Questions: Chapter 1: 10,11

Chapter 2: 20, 23, 24

Chapter 3: 11, 25, 33 (***Modification: Please assume selection sort is used to sort and that the list has 12,000 items***).

Chapter 1:

Q 10: The following is Euclid's 2,300-year-old algorithm for finding the greatest common divisor of two positive integers I and J.

Step Operation

1. 1 Get two positive integers as input; call the larger value I and the smaller value J
2. 2 Divide I by J, and call the remainder R
3. 3 If R is not 0, then reset I to the value of J, reset J to the value of R, and go back to Step 2
4. 4 Print out the answer, which is the value of J
5. 5 Stop

   Go through this algorithm using the input values 20and32.After each step of the algorithm is completed, give the values of I, J, and R. Determine the final output of the algorithm.

   Does the algorithm work correctly when the two inputs are 0 and 32? Describe exactly what happens, and modify the algorithm so that it gives an appropriate error message.

Q10: Solution

a.

1. I = 32, J = 20
2. I = 32, J = 20, R = 12
3. I = 20, J = 12
4. I = 20, J = 12, R = 8
5. I = 12, J = 8
6. I = 12, J = 8, R = 4
7. I = 8, J = 4
8. I = 8, J = 4, R = 0
9. Ans = 4.

b.

1. I = 32, J =0
2. I = 32, J = 0, R = Undefined
   The algorithm cannot proceeded as it tries to divide 32 by 0 which leads to an undefined answer as a number is not divisible by 0.

   Modification:

   1. Get two positive integers as input; call the larger value I and the smaller value J
   2. If J = 0, print "Error number cannot be divided by 0!"
   3. Divide I by J, and call the remainder R
   4. If R is not 0, then reset I to the value of J, reset J to the value of R, and go back to Step 2
   5. Print out the answer, which is the value of J

6. Stop

Q11: A salesperson wants to visit 25 cities while minimizing the total number of miles she must drive. Because she has studied computer science, she decides to design an algorithm to determine the optimal order in which to visit
the cities to (1) keep her driving distance to a minimum, and (2) visit each city exactly once. The algorithm that she has devised is the following:

The computer first lists all possible
ways to visit the 25 cities and then, for each one, determines the total mileage associated with that particular ordering. (Assume that the computer has access to data that gives the distances between all cities.) After determining the total mileage for each possible trip, the computer searches for the ordering with the minimum mileage and prints out the list of cities on that optimal route, that is, the order in which the salesperson should visit her destinations.

If a computer could analyze 10,000,000 separate paths per second, how long would it take to determine the optimal route for visiting these 25 cities? On the basis of your answer, do you think this is a feasible algorithm? If
it is not, can you think of a way to obtain a reasonable solution to this problem?

Q11: Solution

At each choice node, number of possible cities decreases by 1. At first choice, number of possible cities = n, second n-1, third n -1 -1… Where n is the number of cities she wants to visit. Hence total number of path = 25!, 25! = 1.551121*10^25 ways. Analysing 10,000,000 a second for these paths = 25!/10,000,000, which is = 1.511121*10^18 seconds. Such a solution is not feasible as this is a very large amount of time and not practical.

Introducing a central location from this

Chapter 2:

1. Design an algorithm that is given a positive integer N and determines whether N is a prime number, that is, not evenly divisible by any value other than 1 and itself. The output of your algorithm is either the message ‘not prime’, along with a factor of N, or the message ‘prime’.

Question 20 solution:

```
Set N to be inputted value

Prime = Y

k = 2

While (k<N) AND (Prime == Y)

{

        If(N % k is equal to 0)
        {
                Prime = N;
        }
        k ++;
}
If (Prime == N)
{
        Print "(Not prime and factor is", k) ;
}
Else
{
        Print(Prime);
}
End
```

Q23. Design and implement an algorithm that gets as input a list of k integer values $N_1, N_2, \ldots, N_k$ as well as a special value SUM. Your algorithm must locate a pair of values in the list N that sum to the value SUM. For example, if your list of values is 3, 8, 13, 2, 17, 18, 10, and the value of SUM is 20, then your algorithm would output either of the two values (2, 18) or (3, 17). If your algorithm cannot find any pair of values that sum to the value SUM, then it should print the message 'Sorry, there is no such pair of values'.

Question 23 Solution

```
Set the elements of the list as $N_1, N_2, \ldots, N_k$
Set value inputted as SUM
pair = N;
Set i = 1;
While (pair == N and (i <= k))
{
    j = i +1;
    While ((j <= k) and pair == N)
    {
            If ($N_i$ + $N_j$ = SUM)
            {
                    pair = Y;
                    $P_1$ = $N_i$ ;
                    $P_2$ = $N_j$;
            }
            j++;
    i++;
    }
    If (pair == Y)
    {
          Print ($P_1$, $P_2$);
    }
    Else
    {
        Print(" Sorry, there is no such pair of values");
    }
```

Q24. Instead of reading in an entire list N_1, N_2,
. . . all at once, some algorithms (depending on the task to be done) read in only one element at a time and process that single element completely before inputting the next one. This can be a useful technique when the list is very big (e.g., billions of elements) and there might not be enough memory in the computer to store it in its entirety. Write an algorithm that reads in a sequence of values $V \geq 0$, one at a time, and computes the average of all the numbers. You should stop the computation when you input a value of $V = -1$. Do not include this negative value in your computations; it is not a piece of data but only a marker to identify the end of the list.

