Quiz 2

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 75 minutes to earn 180 points. **However, 150 points is viewed as full mark. Anything above that are bonus points.** Do not spend too much time on any one problem. Read them all first, and attack them in the order that allows you to make the most progress.
- **This is a closed-book exam**. No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions **using non-erasable pens** in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is also okay but not required.
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.
- Turn in your exam paper to your designated AI after the exam, and sign your name on the sign-in sheet.

Problem	Parts	Points	Grade	Grader
Name	0	4		
1	7	63		
2	2	20		
3	1	30		
4	1	30		
5	7	33		
Total	–	180		

Name: ____________________

Problem 1. Single Choice Problems [63 points] (7 parts)

For each of the following questions, choose the unique best answer and write the corresponding letter in the bracket before the problem. There is no need to justify the answers. Each problem is worth 9 points.

(a)( ) Let ω be a 12$^{\text{th}}$ primitive root of unity. Among the following powers of ω, choose the one that is also a 12$^{\text{th}}$ primitive root of unity.

A. ω^2.
B. ω^3.
C. ω^5.
D. ω^9.

Solution: C. Since $\gcd(5, 12) = 1$.

(b)( ) Let n be a positive integer and $\omega = e^{\frac{2\pi i}{2n}}$. What is $1 + \omega + \omega^2 + \cdots + \omega^n$?

A. $\cot(\pi/n)i$.
B. $\cot(\pi/(2n))i$.
C. $\cot(\pi/n)$.
D. $\cot(\pi/(2n))$.
E. $i/\sin(\pi/n)$.
F. $\cos(\pi/(2n))$.

Solution: B. By symmetry we know that the sum must be a imaginary number. Therefore, we can write the sum as

$$\frac{1-\omega^{n+1}}{1-\omega} = \frac{1+\omega}{1-\omega} = \frac{|1+\omega|}{|1-\omega|} \cdot i.$$

Note that $\frac{|1+\omega|}{2} = \cos(2\pi/(4n))$ and $\frac{|1-\omega|}{2} = \sin(2\pi/(4n))$. Hence B is the correct answer.

(c)( ) Consider the problem of multiplying two n-bit binary integers. Choose the item that is the least relevant to the problem.

A. Karatsuba's algorithm.
B. The Fast Fourier Transform algorithm.
C. $\Theta(n^{\log_2 3})$ running time.
D. Dynamic Programming technique.

Solution: D.

(d)( ) Which among the following choices **is not** a property of a problem that can be solved in polynomial time via Dynamic Programming?

A. An optimal solution to the problem contains optimal solutions to the subproblems.
B. Make the choice that seems best at the moment and solve the subproblems that arise later.
C. There are only polynomially many subproblems.

D. There is an order on the subproblems so that larger subproblems are reduced to smaller subproblems in the order.

Solution: B.

(e)(**)** Recall that the Fast Fourier Transform algorithm takes a polynomial $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ as input and evaluates it at all n-th roots of unity. Which of the following statements is **not true** regarding the algorithm?

A. n has to be in the form of $n = 2^k (k = 0, 1, 2, \dots)$ in the recursive algorithm.

B. Space complexity is $\Theta(n \log n)$.

C. The same algorithm (after small modification) can be used to compute the coefficients of $A(x)$ once given the evaluations at n-roots of unity.

D. Times complexity is $\Theta(n \log n)$ arithmetic operations.

Solution: B. At the i-th recursive call, the algorithm solves a problem of size $\frac{n}{2^i}$ and therefore uses space $O\left(\frac{n}{2^i}\right)$ (not counting space used by further recursive calls). Therefore the total space complexity is $\sum_{i=0}^{\log_2 n} O\left(\frac{n}{2^i}\right) = O(n)$.

(f)(**)** Consider the problem of multiplying two $n \times n$ matrices. The straightforward algorithm takes $O(n^3)$ time. Now we would like to design a divide-and-conquer algorithm that divides the problem into q subproblems, where in each subproblem the task is to multiply two $\frac{n}{2} \times \frac{n}{2}$ matrices. In order to make the algorithm run faster than $O(n^3)$ time, q has to be at most

A. 2.

B. 3.

C. 7.

D. 8.

Solution: C.

(g)(**)** We have learnt the divide-and-conquer algorithm for 2-Dimensional Closest Pair of Points. Now let us consider the same problem in 3-D space (i.e. each point has 3 coordinates (x_i, y_i, z_i)). The following algorithm is a simple analogue of the algorithm for 2-D, where the difference is marked by **boldface**.

i. Choose x_0 and split the given set of n points into 2 sets of $n/2$ points by the plane $x = x_0$.

ii. Solve the two subproblems recursively. Let ϵ be the smallest distance between pair of points within each subproblem.

iii. Let T be the set of points (x_i, y_i, z_i) such that $|x_i - x_0| < \epsilon$. Sort all points in T **according to** $\boldsymbol{y_i + z_i}$.

iv. Check the distance between each point in T and its **100 neighbors** in the sorted list. Let δ be the smallest distance found.

v. Return $\min(\epsilon, \delta)$.

