Problem Set 3

Divide & Conquer • Dynamic Programming

Course	CS 224 — Algorithms & Data Structures
Instructor	Prof. Maria Chen
Student	Arjun Sharma
Student ID	2024A7PS0134H
Date	March 15, 2025
Institution	Department of Computer Science Stanford University

---

Problem 1: Recurrence Relations & the Master Theorem

Consider the recurrence $T(n) = 2\,T(n/2) + n\log_2 n$ with base case $T(1) = 1$. Determine the asymptotic behavior of $T(n)$.

Solution. We apply the Master Theorem. The recurrence has the form $T(n) = a\,T(n/b) + f(n)$ with $a = 2$, $b = 2$, and $f(n) = n\log_2 n$.

The critical exponent is $c_{\text{crit}} = \log_b a = \log_2 2 = 1$. Comparing $f(n) = n\log_2 n$ with $n^{c_{\text{crit}}} = n$:

$$\frac{f(n)}{n^{c_{\text{crit}}}} = \frac{n\log_2 n}{n} = \log_2 n$$

This grows slower than any n^{ε} for $\varepsilon > 0$, yet faster than a constant. This places us in **Case 2** of the Master Theorem (extended form): when $f(n) = \Theta\big(n^{\log_b a}\log^{k} n\big)$ with $k \geq 0$, we have

$$T(n) = \Theta\big(n^{\log_b a}\log^{k+1} n\big).$$

Here $k = 1$, so

$$\boxed{T(n) = \Theta\big(n\log^2 n\big)}.$$

Remark 0.1. The recurrence $T(n) = 2\,T(n/2) + n\log n$ arises in the analysis of certain divide-and-conquer sorting networks and in the expected-time analysis of randomized quicksort with a median-of-three pivot strategy.

Problem 2: Closest Pair of Points

Given a set $P = \{p_1, \dots, p_n\}$ of n points in $\mathbb{R}^2$, design a divide-and-conquer algorithm to find the closest pair under the Euclidean distance $d(p, q) = \|p - q\|_2$. Your algorithm must run in $\mathcal{O}(n\log n)$ time.

Solution. The algorithm proceeds in three phases:

1. **Pre-sort.** Build two sorted copies of P: P_x sorted by x-coordinate and P_y sorted by y-coordinate. This costs $\mathcal{O}(n\log n)$.
2. **Divide.** Find the median x-coordinate x_m and partition P into the left half P_L and right half P_R. Also split P_y into $P_{y,L}$ and $P_{y,R}$ while preserving the y-sorted order.

3. **Conquer & Combine.** Recursively compute $\delta_L = \textsc{ClosestPair}(P_L)$ and $\delta_R = \textsc{ClosestPair}(P_R)$. Let $\delta = \min(\delta_L, \delta_R)$. Build the *strip* $S = \{\, p \in P_y \mid |x(p) - x_m| < \delta \,\}$ and check every point in S against the next 7 points in y-order. Return the minimum distance found.

Algorithm 1: ClosestPair(P_x, P_y)

Input: Sets P_x, P_y of points sorted by x and y, respectively.
Output: Minimum Euclidean distance between any two distinct points in P.

if $|P_x| \le 3$ **then**
    Compute all pairwise distances by brute force;
    return the minimum;
Let $x_m \leftarrow$ median x-coordinate of P_x;
Partition P into P_L, P_R about the vertical line $x = x_m$;
Build $P_{y,L}$ and $P_{y,R}$ from P_y accordingly;
$\delta_L \leftarrow \textsc{ClosestPair}(P_{x,L},\ P_{y,L})$;
$\delta_R \leftarrow \textsc{ClosestPair}(P_{x,R},\ P_{y,R})$;
$\delta \leftarrow \min(\delta_L,\ \delta_R)$;
$S \leftarrow \{\, p \in P_y \mid |x(p) - x_m| < \delta \,\}$ `// the strip`
for $i \leftarrow 1$ **to** $|S|$ **do**
    for $j \leftarrow i + 1$ **to** $\min(i + 7,\ |S|)$ **do**
        $\delta \leftarrow \min(\delta,\ d(S[i],\ S[j]))$;
return δ;

Correctness of the combine step. The only pairs that could beat δ are those where one point lies in P_L and the other in P_R *and* both lie within distance δ of the split line $x = x_m$. Every such pair belongs to the strip S.

