The Knapsack Problem.

We are given a set of items $\{1, 2, \ldots, n\}$. Each item has an integer <u>weight</u> $w_i > 0$ and an integer <u>value</u> $v_i > 0$. We also have a Knapsack whose capacity is a weight $W > 0$. The goal is to find the most valuable subset of items among all subsets ~~whose~~ with weight at most W. (Only such subsets fit into the Knapsack.)

That is, we want ~~among~~ to maximize $\sum_{i \in S} v_i$ over all subsets S such that $\sum_{i \in S} w_i \leq W$.

As usual, let us begin by analyzing an optimal solution O to the problem.

Either $n \in O$, or $n \notin O$.

(1) If $n \notin O$, O must be an optimal solution for the set of items $\{1, 2, .., n-1\}$ and Knapsack of capacity W.

(2) If $n \in O$, $O \setminus \{n\}$ must be an optimal solution for the set of items $\{1, 2, \ldots, n-1\}$ and Knapsack of capacity $W - w_n$.

Note that in both cases, we are interested in the subset $\{1, 2, \ldots, n-1\}$. It is the Knapsack capacity that is different in the two cases.

Let $Opt(i, \bar{w})$ denote the value of the optimal solution for items $\{1, 2, \ldots i\}$ and Knapsack of capacity $\bar{w}$.

We define $Opt(0, \bar{w})$ to be 0 for any $\bar{w} \geq 0$. We want $Opt(n, W)$

We have the following:

$$Opt(n, \bar{w}) = Opt(n-1, \bar{w}) \quad \text{if } w_n > \bar{w}.$$

$$Opt(n, \bar{w}) = \max\{ Opt(n-1, \bar{w}), \; v_n + Opt(n-1, \bar{w} - w_n) \} \quad \text{otherwise.}$$

In general, for any $1 \le i \le n$,

$$Opt(i, \bar{w}) = Opt(i-1, \bar{w}) \quad \text{if } w_i > \bar{w}.$$

$$Opt(i, \bar{w}) = \max\{ Opt(i-1, \bar{w}),\ v_i + Opt(i-1, \bar{w} - w_i) \} \quad \text{otherwise.}$$

Read this as: If we know $Opt(i-1, \bar{w})$ and $opt(i-1, \bar{w} - w_i)$ we can compute $opt(i, \bar{w})$ immediately.

Let us define an $(n+1) \times (W+1)$ array $M[\ ,\ ]$:

$M[i, \bar{w}]$ will hold $opt(i, \bar{w})$

For $\bar{w} \leftarrow 0$ to W do

Set $M[0, \bar{w}] \leftarrow 0$

For $\bar{w} \leftarrow 0$ to W do

For $i \leftarrow 1$ to n do

If $w_i > \bar{w}$

$M[i, \bar{w}] \leftarrow M[i-1, \bar{w}]$

else

$M[i, \bar{w}] \leftarrow \max\{ M[i-1, \bar{w}],$

$v_i + M[i-1, \bar{w} - w_i] \}$

See Fig 2. Fig 1 below is for algo on next page.

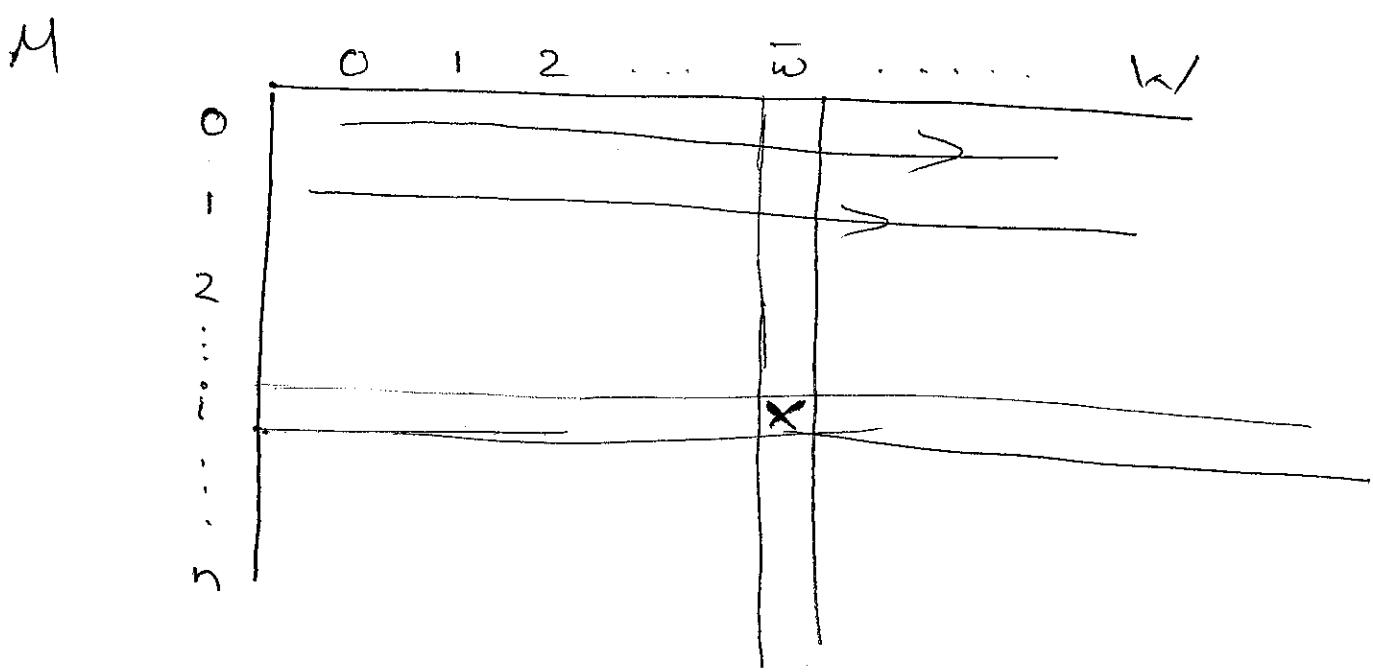

Fig 1

The following also works:

