Solution Manual for Invitation to Computer Science 7th Edition Schneider 1305075773 9781305075771

Solutions to End-of-Chapter Exercises

Chapter 1: An Introduction to Computer Science

1. There is no one correct answer. Common examples are the instructions for using a voice mail system, the instructions for opening a mail box lock, and the instructions for doing laundry.

2. A *heuristic* is a method for finding a reasonably close, "good enough" solution to a problem. It can be viewed as a rule-of-thumb, a method of approximation, an informal technique, or even a way to make an "educated guess." It differs from the concept of an algorithm in that it does not guarantee to produce an optimal solution, just to make a good faith attempt to locate a reasonable one. Heuristics are often used when executing an algorithm might be too time-consuming, and we only need an approximation to the correct answer.

An example of a heuristic for adding two 3-digit numbers, such as 234 + 567, might be:

1. Set the one and tens digit of both operands to 0
2. Increase the hundreds digit of the second operand by 1. These two steps result in changing the problem to the simpler one 200 + 600.
3. Add the hundreds digits, resulting in a final "answer" of 800.

Now, of course, this is not the correct answer, which is 801. But the result we get may be close enough for our needs, and it is certainly a lot easier to add a single column of numbers rather than three columns of numbers.

3. One may argue that the instruction is not well-ordered, since it is unclear whether one should enter the channel first or press CHAN first. Also, it may not be effectively computable if you desire to enter a channel that is out of the DVR's range.

4. (a) Sequential
(b) Conditional
(c) Sequential
(d) Iterative

5. Step 1: *carry* = 0, c_3 = ??, c_2 = ??, c_1 = ??, and c_0 = ??

Step 2: $i = 0$, all others unchanged

Step 4: c_0 = 18, all others unchanged

Step 5: c_0 = 8 and *carry* = 1, all others unchanged

Step 6: i = 1, *carry* = 1, c_3 = ??, c_2 = ??, c_1 = ??, and c_0 = 8

Step 4: c_1 = 7, all others unchanged

Step 5: *carry* = 0, all others unchanged

Step 6: i = 2, *carry* = 0, c_3 = ??, c_2 = ??, c_1 = 7, and c_0 = 8

Step 4: $c_1 = 1$, all others unchanged

Step 5: *carry* = 0, all others unchanged

Step 6: $i = 3$, *carry* = 0, c_3 = ??, $c_2 = 1$, $c_1 = 7$, and $c_0 = 8$

Step 7: $c_3 = 0$, $c_2 = 1$, $c_1 = 7$, and $c_0 = 8$

Step 8: Print out 0178.

6. Replace Step 8 with the following steps:

Step 8: Set the value of *i* to *m*

Step 9: Repeat step 10 until either c_i is not equal to 0 or $i < 0$

Step 10:Subtract 1 from *i*, moving one digit to the right

Step 11: If $i \geq 0$ then print $c_i c_{i-1} \ldots c_0$

7. Assume that *a* has *n* digits $a_{n-1}, \ldots, a_0$, and *b* has *m* digits, $b_{m-1}, \ldots, b_0$, with *n* not necessarily equal to *m*. Add an operation at the beginning of the algorithm that resets the two numbers to the same number of digits by adding non-significant leading zeros to the shorter one. We can then reuse the algorithm of Figure 1.2.

```
If (m > n) then
        Set i to 0
        While (n+i < m)
                Add a leading zero to the number at position a_{n+i}
                Increment i by 1
        End of the loop
Else
        If (n > m)
                Set i to 0
                While (m+i < n)
                        Add a leading zero to the number at position b_{m+i}
                        Increment i by 1
                End of the loop
```

We have now made the two numbers equal in length. All we need do now is set the variable *m* to the larger of the two values:

Set m to the larger of m and n.

The addition algorithm in Figure 1.2 will now work correctly. Note that if m and n are equal in value, neither of the Boolean expressions will be true, and neither of the conditional statements will be executed.

8. It is not effectively computable if $b^2 - 4ac < 0$ (since we cannot take the square root of a negative number if we are limited to real numbers) or if $a = 0$ (since we cannot divide by 0).

9. The first algorithm (Figure 1.3(a)) is a better general purpose algorithm. If you want to shampoo your hair any number *n* times you can change the 2 to *n*. You could even ask the shampooer to input the desired number *n* of washings. For the second algorithm you would have to rewrite the algorithm to repeat steps 4 and 5 998 more times.

10. (a) Trace:

Step 1: $I = 32$, $J = 20$, and R = ??

