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	5	50		
2	3	30		
3	1	32		
4	1	32		
5	1	32		
Total	–	180		

Name: _____

Problem 1. Single-Choice Problems [50 points] (5 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 10 points.

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

Solution: D. 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 D is the correct answer.

(b)( ) 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. There are only polynomially many subproblems.
C. Make the choice that seems best at the moment and solve the subproblems that arise later.
D. There is an order on the subproblems so that larger subproblems are reduced to smaller subproblems in the order.

Solution: C.

(c)( ) Let us suppose we have the following Bellman Equation to solve an optimization problem via Dynamic Programming.

$$\mathrm{OPT}[i] = \min_{j=1,2,4,8,\ldots;j\le i} \{\mathrm{OPT}[i-j] + \mathrm{loss}(i,j)\},$$

where $\mathrm{loss}(i,j)$ can be evaluated in $O(1)$ time for each (i,j) pair, and the boundary condition is $\mathrm{OPT}[0] = 0$. The time complexity to evaluate $\mathrm{OPT}[n]$ is

A. $\Theta(n)$. B. $\Theta(n \log n)$.
C. $\Theta(n \log^2 n)$. D. $\Theta(n^2)$.

Solution: B.

(d)( ) Let us consider a long and narrow $2 \times n$ grid floor and we would like to cover it using 1×2 dominoes. We would like to tile the floor so that each grid is covered by exactly one domino, so it is easy to see that we need n dominoes.

For example, when $n = 2$, there are 2 different ways to cover the floor, drawn as follows.

When $n = 3$, there are 3 different ways to cover the floor, drawn as follows.

Now you are asked to count the number of different covering choices for $n = 10$.

A. 55. B. 67. C. 74.
D. 77. E. 89. F. 97.

Solution: E.

(e)( ) We have learnt the divide-and-conquer algorithm for 2-Dimensional Closest Pair of Points with time complexity $O(n \log n)$. Which one of the following procedures is not part of that algorithm?

A. Choose x_0 and split the given set of n points into 2 sets of $n/2$ points by the plane $x = x_0$.
B. Solve the two subproblems recursively. Let ϵ be the smallest distance between pair of points within each subproblem.
C. Let T be the set of points (x_i, y_i) such that $|x_i - x_0| \leq \epsilon$. Sort all points in T according to y_i.
D. Check the distance between each point in T and its 7 successors in the sorted list. Let δ be the smallest distance found.
E. Return $\min\{\epsilon, \delta\}$.

Solution: C.

Problem 2. Recurrences [30 points] (3 parts) Solve the following recurrences. Your answers should be in the form of $\Theta(f(n))$ and you do not have to show the intermediate steps.

(a) [10 points]

$$\begin{cases} T(n) = 9T(\lfloor n/3 \rfloor) + n^2 \log n & (n \geq 3) \\ T(n) = 1 & (n \leq 1) \end{cases}.$$

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

(b) [10 points]

$$\begin{cases} T(n) = 6T(\lfloor n/4 \rfloor) + n & (n \geq 4) \\ T(n) = 1 & (n \leq 1) \end{cases}.$$

Solution: $T(n) = \Theta(n^{\log_4 6})$.

(c) [10 points]

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

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

Problem 3. Counting Jumping Pairs [32 points]

We are given a sequence of n numbers $a_1, a_2, a_3, \ldots, a_n$. Now we define a pair (i, j) to be a *jumping pair* if $1 \leq i < j \leq n$ and $a_i + M \leq a_j$ where M is a parameter given as input.

For example, when $n = 6, M = 3$ and the list of n numbers are $1, 5, 7, 3, 2, 6$, there are 5 jumping pairs: $(1, 2)$, $(1, 3)$, $(1, 6)$, $(4, 6)$, $(5, 6)$.

Please give an $O(n \log n)$-time algorithm to count the number of jumping pairs. You will receive partial credits if your algorithm is correct and runs in time $O(n \log^2 n)$.

Solution:

The idea goes similar as the one presented in Lecture 10.

Let the input sequence be A. Let us first divide the problem into 2 subproblems using the most straightforward idea. We take $k = \lfloor n/2 \rfloor$ and divide A into an array $X[] = A[1 \to k]$ of k numbers and an array $Y[] = A[k+1 \to n]$ of $(n-k)$ numbers. We first (recursively) solve the 2 subproblems, i.e. figure out c_1 : the number of jumping pairs in $X[]$ and c_2: the number of jumping pairs in $Y[]$.