For $\bar{w} \leftarrow 0$ to W do

  Set $M[0, \bar{w}] \leftarrow 0$

For $i \leftarrow 1$ to n do

  For $\bar{w} \leftarrow 0$ to W do

    If $w_i > \bar{w}$

      $M[i, \bar{w}] \leftarrow M[i-1, \bar{w}]$

    else

      $M[i, \bar{w}] \leftarrow \max\{M[i-1, \bar{w}],$

      $v_i + M[i-1, \bar{w} - w_i]\}$

M

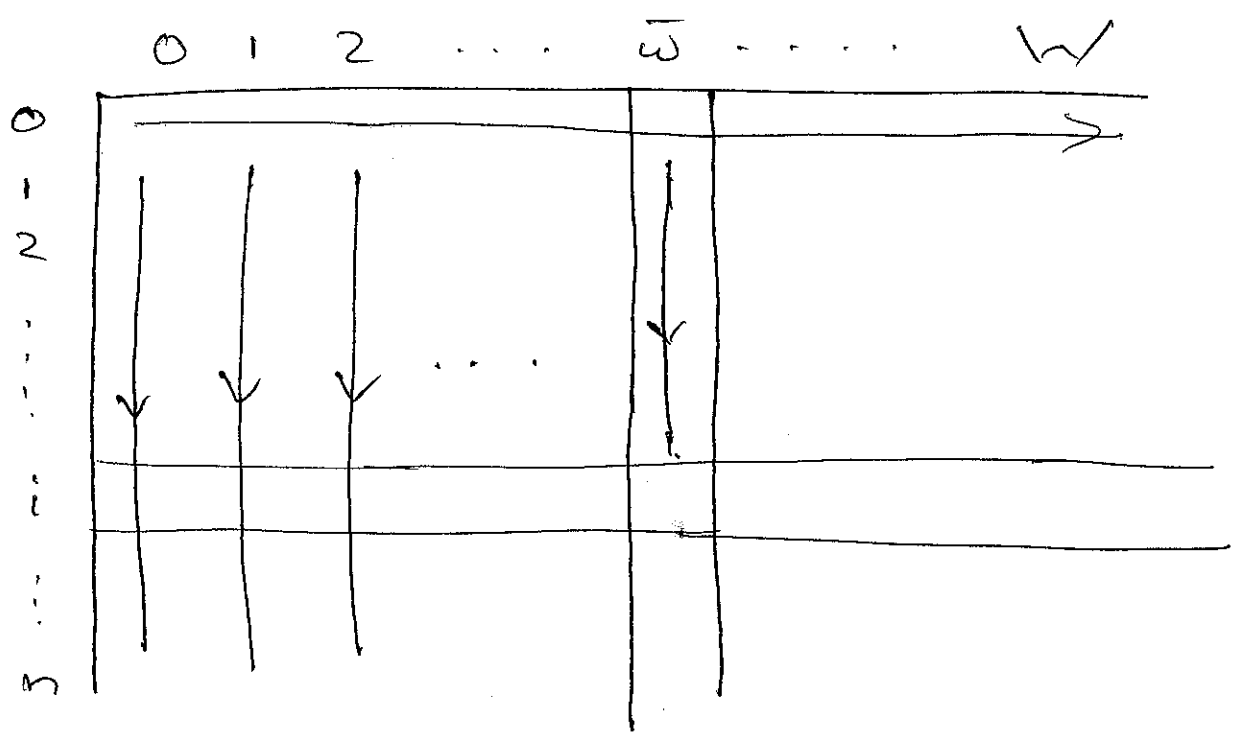

Fig 2

Notice that running time is $O(n\,W)$.

Running time depends not only on n but on capacity of Knapsack. This is unlike the earlier examples.

Is there an algorithm with running time polynomial in n and independent of W? As we'll see later on in the course, there are good reasons to think that this is unlikely.

Let us return to the problem:

Given an array $A[1..n]$ consisting of integers (positive or negative), find the maximum sum of any contiguous subarray.

Example:

31 -41 59 26 -53 58 97 -93 -23 84

Solution:

$59 + 26 + (-53) + 58 + 97$

Let us define $Opt(i)$ as sum of best subarray that ends at $A[i]$.

We want

$\max\{Opt(1), Opt(2), \ldots, Opt(n)\}$.

We know $Opt(1)$ - it is $A[1]$.

~~We~~ ~~to~~ What about $Opt(i)$ for $i > 1$?

We can argue that

$$\boxed{Opt(i) = \max\{A[i],\ Opt(i-1) + A[i]\}}$$

(Argument in class.)

So here is our algorithm.

$M[1] \leftarrow A[1]$

For $i \leftarrow 2$ to n do

~~If Opt(i-1) > 0 then~~

If $M[i-1] > 0$ then

$M[i] \leftarrow M[i-1] + A[i]$

else

$M[i] \leftarrow A[i]$

Thats an $O(n)$ algorithm.