Step 2: $I = 32$, $J = 20$, and $R = 12$

Step 3: $I = 20$, $J = 12$, and $R = 12$

Step 2: $I = 20$, $J = 12$, and $R = 8$

Step 3: $I = 12$, $J = 8$, and $R = 8$

Step 2: $I = 12$, $J = 8$, and $R = 4$

Step 3: $I = 8$, $J = 4$, and $R = 4$

Step 2: $I = 8$, $J = 4$, and $R = 0$

Step 4: Print $J = 4$

(b) At Step 2 we are asked to divide $I = 32$ by $J = 0$, which cannot be done. We can fix the problem by adding a step between Step 1 and Step 2 that says: If $J = 0$, then print "ERROR: division by 0" and Stop.

11. There are 25! possible paths to be considered. That is approximately 1.5×10^{25} different paths. The computer can analyze 10,000,000, or 10^7, paths per second. The number of seconds required to check all possible paths is about $1.5 \times 10^{25}/10^7$, or about 1.5×10^{18} seconds. That's roughly 10^{12} years: about a trillion years. This would not be a feasible algorithm.

12. A Multiplication Algorithm.

Given: Two positive numbers *a* and *b*

Wanted: A number *c* which contains the result of multiplying *a* and *b*

Step 1: Set the value of *c* equal to 0

Step 2: Set the value of *i* equal to *b*

Step 3: Repeat steps 4 and 5 until the value of *i* is 0

Step 4: Set the value of c to be $c + a$

Step 5: Subtract 1 from i

Step 6: Print out the final answer c

Step 7: Stop

This algorithm assumes that we know how to add two multiple-digit numbers together. We may assume this because we have the algorithm from the book which does exactly that.

13. The algorithm will work correctly only if all three numbers are unique. If two or more numbers are identical, none of the Boolean expressions will be true and nothing will be output. To make this a correct solution you either have to specify in the problem statement that the three numbers provided must all be distinct or (better) change all of the comparison operations to ≥ in place of >.

14. This is an essay question. Students may find excellent resources on the Internet.

15. If this problem is assigned, be sure to coordinate with your computing staff ahead of time for students to get the required information.

16. This is an essay question. Because this is a "hot" topic, a great deal of hype and hyperbole is available, as well as useful information. It might be a good opportunity to teach students about finding *reliable* sources on the Internet, and evaluating online and print source materials.

17. Like question 16 this is an essay question. Students may be familiar with Apple iCloud services for iPhone and iPad devices, so it might be a good opportunity to relate their answers to the services provided by Apple.

18. About 130 feet ((((700,000,000 chars/5 chars per word)/300 words per page)/300 pages per inch)/12 inch per foot)

Discussion of Challenge Work

1. We may perform subtraction, like addition, by subtracting one column at a time, starting with the rightmost column and working to the left. Since we know that the first number is larger than the second one, we know that we can always borrow from columns to the left of the current one. Therefore, if the upper number in the column (a_i) is smaller than the lower, we automatically borrow from the next column. We can do this by subtracting one from the a_{i+1} value of the column to the left. If the a_{i+1} value were already zero, then it would become -1. This automatically causes a borrow to occur on the next step. Here is the algorithm:

Step 1: Set the value of i equal to the value of 0

Step 2: Repeat steps 3 to 6 until the value of i is m

Step 3: If $b_i \leq a_i$ then

Step 4: Set c_i equal to $a_i - b_i$

Otherwise ($b_i > a_i$)

Step 5: Set c_i equal to $(a_i + 10) - b_i$ and replace a_{i+1} with $a_{i+1} - 1$

(This amounts to a borrow of 1 from a_{i+1} which adds 10 to a_i)

Step 6: Add 1 to i (moving us one column to the left)

2. Students may need assistance finding or understanding other definitions from other sources.

Chapter 2: Algorithm Discovery and Design

1. (a) Set the value of *area* to ½(*b*   *h*)
(b) Set the value of *interest* to *I*   *B*
Set the value of *FinalBalance* to (1 + *I*)   *B*
(c) Set the value of *FlyingTime* to *M/AvgSpeed*

2. Algorithm:

Step 1: Get values for *B*, *I*, and *S*

Step 2: Set the value of *FinalBalance* to $(1 + I/12)^{12}B$

Step 3: Set the value of *Interest* to *FinalBalance* – *B*

Step 4: Set the value of *FinalBalance* to *FinalBalance* – *S*

Step 5: Print the message 'Interest Earned: '

Step 6: Print the value of *Interest*

Step 7: Print the message 'Final Balance: '

