2}{3}}\right)$, and for $n \in [2, 8]$, $\frac{\Theta(1)}{n} = \Theta(1)$. Hence, the recurrence above can be rewritten as follows.

$$S(n) = \begin{cases} \Theta(1) & \text{if } 2 \leq n \leq 8, \\ S\left(n^{\frac{1}{3}}\right) + S\left(n^{\frac{2}{3}}\right) + \Theta(\log n) & \text{otherwise.} \end{cases}$$

$2(b)$ [**4 Points**] Show that the recurrence in $2(a)$ can be rewritten as follows, where $P(x) = S(2^x)$.

$$P(x) = \begin{cases} \Theta(1) & \text{if } 1 \leq x \leq 3, \\ P\left(\frac{x}{3}\right) + P\left(\frac{2x}{3}\right) + \Theta(x) & \text{otherwise.} \end{cases}$$

Solution. Let $n = 2^x$, where $x = \log_2 n$. Then from the recurrence for $S(n)$ we get:

$$S(2^x) = \begin{cases} \Theta(1) & \text{if } 2 \leq 2^x \leq 8, \\ S\left(2^{\frac{x}{3}}\right) + S\left(2^{\frac{2x}{3}}\right) + \Theta(x) & \text{otherwise.} \end{cases}$$

But $S(2^x) = P(x)$, $S\left(2^{\frac{x}{3}}\right) = P\left(\frac{x}{3}\right)$, $S\left(2^{\frac{2x}{3}}\right) = P\left(\frac{2x}{3}\right)$, and $2 \leq 2^x \leq 8 \Rightarrow 1 \leq x \leq 3$. Hence, the recurrence above can be rewritten as follows.

$$P(x) = \begin{cases} \Theta(1) & \text{if } 1 \leq x \leq 3, \\ P\left(\frac{x}{3}\right) + P\left(\frac{2x}{3}\right) + \Theta(x) & \text{otherwise.} \end{cases}$$

$2(c)$ [**9 Points**] Solve the recurrence from part $2(b)$ to show that $P(x) = \Theta(x \log x)$.

Solution. The given recurrence is in the Akra-Bazzi form since for this recurrence:

$$k = 2 \geq 1 \text{ is an integer constant,}$$
$$a_1 = 1 \geq 0, b_1 = \frac{1}{3} \in (0,1),$$
$$a_2 = 1 \geq 0, b_2 = \frac{2}{3} \in (0,1),$$
$$x \geq 1 \text{ is a real number,}$$
$$x_0 = 3 \geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\}, \text{ for } i \in \{1,2\},$$
$$\text{and } g(x) = \Theta(x) = \Theta\left(x^1 \log^0 x\right), \text{ which satisfies the polynomial growth condition.}$$

Now in order to find the Akra-Bazzi solution for this recurrence, we need to find the unique real number p for which $a_1 {b_1}^p + a_2 {b_2}^p = 1 \Rightarrow \left(\frac{1}{3}\right)^p + \left(\frac{2}{3}\right)^p = 1$. This gives us $p = 1$.

Hence, the Akra-Bazzi solution:

$$\begin{aligned}
P(x) &= \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right) & \\
&= \Theta\left(x\left(1 + \int_1^x \frac{u}{u^2} du\right)\right) & \{\because p = 1\} \\
&= \Theta\left(x\left(1 + \int_1^x \frac{1}{u} du\right)\right) & \\
&= \Theta\left(x\left(1 + [\ln x + c]_1^x\right)\right) & \left\{\because \int \frac{du}{u} = \ln u + c, \textit{ where } c \textit{ is a constant}\right\} \\
&= \Theta\left(x\left(1 + \ln x\right)\right) & \\
&= \Theta\left(x \ln x\right) & \{\because \ln x = \omega(1)\} \\
&= \Theta\left(x \log x\right) & \{\because \ln x = \Theta(\log x) \textit{ for any constant base of } \log x\}
\end{aligned}$$

$2(d)$ [**8 Points**] Use your results from part $2(c)$ to show that $T(n) = \Theta(n \log n \log\log n)$.

Solution. From part $2(b)$ we know: $S(n) = P(\log n)$.

But from part $2(c)$, we have: $P(x) = \Theta(x \log x)$.

Hence, $S(n) = P(\log n) = \Theta(\log n \log\log n)$.