Question 24 Solution

```
i = 0;
Total = 0;
Input value of V to $N_1$
```

```
While ( V != -1)

{

        Total = Total + V;

        i++;

        V = N_{i + 1} ;

}

Avg = Total/I;

Print (Avg);
```

Chapter 3:

Exercises 11–14 refer to another algorithm, called **bubble sort**, which sorts an n-element list. Bubble sort makes multiple passes through the list from front to back, each time exchanging pairs of entries that are out of order. Here is a pseudocode version:

```
1. Get values for n and the n list items

2. Set the marker U for the unsorted section at the end of the list

3. While the unsorted section has more than one element, do Steps 4 through 8

4. Set the current element marker C at the second element of the list

5. While C has not passed U, do Steps 6 and 7

6. If the item at position C is less than the item to its left, then exchange these two items

7. Move C to the right one position

8. Move U left one position

9. Stop
```

11. For each of the following lists, perform a bubble sort, and show the list after each exchange. Compare the number of exchanges done here and in the Practice Problem at the end of Section 3.3.3.

    a. 4,8,2,6
    b. 12, 3, 6, 8, 2, 5, 7
    c. D,B,G,F,A,C,E,H

    Question 11 Solution

    a.

1. n = 4 list n = 4,8,2,6
2. 4,8(C),2,6(U) (As 8 > 4 -> C moves to the right)
3. 4,8,2(C),6(U) (As 2 < 8 -> **swap** and C moves to the right)
4. 4,2,8,6(U)(C) (As 6 < 8 -> **swap**, C passes U -> U moves to the right and C moves to 2nd)
5. 4,2(C),6,(U)8 (As 2 < 4 -> **swap** and C moves to the right)
6. 2,4,6(C)(U),8 (As 6 > 4 -> C moves to the right)
7. 2,4,6(U),8(C) (As C passes U -> U moves to the right and C moves to 2nd)
8. 2,4(C)(U),6,8 (As 4 > 2 -> C moves to the right)
9. 2(U),4(C),6,8 (As C passes U -> U moves to the right and C moves to 2nd)
10. Stop 2,4,6,8 (Unsorted section has only 1 element)

3 swaps needed. More swaps for bubble sort than selection sort

b.

1. n = 7 list n 12, 3, 6, 8, 2, 5, 7
2. 12, 3(C), 6, 8, 2, 5, 7(U) (As 3 < 12 -> **swap** and C moves to the right)
3. 3, 12, 6(C), 8, 2, 5, 7(U) (As 6 < 12 -> **swap** and C moves to the right)
4. 3, 6, 12, 8(C), 2, 5, 7(U) (As 8 < 12 -> **swap** and C moves to the right)
5. 3, 6, 8, 12, 2(C), 5, 7(U) (As 2 < 12 -> **swap** and C moves to the right)
6. 3, 6, 8, 2, 12, 5(C), 7(U) (As 5 < 12 -> **swap** and C moves to the right)
7. 3, 6, 8, 2, 5, 12, 7(C)(U) (As 7 < 12 -> **swap** and C moves to the right)
8. 3, 6, 8, 2, 5, 7, 12(U)(C) (As C passes U -> U moves to the right and C moves to 2nd)
9. 3, 6(C), 8, 2, 5,7(U), 12 (As 6 > 3 -> C moves to the right)
10. 3, 6, 8(C), 2, 5,7(U), 12 (As 8 > 6 -> C moves to the right)
11. 3, 6, 8, 2(C), 5,7(U), 12 (As 2 < 8 -> **swap** and C moves to the right)
12. 3, 6, 2, 8, 5(C),7(U), 12 (As 5 < 8 -> **swap** and C moves to the right)
13. 3, 6, 2, 5, 8,7(C)(U), 12 (As 7 < 8 -> **swap** and C moves to the right)
14. 3, 6, 2, 5, 7,8(U), 12 (C)(As C passes U -> U moves to the right and C moves to 2nd)
15. 3, 6(C), 2, 5, 7(U),8, 12 (As 6 > 3 -> C moves to the right)
16. 3, 6, 2(C), 5,7(U), 8,12 (As 2 < 6 -> **swap** and C moves to the right)
17. 3, 2, 6, 5(C)(U),7, 8,12 (As 5 < 6 -> **swap** and C moves to the right)
18. 3, 2, 5, 6,7(C)(U), 8,12 (As 7 > 6 -> C moves to the right)
19. 3, 2, 5, 6,7(U), 8(C),12 (As C passes U -> U moves to the right and C moves to 2nd)
20. 3, 2(C), 5, 6(U),7, 8,12 (As 2 < 3 -> **swap** and C moves to the right)
21. 2,3,5(C)(U),6,7,8,12 (As 5 > 3 -> C moves to the right)
22. 2,3,5(U),6(C),7,8,12 (As C passes U -> U moves to the right and C moves to 2nd)
23. 2,3(C)(U),5,6,7,8,12 (As 3 > 2 -> C moves to the right)
24. 2,3(U),5(C),6,7,8,12 (As C passes U -> U moves to the right and C moves to 2nd)
25. 2,3,5,6,7,8,12 (Unsorted section has only 1 element)

