Zayd Hammoudeh
CS255-02
M/W 12pm-1:15pm
009418877

CS255, Spring 2014, SJSU
Homework 2

Fernando Lobo

Due 2/24

Due: Feb 24, 2014

Problem 1
Exercise 2.3-4, page 39 from textbook. ✓

Problem 2
Exercise 4.4-4, page 93 from textbook. ✓

Problem 3
Exercise 4.5-1, page 96 from textbook. ✓

Problem 4
Exercise 4.5-2, page 97 from textbook. ✓

Problem 5
Problem 4-1, page 107 from textbook. ✓

Problem 6
Write a divide and conquer algorithm to compute the minimum and maximum of an array of n integers. Write and solve the recurrence for the worst-case running time of your algorithm. ✓

Problem 7
Exercise 2.3-7, page 39 from textbook. ✓

Problem 8
Exercise 4.1-2, page 74 from textbook. ✓

Problem 9
Given an array with n distinct integers sorted in ascending order, we would like to know if there is an index i such that $A[i] = i$. Specify a divide and conquer algorithm to solve this problem with running time $O(\lg n)$. ✓

Problem 10
Problem 7-2, page 186 from textbook. ✓

Question #1- Exercise 2.3-4, page 39


```
//-------------------------------------------------------------------------//
//  Homework #2 Question #1 - Exercise 2.3-4, page 39 from the textbook. //
//  Express insertion sort as a recursive problem as follows: in  order  //
//  to sort A[1..n], we recursively sort A[1..n-1] then insert // A[n]   //
//  into the sort array A[1..n-1].                                       //
//-------------------------------------------------------------------------//


static void Q1_Insertion_Sort(int[] search_array, int right)
{

    //----- Recursion stop condition
    if (right == 0) return;

    //----- Divide Step
    Q1_Insertion_Sort(search_array, right - 1);

    //---- Combine (i.e. insertion step)
    while (right > 0 && search_array[right] < search_array[right - 1])
    {
        Swap<int>(ref search_array[right], ref search_array[right - 1]);
        right--;
    }


}
```

← Divide Step $T(n-1)$

} $\Theta(n)$

$T(n) = T(n-1) + \Theta(n)$ ← cannot use master method as not in the form $T(n) = aT(\frac{n}{b}) + f(n)$

$T(n) \leq c[1+2+3+\dots+n]$

$T(n) \leq c\sum_{i=1}^{n} i$

$T(n) \leq c \cdot \left(\frac{n \cdot (n+1)}{2}\right)$

$T(n) \leq \frac{cn^2}{2} + \frac{cn}{2}$

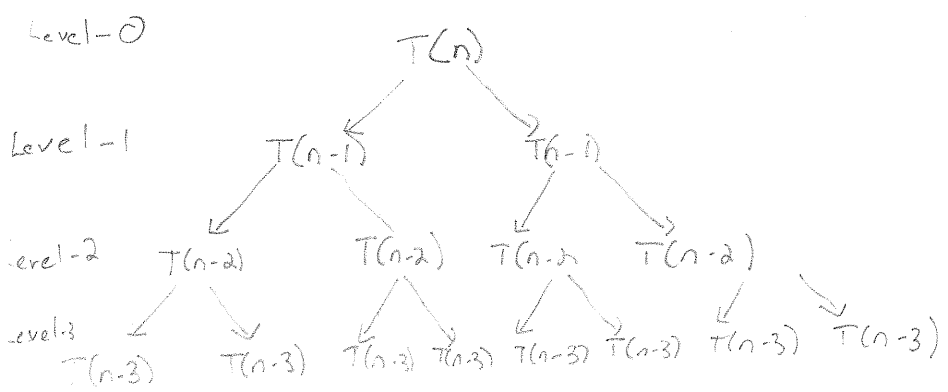
$T(n) \leq cn^2 - \frac{cn^2}{2} + \frac{cn}{2}$

$T(n) \leq cn^2 - \left(\frac{cn^2}{2} - \frac{cn}{2}\right)$

$T(n) = O(n^2)$ if $\frac{cn^2}{2} - \frac{cn}{2} > 0$, $n > 1$

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \text{ (no sorting as array of 1 is sorted).} \\ T(n-1) + \Theta(n) = \Theta(n^2), & \text{if } n > 1 \end{cases}$$