Step 8: Print the value of *FinalBalance*

3. Algorithm:

Step 1: Get values for *E1*, *E2*, *E3 and F*

Step 2: Set the value of *Ave* to (*E1* + *E2* + *E3* + *2F*)/5

Step 3: Print the value of *Ave*

4. Algorithm:

Step 1: Get values for *P* and *Q*

Step 2: Set the value of *Subtotal* to *P*   *Q*

Step 3: Set the value of *TotalCost* to (1.06)   *Subtotal*

Step 4: Print the value of *TotalCost*

5. (a) If y 0 then

(b) If $r \geq 1.0$, then
Set the value of Area to r^2
Set the value of *Circum* to 2 r

6. Algorithm:

Step 1: Get a value for *B*, *I*, and *S*

Step 2: Set the value of *FinalBalance* to $(1 + I/12)^{12}B$

Step 3: Set the value of *Interest* to *FinalBalance* – *B*

Step 4: If *B* < 1000 then Set the value of *FinalBalance* to *FinalBalance* – *S*

Step 5: Print the message 'Interest Earned: '

Step 6: Print the value of *Interest*

Step 7: Print the message 'Final Balance: '

Step 8: Print the value of *FinalBalance*

7. Algorithm:

Step 1: Set the value of *i* to 1

Step 2: Set the values of *Won*, *Lost*, and *Tied* all to 0

Step 3: While $i \leq 10$ do

Step 4: Get the value of CSU_i and OPP_i

Step 5: If $CSU_i > OPP_i$ then

Step 6: Set the value of *Won* to *Won* + 1

Step 7: Else if $CSU_i < OPP_i$ then

Step 8: Set the value of *Lost* to *Lost* + 1

Step 9: Else

Step 10: Set the value of *Tied* to *Tied* + 1

Step 11: Set the value of *i* to *i* + 1

Step 14: Print the message, 'Congratulations on your undefeated season.'

8. Algorithm:

Step 1: Set the value of *i* to 1

Step 2: Set the value of *Total* to 0

Step 3: While $i \leq 14$ do

Step 4: Get the value of E_i

Step 5: Set the value of *Total* to *Total* + E_i

Step 6: Set the value of *i* to *i* + 1

End of While loop

Step 7: Get the value of *F*

Step 8: Set the value of *Total* to *Total* + 2*F*

Step 9: Set the value of *Ave* to *Total* / 16

Step 10: Print the value of *Ave*

9. Algorithm:

Step 1: Set the value of *TotalCost* to 0

Step 2: Do

Step 3: Get values for *P* and *Q*

Step 4: Set the value of *Subtotal* to *P* *Q*

Step 5: Set the value of *TotalCost* to *TotalCost* + (1.06) *Subtotal*

While (*TotalCost* < 1000)

Step 6: Print the value of *TotalCost*

10. The tricky part is in steps 6 through 9. If you use no more than 1000 kilowatt hours in the month then you get charged $.06 for each. If you use more than 1000, then you get charged $.06 for the first 1000 hours and $.08 for each of the remaining hours. There are M_i – 1000 remaining hours, since M_i is the number of hours in the *i*th month. Also, remember that $KWBegin_i$ and $KWEnd_i$ are meter readings, so we can determine the total kilowatt-hours used for the whole year by subtracting the first meter reading ($KWBegin_1$) from the last ($KWEnd_{12}$).

Step 1: Set the value of *i* to 1

Step 2: Set the value of *AnnualCharge* to 0

Step 3: While $i \leq 12$ do

Step 4: Get the value of $KWBegin_i$ and $KWEnd_i$

Step 5: Set the value of M_i to $KWEnd_i – KWBegin_i$

Step 6: If $M_i \leq 1000$ then

Step 7: Set *AnnualCharge* to *AnnualCharge* + (.06M_i)

Step 8: Else

Step 9: Set *AnnualCharge* to *AnnualCharge* + (.06)1000

+ (.08)(M_i – 1000)

Step 10: Set the value of *i* to *i* + 1

End of While loop

Step 11: Print the value of *AnnualCharge*

Step 12: If ($KWEnd_{12} – KWBegin_1$) < 500, then

Step 13: Print the message 'Thank you for conserving electricity.'

11. Algorithm:

Step 1: Do

Step 2:Get the values of *HoursWorked* and *PayRate*

Step 3:If *HoursWorked* > 54 then

Step 4: $DT = HoursWorked - 54$

Step 5: TH=14

Step 6: Reg = 40

Step 7: Else if *HoursWorked* > 40 then

Step 8: DT=0

Step 9: $TH = HoursWorked - 40$

Step 10: Reg = 40

Step 11: Else (*HoursWorked* $\leq$ 40)

Step 12: DT=0

Step 13: TH=0

Step 14: $Reg = HoursWorked$

Step 15: *GrossPay* = *PayRate*   *Reg*

+ 1.5   *PayRate*   *TH* + 2   *PayRate*   *DT*

Step 16: Print the value of *GrossPay*

Step 17: Print the message 'Do you wish to do another computation?'

