HW 1 Solutions

1. Section 1.1
a) How can we build a computer? b) How can we program a computer?

2. Section 1.2
Fixed-program devices are designed around and can execute only a single algorithm, whose implementation is hard-wired into the device. Stored-program devices have users store the sequence of operations they want performed on the input.

3. Section 1.3
Abstraction is the division of a system into separate subsystems, and we understand the system as a whole by thinking only about how these subsystems interact with each other. Abstraction simplifies the system because we will only need to worry about 1 subsystem at a time.

4. Section 1.4.1 and 1.4.2
a) false
b) true

5. Section 1.4.3.1
An algorithm is a series of steps to achieve a goal. An algorithm must 1) be precise and 2) terminate.

Alternatively:
An algorithm is a procedure to solve a problem. An algorithm must satisfy the finiteness (must terminate) and definiteness (must be precise) property.

6. Section 1.4.3.1 helps
Many solutions are accepted

Sample solution 1:
1) Create a counter variable n=0
2) Initialize C at 0
3) Add 1 to n
4) Add A to the value of C
5) If n < B, return to step 3. Else terminate.

Step 1: **n=0;** A=2; B=3; C=X
Step 2: n=0; A=2; B=3; **C=0**
Step 3: **n=1;** A=2; B=3; C=0
Step 4: n=1; A=2; B=3; **C=2**
Step 5: return to step 3
Step 3: **n=2;** A=2; B=3; C=2
Step 4: n=2; A=2; B=3; **C=4**
Step 5: return to step 3
Step 3: **n=3;** A=2; B=3; C=2
Step 4: n=2; A=2; B=3; **C=6**
Step 5: terminate

Sample solution 2:
1) Set C=0
2) Add -1 to B and store the sum in B
3) Add A to C and store the sum in C
4) If B > 0, return to step 2. Else terminate.

Step 1: A=2; B=3; **C=0**
Step 2: A=2; **B=2;** C=0
Step 3: A=2; B=2; **C=2**
Step 4: return to step 2
Step 2: A=2; **B=1;** C=2
Step 3: A=2; B=1; **C=4**
Step 4: return to step 2
Step 2: A=2; **B=0;** C=4
Step 3: A=2; B=0; **C=6**
Step 4: terminate

Sample solution 3:

1. **Store 1 in X**
2. **Store A in C**
3. **IF X == B goto 7**
4. **Add A to C, store in C**
5. **Add 1 to X, store in X**
6. **goto 3**
7. **return**

In demonstration:

a. **1. Store 1 in x. X = 1 A = 2 B = 3 C = NA**
b. **2. Store A in C X = 1 A = 2 B = 3 C = 2**
c. **3. If X == B goto 7 X = 1 A = 2 B = 3 C = 2**

d. **4. Add A to C X = 1 A = 2 B = 3 C = 4**
e. **5. Add 1 to X X = 2 A = 2 B = 3 C = 4**
f. **6. goto 3 X = 2 A = 2 B = 3 C = 4**
g. **3. If X == B goto 7 X = 2 A = 2 B = 3 C = 4**
h. **4. Add A to C X = 2 A = 2 B = 3 C = 6**
i. **5. Add 1 to X X = 3 A = 2 B = 3 C = 6**
j. **6. goto 3 X = 3 A = 2 B = 3 C = 6**
k. **3. IF X == B goto 7 X = 3 A = 2 B = 3 C = 6**
l. **7. return**

7. Section 1.4.3.1
Algorithm complexity is the amount of resources - either through the amount of steps, memory, power, or otherwise - it takes to execute an algorithm. Trade-offs are made when we cannot have the best of all worlds and must exchange resources. For example, one algorithm might be slower but more energy efficient than another algorithm.

8. Section 1.4.3.2
Less overhead, more flexibility and control (over memory locations), and possibly better performance.

9. Section 1.4.3.3
a) correct
b) syntax error
c) logic error (runtime error is also accepted due to older editions of the book)
d) syntax error

10. Section 1.6
A Turing machine is said to be Turing complete if it is capable of simulating any other Turing machine (and therefore of running any algorithm, if you believe Church's Thesis).