Question #2 Exercise 4.4-4 page 93



$$T(n) = \sum_{i=0}^{n} 2^i$$

$$\sum_{i=m}^{n-1} a^i = \frac{a^m - a^n}{1-a}$$

$m = 0$, $n = n+1$

$$\sum_{i=0}^{n} 2^i = \frac{2^0 - 2^{n+1}}{1-2}$$

$$\sum_{i=0}^{n} 2^i = \frac{1-2^{n+1}}{-1}$$

$$T(n) \leq 2^{n+1} - 1$$

$$T(n) \leq 2 \cdot 2^n - 1$$

$T(n) = O(2^n)$ if $c \geq 2$

$$T(n) = 2T(n-1) + 1$$

$$2 \cdot 2^{n-1} + 1 \leq c \cdot 2^n$$

$$2^{n-1+1} + 1 \leq c 2^n$$

$$2^n + 1 \leq c \cdot 2^n$$

$n \geq 0$, $c \geq 2$ Asymptotic upper bound holds.

$$T(n) = O(2^n)$$

Question #3 Exercise 4.5-1 page 96

a) $T = 2\,T(\frac{n}{4}) + 1$

$T(n) = aT(\frac{n}{b}) + f(n)$

$f(n) = 1$

$a = 2$

$b = 4$

Case #1 of Master Method

$f(n) = O(n^{\log_4(2-\varepsilon)})$ if $\varepsilon = 1$

$f(n) = O(n^{\log_4 1}) = O(n^0) = O(1)$

$\boxed{T(n) = \Theta(n^{\log_4 2}) = \Theta(\sqrt{n})}$

b) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

$a = 2$

$n = 4$

$f(n) = \sqrt{n}$

$\Theta(n^{\log_4 2}) = \Theta(n^{\frac{1}{2}})$

$\Theta(f(n)) = \Theta(\sqrt{n}) = \Theta$

Case #2 of Master Method.

$\boxed{T(n) = \Theta(\sqrt{n} \cdot \lg(n))}$

c) $T(n) = 2T(\frac{n}{4}) + n$

$\Theta(f(n)) = \Theta(n)$

$f(n) = \Omega(n^{\log_4 2 + \varepsilon})$ if $\varepsilon = 2$

$\Omega(n^{\log_4 4}) = \Omega(n^1) = f(n) = n$

Case #3 of Master Method

$T(n) = \Theta(f(n))$

$\boxed{T(n) = \Theta(n)}$

d) $T(n) = 2T(\frac{n}{4}) + n^2$

$\Theta(f(n)) = \Theta(n^2)$

$f(n) = \Omega(n^{\log_4(2+\varepsilon)})$ if $\varepsilon = 14$

$\Omega(n^{\log_4 16}) = \Omega(n^2) = f(n) = n^2$

Case #3 of Master method

$\boxed{T(n) = \Theta(f(n)) = \Theta(n^2)}$

Question # 4 - Exercise 4.5-2, page 97

Strassens Algorithm

$$T_1(n) = 7T(\frac{n}{2}) + \Theta(n^2) \quad T_1 = \Theta(n^{\lg_2 7})$$

Caesar's Algorithm

$$T_2(n) = aT(\frac{n}{4}) + \Theta(n^2)$$

$$T_2 < \Theta(n^{\lg_2 7})$$

$$\Theta(n^{\log_4 a}) \leq \Theta(n^{\lg_2 7})$$

$$\log_4 a < \log_2 7$$

$$\frac{\lg a}{\lg 4} < \lg 7$$

$$\frac{1}{2}\lg(a) < \lg(7)$$

$$\lg\sqrt{a} < \lg 7$$

$$\sqrt{a} < 7$$

$$a < 49$$

48

Problem #5 – Problem #4-1, page #107

a) $T(n) = 2 \cdot T(\frac{n}{2}) + n^4$

$a = 2$
$b = 2$
$f(n) = n^4$
$n^4 = \Omega(n^{\log_2(2+\varepsilon)})$
if $\varepsilon = 14 \rightarrow n^4 = \Theta(n^{\log_2(2+14)})$
Case #3 of master method
$\boxed{T(n) = \Theta(n^4)}$