However, the total number of jumping pairs in $A[]$ consists of c_1, c_2, and another part, namely c_3: the number of pairs $X[i]$ and $Y[j]$ such that $X[i] + M \leq Y[j]$.

Here is the algorithm.

Algorithm 1 Sort-and-Count($n, A[], M$)

1: **if** $n \leq 1$ **then**
2: **Return:** $(A, 0)$.
3: $k = \lfloor n/2 \rfloor$.
4: $(X[], c_1) \leftarrow$ Sort-and-Count$(k, A[1 \to k], M)$.
5: $(Y[], c_2) \leftarrow$ Sort-and-Count$(n-k, A[k+1 \to n], M)$.
6: $(Z[], c_3) \leftarrow$ Merge-and-Count$(k, X[], n-k, Y[], M)$.
7: **Return:** $(Z[], c_3)$.

Algorithm 2 Merge-and-Count($n_1, X[], n_2, Y[], M$)

```
1: i ← 1, j ← 1, k ← 1.
2: while i ≤ n_1 or j ≤ n_2 do                                   ▷ Merge
3:     if j > n_2 or (i ≤ n_1 and X[i] < Y[j]) then
4:         Z[k] ← X[i].
5:         i ← i + 1.
6:     else
7:         Z[k] ← Y[j].
8:         j ← j + 1.
9:     k ← k + 1.
10: i ← 1, j ← 1, c_3 ← 0.
11: while i ≤ n_1 or j ≤ n_2 do                                  ▷ Count
12:     if X[i] + M > Y[j] then
13:         j ← j + 1.
14:     else
15:         c_3 ← c_3 + n_2 − j + 1.
16:         i ← i + 1.
17: Return: (Z[], c_3).
```

We see that the procedure is almost the same as the merge sort algorithm, except for that we also keep track of the number of jumping pairs along the way which takes only $O(n)$ time. Since we also sorted the 2 arrays in the divide steps, Merge-and-Count will take 2 sorted arrays as input, meeting our original assumption. Of course, Merge-and-Count will also return a sorted array.

The time complexity of Sort-and-Count is the same as that of merge sort, which is $O(n \log n)$.

Problem 4. Fast Fourier Transform with Base 3 [32 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 5. Traveling by a Canoe [32 points]

You are traveling by a canoe down a river and there are n trading posts along the way. Before starting your journey, you are given for each $1 \leq i < j \leq n$, the fee $f_{i,j}$ for renting a canoe from post i to post j. Note that you cannot rent a canoe to go from post j to post i when $i < j$. The rental fees are arbitrary, and may be negative as the rental company is advertising a few routes. For example, if $f_{1,3} = 10$, you pay 10 dollars to go from post 1 to post 3. If $f_{3,5} = -5$, you receive 5 dollars incentive to go from post 3 to post 5. You begin at post 1 and must end at post n (using rented canoes). Your goal is to minimize the total rental cost (it may be negative which means that you earn money). Give a polynomial-time dynamic programming algorithm for this problem. Be sure to prove that your algorithm yields an optimal solution and analyze the time complexity.

Solution:

Let $m[i]$ be the rental cost for the best solution to go from post i to post n for $1 \leq i \leq n$. The final answer is in $m[1]$. We can recursively, define $m[i]$ as follows:

$$m[i] = \begin{cases} 0 & \text{if } i = n \\ \min_{i<j\leq n}(f_{i,j} + m[j]) & \text{otherwise} \end{cases}$$

We now prove this is correct. The canoe must be rented starting at post i (the starting location) and then returned next at a station among $i+1, \ldots, n$. In the recurrence we try all possibilities (with j being the station where the canoe is next returned). Furthermore, since $f_{i,j}$ is independent from how the subproblem of going from post $j, \ldots, n$ is solved, we have the optimal substructure property.

For the time complexity there are n subproblems to be solved each of which takes $O(n)$ time. These subproblems can be computed in the order $m[n], m[n-1], \ldots, m[1]$. Hence the overall time complexity is $O(n^2)$.

SCRATCH PAPER

SCRATCH PAPER