Choose the correct statement about this algorithm.

A. It **can** be made to run in $O(n \log n)$ time and **gives** the correct answer.

B. It **cannot** be made to run in $O(n \log n)$ time but **gives** the correct answer.

C. It **can** be made to run in $O(n \log n)$ time but **does not give** the correct answer.

D. It **cannot** be made to run in $O(n \log n)$ time and **does not give** the correct answer.

Solution: C.

Problem 2. Short Answer Problems [20 points] (2 parts)

(a) [10 points] Solve the following recurrence.

$$\begin{cases} T(n) = 4T(n/2) + n^2 & (n \geq 2) \\ T(n) = 1 & (n \leq 1) \end{cases} .$$

Your answer should be in the form of $\Theta(f(n))$ and you do not have to show the intermediate steps.

Solution: $T(n) = \Theta(n^2 \log n)$.

(b) [10 points] Solve the following recurrence.

$$\begin{cases} T(n) = T(n/2) + T(n/3) + T(n/6) + n & (n \geq 6) \\ T(n) = 1 & (n < 6) \end{cases} .$$

Your answer should be in the form of $\Theta(f(n))$ and you do not have to show the intermediate steps.

Solution: $T(n) = \Theta(n \log n)$.

Problem 3. Fast Fourier Transform with Base 3 [30 points]

We have learned the Fast Fourier Transform (FFT) algorithm to evaluate a given polynomial $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ at the n n-th roots of unity $1, \omega, \omega^2, \ldots, \omega^{n-1}$ (where $\omega = e^{\frac{2\pi i}{n}}$), using $O(n \log n)$ arithmetic operations. The algorithm we saw in class works only when $n = 2^k$ is an integer power of 2.

In this problem, you are asked to modify the FFT algorithm presented in class so that it works for the case when $n = 3^k$ is an integer power of 3, still using $O(n \log n)$ arithmetic operations.

Please note that our task is different from polynomial multiplication so the simple padding 0's trick may not work.

Solution:

The idea goes similar as the one presented in Lecture 13.

Here we introduce $A_1(x) = a_0 + a_3x + \cdots + a_{n-3}x^{n/3-1}$, $A_2(x) = a_1 + a_4x + \cdots + a_{n-2}x^{n/3-1}$, and $A_3(x) = a_2 + a_5x + \cdots + a_{n-1}x^{n/3-1}$. Hence, it holds that

$$A(w^j) = A_1((w^3)^j) + w^j \cdot A_2((w^3)^j) + w^{2j} \cdot A_3((w^3)^j).$$

Hence to evaluate $A(x)$ at $x = w^1, w^2, \ldots, w^n = 1$, we can first evaluate $A_1(x), A_2(x), A_3(x)$ at $x = w^3, w^6, \ldots, (w^3)^n$, and then use $O(n)$ time to get the final n evaluations of $A(x)$. Note that $w^{n+3} = w^3, \ldots, w^{3n} = 1$. There are totally $n/3$ different values of $w^3, w^6, \ldots, (w^3)^n$. And this reduces the problem of evaluating the degree-$(n-1)$ polynomial $A(x)$ at n n-th roots of unity to the three subproblems of evaluating degree-$(n/3-1)$ polynomials at $(n/3)$ $(n/3)$-th roots of unity.

Let $T(n)$ denote the time complexity of evaluating $A(x)$ at $x = w^1, w^2, \ldots, w^n = 1$. We have $T(n) = 3T(n/3) + O(n)$ from which we can see $T(n) = O(n \log n)$ according to the Master Theorem.

Problem 4. Merging sorted lists [30 points]

When learning the Merge-Sort algorithm, we saw how to merge 2 sorted lists in linear time. Now let us extend the merging algorithm to deal with more sorted lists.

Suppose you are given $k \geq 2$ lists as input, where each list L_i ($i = 1, 2, 3, \ldots, k$) contains n/k numbers sorted in non-decreasing order. Please describe an algorithm that runs in $O(n \log k)$ time and merges the k sorted lists into one sorted list L of all n numbers. You do not have to formally prove the correctness of the algorithm. However, you should prove the runtime of your algorithm.

Your algorithm description is worth 18 points; the runtime analysis is worth 12 points. You will get the latter 12 points only when your algorithm is correct (or almost correct).