b) $T(n) = T(\frac{7n}{10}) + n$

$a = 1$
$b = \frac{10}{7}$
$f(n) = n$
$n = \Omega(n^{\log_{\frac{10}{7}}(\varepsilon+1)})$
$\varepsilon = \frac{3}{7} \rightarrow n = \Omega(n^{\log_{\frac{10}{7}}(\frac{10}{7})})$
$n = \Omega(n)$
Case #3
$\boxed{T(n) = \Theta(n)}$

c) $T(n) = 16T(\frac{n}{4}) + n^2$

$a = 16$
$b = 4$
$f(n) = n^2$
$\Theta(f(n)) = \Theta(n^{\log_4 16}) = \Theta(n^2)$
Case #2
$\boxed{T(n) = \Theta(n^2 \lg(n))}$

d) $T(n) = 7T(\frac{n}{3}) + n^2$

$a = 7$
$b = 3$
$f(n) = n^2$
$n^2 = \Omega(n^{\log_3 7 + \varepsilon})$
if $\varepsilon = 2 \rightarrow$
case #3 of master method
$\boxed{T(n) = \Theta(n^2)}$

e) $T(n) = 7T(\frac{n}{2}) + n^2$

$a = 7$
$b = 2$
$f(n) = n^2$
Case #1 of master method
$n^2 = O(n^{\log_2 7 - \varepsilon})$
if $\varepsilon = 3$
$n^2 = O(n^{\log 4}) = O(n^2)$
$\boxed{T(n) = \Theta(n^{\lg(7)})}$

f) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

$a = 2$
$b = 4$
$f(n) = \sqrt{n}$
Case #2
$\sqrt{n} = \Theta(n^{\log_4 2}) = \Theta(n^{\frac{1}{2}}) = \Theta(\sqrt{n})$
$\boxed{T(n) = \Theta(\sqrt{n} \lg(n))}$

g) $T(n) = T(n-2) + n^2$

Two Cases
n is odd
$T(n) = (n)^2 + (n-2)^2 + \ldots + 1^2$

n is even
$T(n) = n^2 + (n-2)^2 + \ldots 10^2$

If odd $T(n) \leq T(n+1)$ so increment by one. I will solve the even case which by extension covers all odd cases

$T(n) = \sum_{i=0}^{\frac{n}{2}} (2i)^2$

$T(n) = 4 \sum_{i=0}^{\frac{n}{2}} i^2$

$T(n) = 4 \left(\frac{(\frac{n}{2})(\frac{n}{2}+1)(2\cdot\frac{n}{2}+1)}{6} \right)$

$\rightarrow T(n) = 4\left(\frac{(\frac{n}{2})^3}{3} + \frac{(\frac{n}{2})^2}{2} + \frac{(\frac{n}{2})}{4} \right)$

$T(n) = 4\left(\frac{n^3}{24} + \frac{n^2}{8} + \frac{n}{12} \right)$

$T(n) = \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3}$

Lower bound $T(n) = \Omega(n^3)$
If $c_1 = \frac{1}{6}$ then $\forall n \left[\frac{n^3}{6} \leq \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3} \right]$
If $c_2 = 1$ then Upper bound $T(n) = O(n^3)$
$\frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3} \leq n^3$ for all $n \quad \frac{n^3}{6} \leq \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3}$

$\therefore \boxed{T(n) = \Theta(n^3)}$

Question #6 - Algorithm to find min and max,

Untitled

```
static void Q6_Find_Min_and_Max(int[] search_array, int left, int right, ref int min, ref int max)
{
    int mid;

    //----- Recursive stop condition
    if (right < left) return;

    //----- Divide step
    mid = (right + left) / 2;
    Q6_Find_Min_and_Max(search_array, left, mid - 1, ref min, ref max); //----- Left split
    Q6_Find_Min_and_Max(search_array, mid + 1, right, ref min, ref max); //---- Right split

    //----- Combine step
    if (search_array[mid] < min) min = search_array[mid];
    if (search_array[mid] > max) max = search_array[mid];

    return;
}
```

Written in C#

Algorithm Summary

- Find min and max in left half of the array
- Find min and max in right half of the array. min and max in right half takes lefts min and max, If this code was written multithreaded, it would not be as efficient but since single threaded it is fine
- Compare min and max from left and right side to determine overall min/max.