Again from part $2(a)$ we know: $T(n) = nS(n)$.

Hence, using the solution for $S(n)$ we have: $T(n) = \Theta(n \log n \log\log n)$.

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

Question 3. [**20 Points**] **Closest Pair of Points.** Consider the algorithm Closest-Pair given below that finds the closest pair of points among a given set of points in the plane.

Closest-Pair(P, n)

Input: A set $P = \{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$ of n points in the plane. Assume for simplicity that (a) $n = 2^k$ for some integer $k > 0$, (b) all x_i's are distinct, and (c) all y_i's are distinct.

Output: Two distinct points $p_i, p_j \in P$ such that the distance between p_i and p_j is the smallest among all pairs of points in P.

Algorithm:

1. ***if*** $n = 2$ ***then return*** $\langle p_1, p_2 \rangle$
2. ***else***
3. Find a value x such that exactly $\frac{n}{2}$ points in P have $x_i < x$, and the other $\frac{n}{2}$ points have $x_i > x$
4. Let L be the subset of P containing all points with $x_i < x$
5. Let R be the subset of P containing all points with $x_i > x$
6. $\langle p_L, q_L \rangle \leftarrow$ Closest-Pair(L, $\frac{n}{2}$)
7. $\langle p_R, q_R \rangle \leftarrow$ Closest-Pair(R, $\frac{n}{2}$)
8. $d_L \leftarrow$ distance between p_L and q_L
9. $d_R \leftarrow$ distance between p_R and q_R
10. $d \leftarrow \min\{\ d_L,\ d_R\ \}$
11. Scan P and remove each $p_i = (x_i, y_i) \in P$ with $x_i < x - d$ or $x_i > x + d$
12. Sort the remaining points of P in increasing order of y-coordinates
13. Scan the sorted list, and for each point compute its distance to the 7 subsequent points in the list. Let $\langle p_M, q_M \rangle$ be the closest pair of points found in this way.
14. Let $\langle p, q \rangle$ be the closest pair among $\langle p_L, q_L \rangle$, $\langle p_R, q_R \rangle$ and $\langle p_M, q_M \rangle$
15. ***return*** $\langle p, q \rangle$

3(a) [**10 Points**] Argue that for a set of n points, steps 3–5 take $\mathcal{O}(n)$ time while steps 8–15 take $\mathcal{O}(n \log n)$ time.

Solution. In step 3, we use the deterministic Select algorithm we saw in the class to find elements x' and x'' such that $rank(x', \{x_1, x_2, \ldots, x_n\}) = \frac{n}{2}$ and $rank(x'', \{x_1, x_2, \ldots, x_n\}) = \frac{n}{2} + 1$. This will take $\mathcal{O}(n)$ time. Then we set $x = \frac{x' + x''}{2}$. Clearly, exactly $\frac{n}{2}$ points in P will have $x_i < x$, and the other $\frac{n}{2}$ points will have $x_i > x$. Steps 4–5 require scanning the points of P once, and hence take $\mathcal{O}(n)$ time. Thus steps 3–5 take $\mathcal{O}(n)$ worst-case time.

Steps 8–10 take only $\mathcal{O}(1)$ time. Step 11 requires scanning P once and hence takes $\mathcal{O}(n)$ time. Step 12 sorts the points of P which can be done in $\mathcal{O}(n \log n)$ worst-case time using mergesort or heapsort (not standard quicksort as it takes $\Theta\left(n^2\right)$ time in the worst case). Step 13 scans the sorted list in $\mathcal{O}(n)$ time. Finally, steps 14–15 require only $\mathcal{O}(1)$ time. Overall, steps 8–15 run in $\mathcal{O}(n \log n)$ worst-case time.

Observe that since input size is n and one cannot sort n real numbers in less than $\Theta(n \log n)$ time, the two bounds above are tight, i.e., $\Theta(n)$ and $\Theta(n \log n)$, respectively.

3(b) [**10 Points**] Let $T(n)$ be the running time of Closest-Pair on a set of n points. Write a recurrence relation for $T(n)$ and solve it.