    12 swaps needed. More needed for bubble sort than selection sort

c. (Swaps only)

1. D,B(C),G,F,A,C,E,H(U) (As B < D -> **swap** and C moves to the right)
2. B,D,G,F(C),A,C,E,H(U) (As F < G -> **swap** and C moves to the right)
3. B,D,F,G,A(C),C,E,H(U) (As A < G -> **swap** and C moves to the right)
4. B,D,F,A,G,C(C),E,H(U) (As C < G -> **swap** and C moves to the right)
5. B,D,F,A,C,E,G,(C)H(U) (As E < G -> **swap** and C moves to the right)
6. B,D,F,A(C),C,E,G(U),H (As A < F -> **swap** and C moves to the right)
7. B,D,A,F,C(C),G,E(U),H (As C < F -> **swap** and C moves to the right)
8. B,D,A,C,F,E(C),G(U),H (As E < F -> **swap** and C moves to the right)
9. B,D,A(C),C,E,F,G(U),H (As A < D -> **swap** and C moves to the right)
10. B,A,D,C(C),E,F,G(U),H (As C < D -> **swap** and C moves to the right)
11. B,A(C),C,D,E,F(U),G,H (As A < B -> **swap** and C moves to the right)

12. A,B,C(C),E,F,G(U),H (**FINAL**)

11 swaps needed. More swaps for bubble sort than selection sort.

25. Consider the following sorted list of names.

Arturo, Elsa, JoAnn, John, José, Lee, Snyder, Tracy

1. Use binary search to decide whether Elsa is in this list. What names will be compared with Elsa?
2. Use binary search to decide whether Tracy is in this list. What names will be compared with Tracy?
3. Use binary search to decide whether Emile is in this list. What names will be compared with Emile?

Question 25 Solution

1.
    1. Start = Arturo Finish = Tracy Middle = John
    2. Compare Elsa to John. Elsa is before John
    3. Start = Arturo End = JoAnn(Middle -1) Middle = Elsa
    4. Compare Elsa to Elsa. Elsa is found
    5. Found = Y
    6. *Names compared = John and Elsa*
2.
    1. Start = Arturo Finish = Tracy Middle = John
    2. Compare Tracy to John. Tracy is after John
    3. Start = Jose(Middle +1) End = Tracy Middle = Lee
    4. Compare Tracy to Lee
    5. Start = Snyder Finish = Tracy Middle = Snyder
    6. Compare Tracy to Snyder. Tracy is after Snyder
    7. Start = Tracy Finish = Tracy Middle = Tracy
    8. Compare Tracy to Tracy. Tracy is found
    9. Found = Y
    10. *Names compared = John, Lee, Snyder and Tracy*
3.
    1. Start = Arturo Finish = Tracy Middle = John
    2. Compare Emile to John. Emile is before John
    3. Start = Arturo End = JoAnn(Middle -1) Middle = Elsa
    4. Compare Emile to Elsa. Emile is after Elsa
    5. Start = End = JoAnn(Middle +1)
    6. End of list -> Emile is not found
    7. Found = N
    8. *Names compared = John, Elsa*

33. At the end of Section 3.4.2, we talked about the trade-off between using sequential search on an unsorted list as opposed to sorting the list and then using binary search. If the list size is n = 100,000, about how many worst-case searches must be done before the second alternative is better in terms of number of comparisons? (Hint: Let p represent the number of searches done.)

Question 33 Solution

Sorting is Big O of n(n-1)/2 in worst case

Binary search is Big O of lgn in worst case

Sequential search is Big O of n in worst case

P = number of searches

Sorting = (12000)(12,000-1)/2

Sequential search = (12000)

Comparing for P searches:

P(12,000) > (12,000)(12,000 -1)/2 + 15P

P(12,000) – 15P > (12,000)(12,000 -1)/2

P > (12,000)(12,000 -1)/2 / [(12,000) - 15]

P > 6007.0 (5S.F)

If more than 6008 worst case sequential searches are performed, the second alternative of sorting and preforming a binary search is the better outcome.