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(1) = 2T\left(\frac{n}{2}\right) + c$$

Master method

Case #1

$$f(n) = O\left(n^{\lg_2 2 - \varepsilon}\right)$$

$$\varepsilon = 1$$

$$f(n) = \Theta\left(n^{\lg_2 1}\right) = O(n^0) = \Theta(1)$$

$$\boxed{T(n) = \Theta\left(n^{\lg_2 2}\right) = \Theta(n)}$$

7) Question #7 - Exercise 2.3-7, page 39.

Algorithm Description

- Step #1 - Iterate through array containing set "S" and calculate $\delta_i = x - s_i$ where $s_i \in S$. Store these in an array δ.

$$\forall s_i \{ s_i \in S \rightarrow \delta_i = x - s_i$$

$T_1(n) = \Theta(n)$ - n times through the loop with constant time actions each time (e.g. subtraction, index increment, etc.)

- Step #2 - Use merge sort to sort array S from minimum to maximum. Call this array S'.

$T_2(n) = \Theta(\text{merge sort}) = \Theta(n \cdot \lg(n))$

- Step #3: For all elements, δ_i, in δ, use a binary search to see if it exists in the sorted array S'.

$$T_3(n) = n \cdot \Theta(\text{Binary Search}) = n \cdot \lg(n)$$

Total execution time: $T'(n)$

$$T'(n) = T_1(n) + T_2(n) + T_3(n)$$

$$T'(n) = \Theta(n) + \Theta(n\lg(n)) + \Theta(n\lg(n))$$

$$T'(n) = cn + dn\lg(n) + e\,n\lg(n)$$

$$T'(n) = \Theta(\max\{n, n\lg(n), n\lg(n)\})$$ (By Big-O/Big Theta rule of sums)

$$T'(n) = \Theta(n\lg(n))$$

Question # 8- Exercise 4.1.2 page 74

Untitled

```
static Tuple<int, int, int> Q8_Brute_Force_Maximum_Subarray(int[] search_array, int n)
{
    int i, j;
    int max_value = System.Int32.MinValue;
    int max_start =-1;
    int max_end = -1;
    int cur_sum;
    int[] diff_array;
    Tuple<int, int, int> output;


    //---- Calculate the difference on a daily basis (O(n)).
    diff_array = new int[n];
    for(i=1; i<n; i++)
        diff_array[i] = search_array[i]-search_array[i-1];

    //----- Embedded for loop for loops to calculate maximum subarray
    for(i=0; i < n-1; i++){
        cur_sum = 0; //---- Reset the counter

        //----- inner loop and check each min.max
        for (j = i + 1; j < n; j++)
        {
            cur_sum += diff_array[j];
            if(cur_sum > max_value){
                max_value = cur_sum;
                max_start = i;
                max_end = j;
            }
        }
    }

    output = new Tuple<int, int, int>(max_value, max_start, max_end);
    return output;

}
```

Code above written in C#

Pseudo code

```
//--- CalculateDiffArray
for i=2 To n
  Diff[.] = A[.] - A[i-1]

for i=1 to n{
   temp_sum=0
   for j=i+1 to n{
      temp_sum += Diff[j]
      if(temp_sum > max_subarray){
         max_subarray = temp_sum
         start = i
         end=j
      }
   }
}
```

Question #9 - Search function with $\Theta(\lg(n))$

Untitled

```
static int Q9_Find_Element(int[] search_array, int search_element, int left, int right, ref int
numb_calls)
{
    int mid;
    //----- Recursion halt condition
    if (right < left) return -1;

    numb_calls++;

    mid = (left + right) / 2;
    if (search_element == search_array[mid])
        return mid;
    else if(search_element > search_array[mid])
        return Q9_Find_Element(search_array, search_element, mid + 1, right, ref numb_calls);
    else //if(search_element < search_array[mid])
        return Q9_Find_Element(search_array, search_element, left, mid - 1, ref numb_calls);

}
```

Simple Binary search algorithm

$$T(n) = T\left(\left\lceil \frac{(n-1)}{2} \right\rceil\right) + O(1) \quad \text{— worst case}$$

$$\left\lceil \frac{(n-1)}{2} \right\rceil < \frac{n}{2}$$

$$T(n) \leq T\left(\frac{n}{2}\right) + O(1)$$

Solve using master method case #2

$$= \Theta\left(n^{\log_2 1}\right) = \Theta(n^0) = \Theta(1)$$

$$\boxed{T(n) = \Theta(\lg(n))}$$

Question #10 - Problem 7.2 page 186

a) All elements equal is the same as presorted.