Solution. Observe that Closest-Pair is a divide-and-conquer algorithm. We know from part 3(a) that for a set of n points the cost of divide (steps 3–5) is $\Theta(n)$, and the cost of combine (steps 8–15) is $\Theta(n \log n)$. Thus cost of divide and combine, $f(n) = \Theta(n) + \Theta(n \log n) = \Theta(n \log n)$. Also the algorithm divides a problem of size n into $a = 2$ subproblems of size $\frac{n}{b}$ each, where $b = 2$, and recursively solves (i.e., conquers) them in steps 6 and 7.

Hence, for $n = 2^k$, where $k > 0$ is an integer, the recurrence for $T(n)$ can be written as follows.

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

Though the recurrence above has a base case size of $n = 2$ and not $n = 1$, Master Theorem still applies. To see why, observe that since n is an even number, $T(n)$ can also be described in terms of $\frac{n}{2}$ (i.e., number of pairs of points) instead of n (i.e., number of points). Then $T(n) = T'\left(\frac{n}{2}\right)$, where for $n = 2^l$ with integer $l \geq 0$ the recurrence for $T'(n)$ can be written as follows.

$$T'(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T'\left(\frac{n}{2}\right) + \Theta(n \log n) & \text{otherwise.} \end{cases}$$

One can apply case 2 of Master Theorem (see appendix) with $a = b = 2$ and $k = 1$, and obtain $T'(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right) = \Theta\left(n \log^2 n\right)$.

Hence, $T(n) = T'\left(\frac{n}{2}\right) = \Theta\left(n \log^2 n\right)$.

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

Question 4. [**10 Points**] **An Impossible Priority Queue.** Consider a (comparison-based) priority queue Q that supports the following operations.

Make-Queue(Q)**:** Create an empty queue Q.

Insert(Q, x)**:** Insert item x into Q.

Increase-Key(Q, x, k)**:** Increase the key of item x to k assuming $k \geq$ current key of x.

Find-Min(Q)**:** Return a pointer to an item in Q containing the smallest key.

Delete-Min(Q)**:** Delete an item with the smallest key from Q and return a pointer to it.

4(a) [**10 Points**] Suppose Q supports Insert and Increase-Key operations in $\mathcal{O}(1)$ amortized time each, and Delete-Min operations in $\mathcal{O}(\log n)$ worst-case time each, where n is the number of items in Q. It also supports the Make-Queue operation and every Find-Min operation in $\mathcal{O}(1)$ worst-case time.

Argue that such a priority queue cannot exist.

Solution. We show below that if such a priority queue exists one can use it to sort a set of n numbers in $\mathcal{O}(n)$ time which is impossible since no comparison-based sorting algorithm can sort n numbers asymptotically faster than $\Theta(n \log n)$ time.

Let $A[1:n]$ be an array of n distinct unsorted numbers. We sort the numbers as follows.

1. Make-Queue(Q) *{create an empty priority queue}*
2. ***for*** $i \leftarrow 1$ ***to*** n ***do***
3. create item x with *key*(x) $= A[\ i\ ]$
4. Insert($Q,\ x$) *{insert $A[\ i\ ]$ into Q}*
5. ***for*** $i \leftarrow 1$ ***to*** n ***do***
6. $x \leftarrow$ Find-Min(Q) *{find the item x with the smallest key in Q}*
7. $A[i] \leftarrow$ *key*(x) *{$A[\ i\ ]$ now stores the i-th smallest number in the original input}*
8. Increase-Key($Q,\ x,\ +\infty$) *{increase x's key to something larger than the largest number in the input}*

Clearly, the algorithm above puts the input numbers (given in A) back in $A[1:n]$ in sorted order. Now let us compute its running time. Step 1 takes $\mathcal{O}(1)$ worst-case time. Since step 3 takes $\mathcal{O}(1)$ worst-case time and step 4 takes $\mathcal{O}(1)$ amortized time, the ***for*** loop in steps 2–4 takes $\mathcal{O}(n)$ worst-case time. Steps 6 and 7 take $\mathcal{O}(1)$ worst-case time and step 8 takes $\mathcal{O}(1)$ amortized time. Hence, the ***for*** loop in steps 5–8 runs in $\mathcal{O}(n)$ worst-case time. Thus sorting the n numbers requires only $\mathcal{O}(n)$ time in the worst-case using this priority queue which clearly violates the known $\Theta(n \log n)$ lower bound for comparison-based sorting. Hence, such a priority queue cannot exist.

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

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).