Sort S by y-coordinate. For any point $p \in S$, consider a $2\delta \times \delta$ rectangle whose bottom edge passes through p. This rectangle can contain at most 8 points (4 per side of the split) without any two being closer than δ. Hence checking the next 7 points in y-order suffices to find the closest split pair. The recurrence

$$T(n) = 2\,T(n/2) + \mathcal{O}(n)$$

solves to $T(n) = \mathcal{O}(n \log n)$. □

Problem 3: Longest Common Subsequence (LCS)

Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ over an alphabet Σ, find the length of the longest subsequence common to both strings. A subsequence need not consist of contiguous characters.

(a) Recurrence. Let $\mathrm{LCS}(i, j)$ denote the LCS length for the prefixes $X[1 \ldots i]$ and $Y[1 \ldots j]$. Then

$$\mathrm{LCS}(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ \mathrm{LCS}(i-1,\ j-1) + 1 & \text{if } x_i = y_j, \\ \max\big(\mathrm{LCS}(i-1,\ j),\ \mathrm{LCS}(i,\ j-1)\big) & \text{otherwise.} \end{cases}$$

(b) Implementation. The DP solution fills an $(m+1) \times (n+1)$ table bottom-up in $\mathcal{O}(mn)$ time.

Listing 1: Bottom-up DP for LCS length and reconstruction.

```python
def lcs(X: str, Y: str) -> tuple[int, str]:
```

```
    m, n = len(X), len(Y)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    # Reconstruct one LCS
    i, j = m, n
    seq = []
    while i > 0 and j > 0:
        if X[i - 1] == Y[j - 1]:
            seq.append(X[i - 1])
            i -= 1; j -= 1
        elif dp[i - 1][j] >= dp[i][j - 1]:
            i -= 1
        else:
            j -= 1

    return dp[m][n], ''.join(reversed(seq))
```

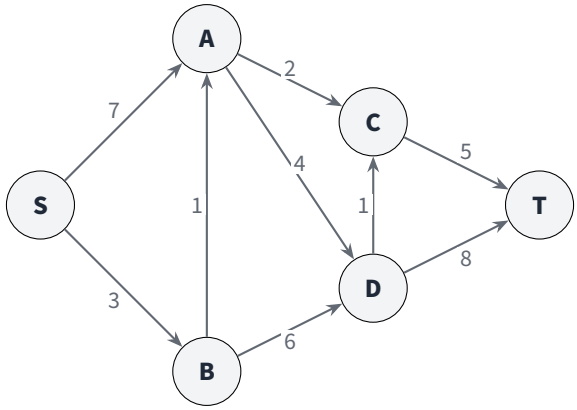
(c) Worked Example. Let $X = \texttt{AGGTAB}$ and $Y = \texttt{GXTXAYB}$. The DP table is:

		G	X	T	X	A	Y	B
	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0	1	1	2	2	3	3	**4**

The LCS length is 4; one LCS is GTAB (traced by following the highlighted cell back through matches). Time complexity is $\mathcal{O}(mn) = \mathcal{O}(42)$ for $|X| = 6, |Y| = 7$; space can be reduced to $\mathcal{O}(\min(m, n))$ by storing only two rows.

Problem 4: Single-Source Shortest Paths

Consider the directed graph $G = (V, E)$ shown below with non-negative edge weights. Run **Dijkstra's algorithm** starting from node S.



Solution. We maintain a min-priority queue of $(d[v], v)$ pairs and relax edges as we extract the minimum. The state at each iteration:

Step	**Extract**	**Distances** $(d[S],\ d[A],\ d[B],\ d[C],\ d[D],\ d[T])$
Init	—	$(0,\ \infty,\ \infty,\ \infty,\ \infty,\ \infty)$
1	S	$(0,\ 7,\ 3,\ \infty,\ \infty,\ \infty)$
2	B	$(0,\ 4,\ 3,\ \infty,\ 9,\ \infty)$
3	A	$(0,\ 4,\ 3,\ 6,\ 8,\ \infty)$
4	C	$(0,\ 4,\ 3,\ 6,\ 8,\ 11)$
5	D	$(0,\ 4,\ 3,\ 6,\ 7,\ 11)$
6	T	$(0,\ 4,\ 3,\ 6,\ 7,\ 11)$

The shortest-path distances from S are:

S	A	B	C	D	T
0	4	3	6	7	11

The shortest path $S \to T$ is $S \to B \to A \to C \to T$ with total weight $3 + 1 + 2 + 5 = 11$. Alternative route $S \to B \to D \to C \to T$ has weight $3 + 6 + 1 + 5 = 15$, confirming Dijkstra's correctness on this instance.

End of Problem Set 3