Solution: The following recursive algorithm merges a sorted lists ($L_1, L_2, \ldots, L_a$), where $\mathrm{Merge}(P, Q)$ is the same algorithm we used in Merge-Sort.

$\mathrm{Merge} - \mathrm{Lists}(a, L_1, L_2, \ldots, L_a)$

- IF a = 1 THEN RETURN L_1
- LET $m = [a/2]$
- LET $P = \mathrm{Merge} - \mathrm{Lists}(m, L_1, L_2, \ldots, L_m)$
- LET $Q = \mathrm{Merge} - \mathrm{Lists}(a - m, L_{m+1}, L_{m+2}, \ldots, L_a)$
- RETURN $\mathrm{Merge}(P, Q)$

Let $T(a)$ be the time complexity of $\mathrm{Merge} - \mathrm{Lists}$ with a lists as input. We have $T(1) = O(n/k)$ as boundary condition. For $a \geq 1$, we have $T(a) = 2T(a/2) + O(an/k)$ (since there are an/k numbers in total for $\mathrm{Merge}(P, Q)$ to process). Solving this recurrence, we get $T(a) = O((n/k) \cdot a \log a)$. Therefore, the time complexity to merge k lists is $T(k) = O(n \log k)$.

Problem 5. Segmented Least Errors [33 points] (7 parts) In class we discussed the Segmented Least Squares problem. Here is a quick recap of the problem. There are n data points $P_1, P_2, P_3, \dots, P_n$, and we would like to partition them into a few segments. Suppose we partition the data points into L segments as follows (where $a_0 = 1$ and $a_L = n + 1$),

$$[a_0, a_1), [a_1, a_2), [a_2, a_3), \dots, [a_{L-1}, a_L).$$

We define the total error to be

$$E = \sum_{i=1}^{L} e_{a_{i-1}, a_i - 1} + cL$$

where c and e_{ij}'s are given as input. The goal is to find out the partition with minimum cost E.

(a) [5 points] Let $\mathrm{OPT}_E[j]$ be the optimal solution (i.e. minimum cost) for the subproblem defined on $P_1, P_2, P_3, \dots, P_j$. Please write down the Bellman Equation for $\mathrm{OPT}_E[j]$ ($1 \le j \le n$) based on the optimal solution of smaller subproblems.

Solution:

$$\mathrm{OPT}_E[j] = \min_{i: 1 \le i \le j} \left\{ \mathrm{OPT}_E[i-1] + e_{i,j} + c \right\}.$$

Now let us consider the question when the total cost is non-linear in L. Suppose we would like to minimize the the new cost defined as follows

$$F = \sum_{i=1}^{L} e_{a_{i-1}, a_i - 1} + c\sqrt{L}.$$

In order to solve this new problem, we need to use the technique of "adding a variable" as we did in the Knapsack problem. We define $\mathrm{OPT}_F[j, \ell]$ to be the optimal solution (i.e. minimum cost) for the subproblem of partitioning $P_1, P_2, P_3, \dots, P_j$ into *exactly* ℓ segments.

(b) [4 points] How many subproblems are there in total (up to the $\Theta(\cdot)$ notation)?

Solution: $\Theta(n^2)$ since $1 \le \ell \le j \le n$.

(c) [4 points] Suppose we have solved all subproblems. What is the optimal solution (i.e. minimum cost) to the original problem (to minimize F)?

Solution:

$$\min_{\ell : 1 \leq \ell \leq n} \mathrm{OPT}_F[n, \ell].$$

(d) [4 points] When $\ell = 1$, write the boundary condition for $\mathrm{OPT}_F[j, 1]$ ($1 \leq j \leq n$).

Solution:

$$\mathrm{OPT}_F[j, 1] = e_{1,j} + c.$$

(e) [7 points] When $\ell > 1$, write the Bellman Equation for $\mathrm{OPT}_F[j, \ell]$ ($\ell \leq j \leq n$). No proofs needed.

Solution:

$$\mathrm{OPT}_F[j, \ell] = \min_{i : \ell \leq i \leq j} \left\{ \mathrm{OPT}_F[i-1, \ell-1] + e_{i,j} + c\left(\sqrt{\ell} - \sqrt{\ell - 1}\right) \right\}.$$

(f) [4 points] What is a proper order to solve all the subproblems according to your Bellman Equation?

Solution: Either increasing order of j or increasing order of ℓ works.

(g) [5 points] What is the time complexity of the Dynamic Programming algorithm based on your Bellman Equation (up to the $\Theta(\cdot)$ notation)?

Solution: $\Theta(n^3)$, since there are $\Theta(n^2)$ subproblems and it takes $\Theta(n)$ time to solve each subproblem.

SCRATCH PAPER

SCRATCH PAPER