Step 18: Get the value of *Again*

While (*Again* = yes)

12. Steps 1, 2, 5, 6, 7, and 9 are sequential operations and steps 4 and 8 are conditional operations. After their completion, the algorithm moves on to the step below it, so none of these could cause an infinite loop. Step 3, however, is a while loop, and it could possibly cause an infinite loop. The true/false looping condition is “*Found* = NO and *i* 10,000.” If NUMBER is ever found in the loop then *Found* gets set to YES, the loop stops, and the algorithm ends after executing steps 8 and 9. If NUMBER is never found, then 1 is added to *i* at each iteration of the loop. Since step 2 initializes *i* to 1, *i* will become 10,001 after the 10,000th iteration of the loop. At this point the loop will halt, steps 8 and 9 will be executed, and the algorithm will end.

13. Algorithm:

Step 1: Get values for *NUMBER*, T_1, ... T_{10000}, and N_1, ..., N_{10000}

Step 2: Set the value of *i* to 1 and set the value of *NumberFound* to 0

Step 3: While (*i*   10,000) do steps 4 through 7

Step 4: If *NUMBER* equals T_i then

Step 5: Print the name of the corresponding person, N_i

Step 6: Set the value of *NumberFound* to *NumberFound* + 1

Step 7: Add 1 to the value of *i*

Step 8: Print the message *NUMBER* ' was found ' *NumberFound* 'times'

Step 9: Stop

14. Let's assume that FindLargest is now a primitive to us, and use it to repeatedly remove the largest element from the list until we reach the median.

Let $M = N / 2$

Else

Let $M = (N + 1) / 2$

Step 3: While (N   M) do steps 4 through 9

Step 4: Use FindLargest to find the *location* of the largest number

in the list $L_1, L_2, \ldots, L_N$

Step 5: Exchange $L_{location}$ and L_N as follows

Step 6: $Temp = L_N$

Step 7: $L_N = L_{location}$

Step 8: $L_{location} = Temp$

Step 9: Set N to $N - 1$ and effectively shorten the list

Step 10: Print the message 'The median is: '

Step 11: Print the value of L_M

Step 12: Stop

15. This algorithm will find the first occurrence of the largest element in the collection. This element will become *LargestSoFar*, and from then on A_i will be tested to see if it is greater than *LargestSoFar*. Some of the other elements are equal to *LargestSoFar* but none are greater than it.

16. (a) If $n \leq 2$, then the test would be true, so the loop would be executed. In fact, the test would never become false. Thus the algorithm would either loop forever, or generate an error when referring to an invalid A_i value. If $n > 2$, then the test would be false the first time through, so the loop would be skipped and A_1 would be reported as the largest value.

(b) The algorithm would find the largest of the first $n - 1$ elements and would not look at the last element, as the loop would exit when $i = n$.

(c) For $n = 2$ the loop would execute once, comparing the A_1 and A_2 values. Then the loop would quit on the next pass, returning the larger of the first two values. For any other value of *n*, the loop would be skipped, reporting A_1 as the largest value.

17. (a) The algorithm would still find the largest element in the list, but if the largest were not unique then the algorithm would find the last occurrence of the largest element in the list.

(b) The algorithm would find the smallest element in the list.

The relational operations are very important, and care must be taken to choose the correct one, for mixing them up can drastically change the output of the algorithm.

18. (a) The algorithm will find the three occurrences of "and". First in the word band, second in the word and, and third in the word handle.

(b) We could search for “ and ”. That is, the word itself surrounded by spaces. Note that the word "and" is special in that it is almost always surrounded by spaces in a sentence. Other words may start or end sentences and be followed by punctuation.

19. It would go into an infinite loop, because *k* will stay at 1, and we will never leave the outside while loop. We will keep checking the 1 position over and over again.

20. Step 1: Get the value for *N*

Step 2: Set the value of *i* to 2

Step 3: Set the value of *R* to 1;

Step 4: While ($i < N$ and *R*   0) do Steps 5-6

Step 5:Set *R* to the remainder upon computing N/i

Step 6:Set the value of *i* to $i + 1$

Step 7: If $R = 0$ then

Print the message 'not prime'

Else

Print the message 'prime'

(This algorithm could be improved upon because it is enough to look for divisors of N less than or equal to $\sqrt{N}$.)

