Goals
- Create good loop invariants
- Use loop invariants so prove DP correctness
- Describe encoding problem

Loop Invariants ... Tips for Success

[See Bubble Sort Pseudocode on next page]

1. Invariant should involve iterated variable.
   - Example: Inner loop of BS $\Rightarrow$ for $j=n$ to $k+1$
     "j" is iterated variable
     - Loop invariant: $A[j]$ is min $\{A[j:n]\}$

2. Invariant generally doesn't involve things outside domain of loop
   - Inner loop $\Rightarrow$ $j=n$ to $k+1$
   - $A[k:j]$ is sorted $\times$ wrong: $j$ hasn't gotten to this part of the array, so we can't be maintaining anything here.
3. To figure out loop invariant, freeze in middle of loop

This is the part of array we can talk about

\[
\begin{array}{cccccc}
1 & 2 & 3 & 6 & 10 & 14 & 11 & 12 & 15 & 14 & 17 \\
1 & k+1 & j & n
\end{array}
\]

\[A[j] \text{ is min of } A[j:n]\]

Later; when termination requires A's elements are same, add that condition
Input: Array $A$ of integers of length $n$

Output: Array containing sorted elements of $A$

1 for $k = 1$ to $n - 1$ do
2    for $j = n$ to $k + 1$ do
4            Swap $A[j]$ and $A[j - 1]$;
5        end
6    end
7 end
8 return $A$;

Algorithm 2: BubbleSort($A$)
Knapsack Problem

Input:  
- $n$ items, each has 
  - value $v_j$  \( \in \mathbb{Z}^+ \) 
  - size $w_j$
- Capacity $W$

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i = V(S)$
  and satisfies $\sum_{i \in S} w_i \leq W$

Constraints: 

Objective function

$K_{i,r}$ = optimal subset if only $i^{th}$ $i$ items are allowed, capacity $r$.

D.P. Pseudocode

// create array $V$ of objective function values of $K_{i,r}$
for $r = 0$ to $W$:
    $V[0, r] = 0$

for $i = 1$ to $n$:
    for $r = 0$ to $W$:
        $V[i, r] = \max \{ V[i-1, r], V[i-1, r - w_i] + v_i \}$

This option:
Not allowed if $r - w_i < 0$

Knapsack Page 1
// pseudocode to get S using array V

S = ∅
i = n
r = W

while i > 0:
    if V[i, r] = V[i-1, r-w_i] + v_i
        S = S + i
        r = r - w_i
    i -=

---

Group Work

- Loop invariants for 2 loops
- Initialization / Termination
- Maintenance

* Put proofs from step 2 in Maintenance of first loop

---

S. Kimmel
Proof of Correctness

Loop 1 Invariant: \( V[j, g] = V[j, g] \) for all elements of \( V \) up to \( V[i, r] \)

- Initialization: \( i=1, r=0 \). \( V[0, r] = V_0, r \) for all \( r \).  
- Maintenance: We will prove the loop sets \( V[i, r] \) to have the correct value. Let \( S \) be optimal solution to \( K_i, r \). Then either \( i \notin S \) or \( i \in S \).
  
  If \( i \notin S \), then \( S \) is optimal soln to \( K_{i-1}, r \).

For contradiction, suppose \( S \) is not opt soln for \( K_{i-1}, r \).

Then there exists an optimal solution \( S' \) for \( K_{i-1}, r \) with \( V(s') > V(s) \). But \( S' \) is also a solution to \( K_i, r \) with \( V(s') > V(s) \) and \( i \notin S' \), so \( S \) is not the optimal solution to \( K_i, r \), a contradiction.

Similarly, if \( i \in S \), \( S - \{i\} \) is an optimal solution to \( K_{i-1}, r - w_i \). Since the optimal
Solution is one of these two options, we take the one with the larger objective value. So

\[ V_{i,r} = \max \left\{ V_{i-1,r}, V_{i-1,r-w_i} + v_i \right\} \]

\[ \uparrow \]

if \( r - w_i > 0 \).

By our loop invariant, this is

\[ V_{i,r} = \max \left\{ V[i-1,r], V[i-1,r-w_i] + v_i \right\} \]

\[ \uparrow \]

if \( r - w_i > 0 \)

which is what our algorithm sets \( V[i,r] \) to be.

Termination: \( V[i,r] = V_i,r \) for all \( 1 \leq i \leq n, 1 \leq i \leq W \).
Pf that loop is correct. Let S be opt. solution. Can assume $V[i, w]$ is value of opt. sol. on $K_i, w$

Loop Invariant:  · $S'$ contains all elements of $S$
   · $r = W - W(s')$

Initially: $i = n$, $S = \emptyset$, $r = W$

Maintenance: Suppose loop invariant is true going into loop.
Then $S'$ contains elements of $S$ larger than $i$,
so we just need to figure out those less than or equal to $i$. Since we've already used up capacity $W(s')$, we need to solve $K_i, w - W(s')$
to figure out remaining elements of $S'$.
We now determine if $i \in$ opt. soln for $K_i, w - W(s')$
Only two options (i), or (ii) and at least
one is true, so either $V[i, W - W(s')] = V[i - 1, W - W(s')]$
or $V[i, W - W(s')] = V[i - 1, W - W(s') - w_i] + v_i$, and which
one it is tells us whether $i$ is in optimal solution. In either case, invariant is maintained.


**Huffman Codes**

Binary code: each letter of alphabet $\Sigma \rightarrow$ binary string

e.g. $\Sigma = \{a, b, \ldots, z, \ldots, \} \rightarrow$ each letter gets unique string from $\{0,1\}^*$

32 letters

$\rightarrow$ ASCII


e.g. $\Sigma = \{a, b, c\}$

Suppose you have a message where a occurs 50%, b occurs 30%, c occurs 20%.

Q. What is the best and most efficient binary encoding of $a, b, c$ to send message?

A) $a \rightarrow 00$, $b \rightarrow 01$, $c \rightarrow 10$
B) $a \rightarrow 0$, $b \rightarrow 1$, $c \rightarrow 01$
C) $a = 0 \rightarrow 01$, $b = 11 \rightarrow 10$, $c = 01 \rightarrow 11$
D) $a = 0 \rightarrow 0$, $b = 10 \rightarrow 10$, $c = 11 \rightarrow 11$
Huffman Codes

Binary code: each letter of alphabet $\Sigma \rightarrow$ binary string
- e.g. $\Sigma = \{a, b, \ldots, z, \ldots, ?\}$ → each letter gets unique string from $\{0,1\}^*$
  - 32 letters
  - ASCII

- e.g. $\Sigma = \{a, b, c\}$

Suppose you have a message where $a$ occurs 50%, $b$ occurs 30%, $c$ occurs 20%.

Q. What is the best and most efficient binary encoding of $a$, $b$, $c$ to send message?

A) $a \rightarrow 00$
   $b \rightarrow 01$
   $c \rightarrow 10$
   Average length: 2

B) $a \rightarrow 0$
   $b \rightarrow 1$
   $c \rightarrow 01$

C) $a = 0$
   $b = 11$
   $c = 01$

D) $a = 0$
   $b = 10$
   $c = 11$

$0110 = abba$
or
$cbba$

$0110$ or beginning of $c$
- start of $b$, end of $c$
- start of $b$, end of $b$
- Need to go to next to figure out

average length

$0.5 \times 1 + 0.5 \times 2 = 1.5$