

13. Our goal is now to consider the Knapsack problem, and develop a method for computing the actual items to be taken in $O(L)$ space and $O(nL)$ time.

- (a) Consider the following problem. The input is the same as for the knapsack problem, a collection of n items I_1, \dots, I_n with weights w_1, \dots, w_n , and values v_1, \dots, v_n , and a weight limit L . The output is in two parts. First you want to compute the maximum value of a subset S of the n items that has weight at most L , as well as the weight of this subset. Let us call this value and weight v_a and w_a . Secondly for this subset S you want to compute the weight and value of the items in $\{I_1, \dots, I_{n/2}\}$ that are in S . Let us call this value and weight v_b and w_b . So your output will be two weights and two values. Give an algorithm for this problem that uses space $O(L)$ and time $O(nL)$.

12. Consider the code for the Knapsack program given in the class notes.

- (a) Explain how one can actually find the highest valued subset of objects, subject to the weight constraint, from the Value table computed by this code.

From the table you can find the highest valued subset of objects by examining the table starting at $Value[k, S]$, where k = number of objects considered and S = the maximum weight. If $Value[k-1, S] = Value[k, S]$, then do not add object k to the solution set and let $k = k - 1$. Otherwise, add object k to the solution set and let $S = S - w_k$, where w_k = the weight of object k , then let $k = k - 1$. Repeat until $k = 0$.

- (b) Explain how to solve the Knapsack problem using only $O(L)$ memory/space and $O(nL)$ time. You need only find the value and weight of the optimal solution, not the actual collection of objects.

In the iterative solution when building the table, the writes for column k depend only on the values stored in column $k - 1$. Therefore, our array only needs to have two columns. At each iteration (when k increases), the columns alternate between being the read column and the write column. This eliminates copy time, although this isn't necessary. The alternating of read-column and write-column is a method taken from the idea of double-buffering.

From 12b, we take the resulting array and then iterate as follows to determine the set of items:

$\ell = 0, v_a = 0, w_a = 0, v_b = 0, w_b = 0$

For $i = 1$ to L :

if $\exists k = \text{Array}[i] - \ell$:

add item k to the solution set S , and $\ell = \text{Array}[i]$

$v_a = v_a + v_k$

$w_a = w_a + w_k$

if $k \leq n/2$:

$v_b = v_b + v_k$

$w_b = w_b + w_k$

- (b) Explain how to use the algorithm from the previous subproblem to get a divide and conquer algorithm for finding the items in the Knapsack problem and uses space $O(L)$ and time $O(nL)$.

We can partition the space for the knapsack problem by considering each of the items alone, and then merging those solutions together in pairs and then continue merging until we have one full set. The process of combining the solution sets uses the process mentioned above as determining which items make up the solution set.

16. Give an algorithm for the following problem whose running time is polynomial in $n + W$:

Input: positive integers $w_1, \dots, w_n, v_1, \dots, v_n$ and W .

Output: The maximum possible value of $\sum_{i=1}^n x_i v_i$ subject to $\sum_{i=1}^n x_i w_i \leq W$ and each x_i is a nonnegative integer.

Pruning Rules:

- 1) At each level, there may be no more than W configurations. Ignore nodes where the cumulative sum is greater than W .
- 2) At each level, multiple children are created. Only create W/w_k children, since more would have a weight greater than W .
- 3) If \exists two or more nodes on the same level with the same weight, prune all but the one with the most value.

Begin with root node $A[k = 0, S = 0, x = 0]$, where k is the *level* of the tree, for 0 to n , and S is the cumulative weight, for 0 to W , and x is the multiplier.

For $k = 0$ to $n - 1$:

For $S = 0$ to W :

For $x = 0$ to W/w_k :

If $A[k, A[k, S, x], x] + v_k \cdot x > \forall V \in A[k, A[k, S, x]]$:

$A[k + 1, A[k, S, x] + x \cdot w_k, x] = A[k, A[k, S, x], x] + v_k \cdot x$.

17. Give an algorithm for the following problem whose running time is polynomial in $n + L$, where $L = \max (\sum_{i=1}^n v_i^3, \prod_{i=1}^n v_i)$.

Input: positive integers v_1, \dots, v_n

Output: A subset S of the integers such that $\sum_{v_i \in S} v_i^3 = \prod_{v_i \in S} v_i$.