

Question 1: Selection Algorithms

We implement and compare two algorithms for finding the k -th smallest element in an array of size $N = 100$: Randomized Selection (Rand-Select) and Deterministic Selection (Median of Medians).

1.1. Randomized Selection (Rand-Select)

This algorithm uses a randomized partition scheme similar to Quicksort. It has an expected runtime of $O(N)$ but a worst-case of $O(N^2)$.

```

1 int randomized_partition(vector<int>& arr, int left, int right, mt19937& g) {
2     uniform_int_distribution<int> dist(left, right);
3     int pivot_idx = dist(g);
4
5     int pivot = arr[pivot_idx];
6     swap(arr[pivot_idx], arr[right]);
7     int partition_idx = left;
8
9     for (int i = left; i < right; i++) {
10         if (arr[i] < pivot) {
11             swap(arr[i], arr[partition_idx++]);
12         }
13     }
14     swap(arr[partition_idx], arr[right]);
15     return partition_idx;
16 }
17
18 int rand_select(vector<int>& arr, int p, int q, int i, mt19937& g) {
19     if (p == q) return arr[p];
20     int r = randomized_partition(arr, p, q, g);
21     int k = r - p + 1;
22     if (i == k) return arr[r];
23     else if (i < k) return rand_select(arr, p, r - 1, i, g);
24     else return rand_select(arr, r + 1, q, i - k, g);
25 }
```

Listing 1: Rand-Select Implementation

Results: Given the input array A (shuffled permutation of $1 \dots 100$), the algorithm correctly identifies the k -th smallest elements.

Table 1: Results for Rand-Select

k -th Order Statistic	Value
1	1
2	2
3	3
...	...
10	10

1.2. Deterministic Selection (Median of Medians)

This algorithm guarantees $O(N)$ worst-case time by selecting a good pivot (the median of medians of groups of 5).

```

1 int median_of_medians(vector<int>& arr, int p, int q) {
2     int n = q - p + 1;
3     if (n <= 5) return median(arr, p, q);
4
5     int groups = (n + 4) / 5;
6     vector<int> medians(groups);
7     for (int i = 0; i < groups; i++) {
8         int l = p + i * 5;
9         int r = min(l + 4, q);
10        medians[i] = median(arr, l, r);
11    }
12    return median_of_medians(medians, 0, groups - 1);
13 }
14
15 int select(vector<int>& arr, int p, int q, int i) {
16     if (p == q) return arr[p];
17     int x = median_of_medians(arr, p, q);
18     int pivot_idx = partition(arr, p, q, x);
19     int k = pivot_idx - p + 1;
20
21     if (i == k) return arr[pivot_idx];
22     else if (i < k) return select(arr, p, pivot_idx - 1, i);
23     else return select(arr, pivot_idx + 1, q, i - k);
24 }
```

Listing 2: Deterministic Select Implementation

Results: The deterministic algorithm also correctly identifies the k -th smallest elements from the shuffled input.

Table 2: Results for Deterministic Select

k -th Order Statistic	Value
1	1
2	2
...	...
10	10

Question 2: Longest Common Subsequence (LCS)

We implement the dynamic programming approach to find **all** Longest Common Subsequences between two strings.

- **Input:** $S1 = "ABCBDAB"$, $S2 = "BDCABA"$
- **Algorithm:** We construct the DP table $C[i][j]$ storing the length of LCS of $S1[1..i]$ and $S2[1..j]$. To find all subsequences, we backtrack from $C[n][m]$, branching whenever multiple optimal paths ($C[i - 1][j] == C[i][j - 1]$) exist.

```

1 void print_all_lcs(const vector<vector<int>>& C, const vector<vector<char>>& b,
2                     const string& s1, int i, int j, string curr_lcs,
3                     unordered_set<string>& all_lcs) {
4     if (i == 0 || j == 0) {
```

```

5     reverse(curr_lcs.begin(), curr_lcs.end());
6     all_lcs.insert(curr_lcs);
7     return;
8 }
9 // If characters match, they must be part of LCS
10 if (s1[i-1] == s2[j-1]) { // Logic adapted from b[i][j] == 'D'
11     curr_lcs.push_back(s1[i - 1]);
12     print_all_lcs(C, b, s1, i - 1, j - 1, curr_lcs, all_lcs);
13 } else {
14     // Branching logic for multiple paths
15     if (C[i - 1][j] == C[i][j])
16         print_all_lcs(C, b, s1, i - 1, j, curr_lcs, all_lcs);
17     if (C[i][j - 1] == C[i][j])
18         print_all_lcs(C, b, s1, i, j - 1, curr_lcs, all_lcs);
19 }
20 }
```

Listing 3: Finding All LCS

Results

Length of LCS: 4

Unique LCS found:

1. BCBA
2. BDAB
3. BCAB