Greedy Algorithm Examples

1. Suppose \( x_1, x_2, \ldots, x_n \) are real numbers. Propose a greedy algorithm to find the minimum number of closed unit intervals whose union contains all the points. Explain how the algorithm addresses the optimal substructure and greedy properties.

We sort the numbers in increasing order but retain the notation \( x_i \); that is, \( x_1 \leq x_2 \leq \ldots \leq x_n \). We consider subproblems \( X_i \) of the form \( x_i \leq x_{i+1} \leq \ldots \leq x_n \), where \( i \in \{1, 2, \ldots, n\} \). We now claim that a greedy choice for this subproblem is the closed unit interval \([x_i, x_i + 1]\). Indeed, suppose \( A_i \), a collection of closed unit intervals, is an optimal selection for subproblem \( X_i \) that does not contain \([x_i, x_i + 1]\). Some member of \( A_i \) however must cover \( x_i \), say \( x_i \in [y, y + 1] \in A_i \). Since \( A_i \) does not contain the greedy choice, \( y < x_i \leq y + 1 \). Since \( x_i \) is the smallest member of subproblem \( X_i \), we can move \([y, y + 1]\) rigidly to the right until it occupies the space \([x_i, x_i + 1]\). That is, no members of \( X_i \) are lost on the left and any gained to the right may overlap other intervals in \( A_i \). In any case, the new set of intervals still covers all the members of \( X_i \), is of the same optimal size, and contains the greedy choice. Therefore, the greedy property holds. Moreover, if any elements remain in \( X_i \setminus [x_i, x_i + 1] \), say \( x_j \leq \ldots \leq x_n \), then by the cut-and-paste procedure, an optimal solution to this subproblem \( X_j \) combined with the greedy interval \([x_i, x_i + 1]\) must constitute an optimal solution to \( X_i \). That is, the optimal substructure property holds.

The following algorithm generates an optimal solution in linear time.

```java
UnitIntervals(x) { // x is x[1] ... x[n]
    n = length(x);
    println([x[1], x[1] + 1]);
    y = x[1] + 1;
    for (i = 2; i <= n; i++)
        if (x[i] > y) {
            println(x[i], x[i] + 1);
            y = x[i] + 1;
        }
}
```

2. Suppose \( A = \{a_1, a_2, \ldots, a_n\} \) and \( B = \{b_1, b_2, \ldots, b_n\} \) are two sets of positive integers. Let \( \pi \) denote a permutation of \( \{1, 2, \ldots, n\} \). For a given permutation \( \pi \), let

\[
V_\pi = \prod_{i=1}^{n} a_i^{b_{\pi(i)}}.
\]

Propose a greedy algorithm that will maximize \( V_\pi \) over all possible permutations \( \pi \). Explain how the algorithm addresses the optimal substructure and greedy properties.

We sort both sets in decreasing order to obtain \( A = \{a_1 \geq a_2 \geq \ldots \geq a_n\} \) and \( B = \{b_1 \geq b_2 \geq \ldots \geq b_n\} \) and consider subproblems \((A_i, B_i)\) where \( A_i = \{a_i \geq a_{i+1} \geq \ldots \geq a_n\} \) and \( B_i = \{b_1 \geq b_{i+1} \geq \ldots \geq b_n\} \). We now claim that a greedy choice for this subproblem matches \( a_i \) with \( b_i \). That is, the optimal \( \pi \) has \( \pi(i) = i \). Indeed, suppose that an optimal \( \pi' \) has \( \pi'(i) = j \neq i \). Since \( \pi' \) must map something to \( i \), suppose \( \pi'(k) = i \), where necessarily \( k > i \). Then, \( a_i \geq a_k \) and \( b_{\pi'(i)} = b_j \leq b_i \).
Therefore,

\[
\frac{a_i b_j a_k}{a_i b_j a_k} = \frac{a_i b_j}{a_i b_j} \geq a_i b_j a_k = a_k = 1
\]

\[
a_i b_j a_k \leq a_i b_j
\]

\[
V_{\pi'} = a_i b_{\pi'(i)} a_k b_{\pi'(k)} \prod_{s \notin \{i, k\}} b_{\pi'(s)} = a_i b_i a_k \prod_{s \notin \{i, k\}} b_{\pi'(s)} \leq a_i b_i a_k \prod_{s \notin \{i, k\}} b_{\pi'(s)} = \prod_{s=1}^{n} a_s b_{\pi'(s)} = V_{\pi''},
\]

where \(\pi''\) maps \(i\) to \(i\), \(k\) to \(j\), and remaining values to the same targets as \(\pi'\). Since \(V_{\pi'}\) was optimal, we must have \(V_{\pi''} = V_{\pi'}\), and therefore \(\pi''\) is also optimal. Moreover, \(\pi''\) contains the greedy choice since it maps \(i\) to \(i\). We conclude that the greedy property holds. Also, after matching establishing \(\pi(i) = i\) for an optimal permutation, this permutation must continue to provide an optimal solution for the subproblem \((A_{i+1}, B_{i+1})\). Otherwise, by cut-and-paste, we could construct a solution to \((A_i, B_i)\) that yields a \(V\)-value larger than the known optimal solution – a contradiction. Consequently, the optimal substructure property also holds. The following \(\Theta(n \log n)\) algorithm outputs the optimal permutation.

```
OptimalPerm(a, b) { // a is a[1] ... a[n], b is b[1] ... b[n]
    n = length(a);
    for (i = 1; i <= n; i++) {
        x[i, 1] = a[i]; x[i, 2] = i;
        y[i, 1] = b[i]; y[i, 2] = i;
    }
    sort rows of x on first element in descending order;
    sort rows of y on first element in descending order;
    for (i = 1; i <= n; i++) {
        z[i, 1] = x[i, 2];
        z[i, 2] = y[i, 2];
    }
    sort rows of z on first element in ascending order;
    for (i = 1; i <= n; i++)
        println(z[i, 2]); // each output line is a mapping within the optimal permutation
}
```