21. Here we assume that we can perform "arithmetic" on characters, so that m + 3 = p, for example. Step 4 is the difficult part that must handle the "wraparound" from the end of the alphabet back to the beginning.

Step 3: Set the value of *outChar* to *nextChar* + *k*

Step 4: If *outChar* > z then

Set the value of *outChar* to (*outChar* -26)

Step 5: Print *outChar*

22. Step 1: Get the values for *k* and N_1, N_2, …, N_k

Step 2: Set the value of *front* to 1

Step 3: Set the value of *back* to *k*

Step 4: While (*front*   *back*) do steps 5 through 9

Step 5:Set the value of *Temp* to N_{back}

Step 6:Set the value of N_{back} to N_{front}

Step 7: Set the value of N_{front} to *Temp*

Step 8: Set *front* = *front* + 1

Step 9 Set *back* = *back* – 1

23. Step 1: Get the values for N_1, N_2, …, N_k, and *SUM*

Step 2: Set the value of *i* to 1

Step 3: Set the value of *j* to 2

Step 4: While ($i < k$) do steps 5 through 11

Step 5: While (j k) do steps 6 through 9

Step 6: If $N_i + N_j = SUM$ then

Step 7: Print (N_i, N_j)

Step 8: Stop

Else

Step 9: Set the value of j to $j + 1$

Step 10: Set the value of i to $i + 1$

Step 11: Set the value of j to $i + 1$

Step 12: Print the message 'Sorry, there is no such pair of values.'

24. Set count to 0

Set sum to 0 Get

a value for V

While V ≠ -1

Set sum to sum + V Set

count to count + 1 Get

the next value for V

End of the loop

Let's make sure that we had at least one value so we don't divide by 0

If (count > 0)

Set average to sum / count

Print the value of average

Else

Print the message 'I was given no input data'

Stop

25. Set adjacent to NO

Get values for V1 and V2 *We can do this since we know there are at least 2 values*

While (V2 ≠ -1) AND (adjacent = NO)

    If V1 = V2

        Set adjacent to YES

    Else

        Set V1 = V2

        Get a new value for V2

End of loop

Print the value of adjacent

Stop

26. We only need to make one simple change. Instead of writing

Print the value of adjacent

we change that to read:

If (adjacent = YES)

    Print the message 'Yes, the numbers ' V1 ' and ' V1 ' are adjacent.'

Else

    Print just the value of adjacent

Discussion of Challenge Work

1. The general algorithm is fairly clear, in English, in the text.

Step 1: Read values for *start, step,* and *accuracy*

Step 2: While |*step*| > *accuracy* do steps 3 through 9

Step 3:If *f(start)* > 0 then set *FirstSign* to +

Step 4:Else set *FirstSign* to –

Step 5: Do steps 6 through 8

Step 6: Set the value of *start* to *start* + *step*

Step 7: If *f(start)* > 0 then set the value of *Sign* to +

Step 8: Else set the value of *Sign* to –

while (*Sign* = *FirstSign)*

Step 9: If |*step*| $\geq$ *accuracy* then set the value of *step* to (-0.1)*step*

Step 10: Set the value of *root* to *start* – *step*/2

Step 11: Print the value of *root*.

2. Many excellent simulations of sorting algorithms are available on the Web, suggest students examine them if they have questions about this algorithm.

The Find Largest algorithm given in the book always searches the whole list. First, we should create a variation that takes, in addition to the list of values, two indices which bound the range of the list that should be searched. Also, it is easiest to return the location of the largest value, for use in the sort algorithm. Below is a sketch of how it should change:

FindLargest(A, *start*, *end*)

Step 1: Set the value of *loc* to *start*

Step 2: Set the value of *i* to *start* + 1

Step 3: While (*i* ≤ *end*) do

Step 4: If $A_i > A_{loc}$ then Step

5: Set *loc* to *i* Step 6:

Add 1 to the value of *i* Step 7: End

of the loop

Step 8: Return the value *loc*

The Selection Sort algorithm is quite simple, once we have a suitable form for the Find Largest portion of it.

Step 1: SelectionSort(*A*, *n*)

Step 2: Set *lastpos* to *n*

Step 3: While (*lastpos* $\geq$ 1) do

Step 4: Set *biggestpos* to FindLargest(*A*, 1, *lastpos*)

Step 5: Swap $A_{lastpos}$ and $A_{biggestpos}$

Step 6: Subtract 1 from *lastpos*

Step 7: End of loop

3. Students should be provided with concrete leads to reference materials about non-European mathematicians, including references to online resources.

4.