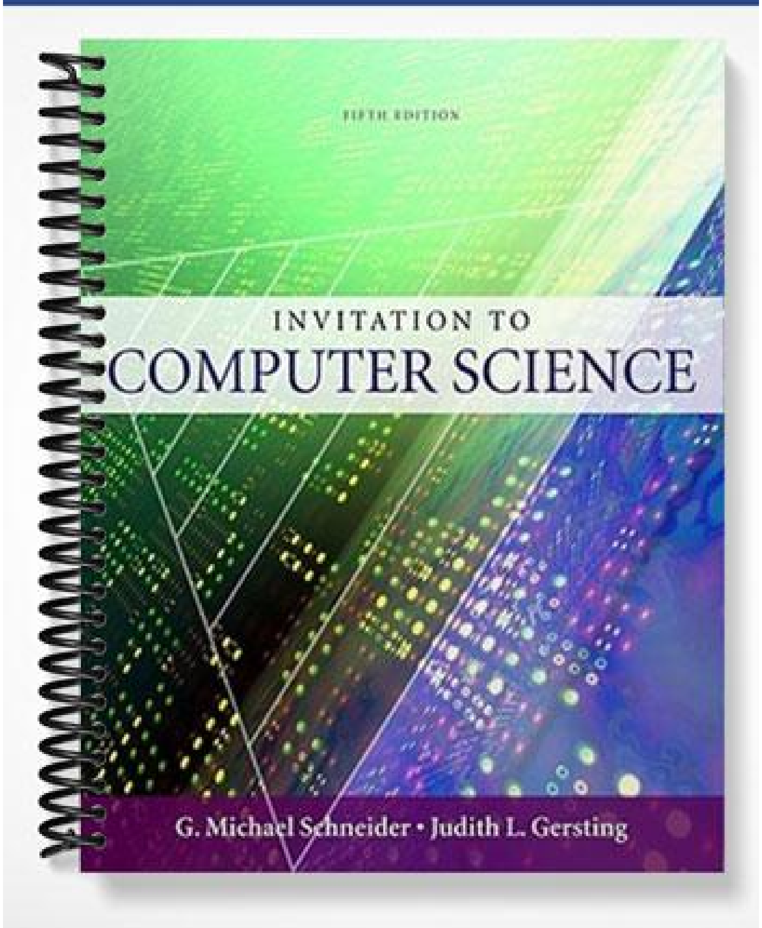
SOLUTIONS MANUAL



Chapter 2: Algorithm Discovery and Design

Solutions to End-of-Chapter Exercises

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 $y \neq 0$ then
Print the value of (x/y)

Else
Print the message "Unable to perform the division."

(b)

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

Else
Set the value of *Circum* to $2 * \pi * r$

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

End of the While loop

Step 12: Print the values of *Won*, *Lost*, and *Tied*

Step 13: If *Won* = 10, then

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 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* ≤ 40)

Step 12: *DT* = 0

Step 13: *TH* = 0

Step 14: *Reg* = *HoursWorked*

Step 15: *GrossPay* = *PayRate* * *HoursWorked*

+ 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 $i \leq 10000$.” If NAME is ever found in the loop then *Found* gets set to YES and the loop stops, and the algorithm ends after executing steps 8 and 9. If NAME 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 *NAME*, N_1, ... N_{10000}, and T_1, ..., T_{10000}

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

Step 3: While ($i \leq 10000$) do steps 4 through 7

Step 4: If *NAME* equals N_i then

Step 5: Print the phone number T_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 *NAME* "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.

Step 1: Get the values L_1, L_2, . . ., L_N of the numbers in the list

Step 2: If *N* is even then

Let $M = N / 2$

Else

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

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

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

in the list L_1, L_2, . . ., 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 changing them 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 $R \neq 0$) 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 1: Get the values for *nextChar* and *k*

Step 2: While (*nextChar* ≠ $) do steps 3 through 5

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 ($front \leq back$) 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 ($j \le k$) do steps 6 through 9

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

Step 7: Print (N_i, N_k)

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."

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 ($i \leq end$) 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.