Partition command is:

"if A[j] ≤ A[r] Then
  Swap A[i], A[j]"

at the end i=r and split will be → $T(n) = T(n-1) + T(0) + \Theta(n)$

$T(n) = \Theta(n^2)$ (can solve via closed form solution where $T(n) = \sum_{i=0}^{n} i$)

b) See attached C# code for function "Q10_Partition_Prime(int[]A, int p, int r)"
- Two loops of complexity $\Theta(r-p)$ so total is $\Theta(r-p)$ (page 11)

c) See attached C# code "Q10_Randomized_Partition_Prime(int[]A, int p, int r)" and "Q10_Quick_Sort_Prime(int[]A, int p, int r)". (page 11 and 12)

d) $0 \le d \le 1$

Previous distinct model $T(n) = T(d(n-1)) + T((1-d)\cdot(n-1)) + \Theta(n)$

where $d\cdot(n-1) + (1-d)\cdot(n-1) = n-1$

In the model that handles objects equal to A[r] the recursion changes slightly to

$$T(n) = T(d\cdot(n-1-s)) + T((1-d)(n-1-s)) + \Theta(n)$$

where $s \ge 0$ and is the number of elements in array "A" that are equal to A[r]. (with no distinct handling)

- In previous recursion for T(n), the complexity of the next level recursion had to be $\Theta(n-1)$. Now with this revised approach, it is $\Theta(n-1-s)$ For the case of all equal elements as in 7.2 (a) above, the run time would be

$$T(n) = T(d(n-1-s)) + T((1-d)\cdot(n-1-s)) + \Theta(n)$$

$$s = n-1$$

$$T(n) = T(0) + T(0) + \Theta(n)$$

$T(n) = \Theta(n)$ for problem a w/ this new algorithm.

```
static void Q10_Quick_Sort_Prime(int[] A, int p, int r)
{
    Tuple<int, int> partition_locations;

    //----- Recursion halt condition
    if(p>=r) return;

    //----- Divide
    partition_locations = Q10_Randomized_Partition_Prime(A, p, r);


    //---- Conquer
    Q10_Quick_Sort_Prime(A, p, partition_locations.Item1 - 1); //---- left half
    Q10_Quick_Sort_Prime(A, partition_locations.Item2 + 1, r);//---- Right half

}

static Tuple<int, int> Q10_Randomized_Partition_Prime(int[] A, int p, int r)
{
    Random rand = new Random();
    Tuple<int, int> Q10_partition_locations;
    int i;

    i = rand.Next(p,r+1);//---- Need to add 1 to r to make it inclusive
    Swap<int>(ref A[i], ref A[r]);

    //----- Perform partition and return the tuple
    Q10_partition_locations = Q10_Partition_Prime(A, p, r);
    return Q10_partition_locations;

}

static Tuple<int, int> Q10_Partition_Prime(int[] A, int p, int r)
{
    int q, t, i, j, end, numb_swapped;
    Tuple<int, int> output;


    i = p - 1;
    end = r;
    j = p;
    while(j<end)
    {
        if( A[j] == A[r] )
        {
            end--;
            Swap<int>(ref A[j], ref A[end]);
            continue;//----- Go to top of the loop to prevent an increment
        }

        if (A[j] < A[r])
        {
            i++;
            Swap<int>(ref A[i], ref A[j]);
        }
        j++;
    }

    //----- Move q into its place split between the two halfs
    q = ++i;
    numb_swapped = r - end + 1;
    for (j = 0; j < numb_swapped; j++)
```

removes duplicates from further sorting

3 Swaps random number

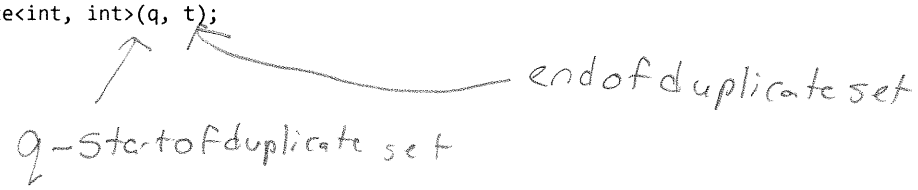
moves all equal elements to end of the array

$\Theta_1(r-p)$

$\Theta_2(r-p)$

```
        {
            Swap<int>(ref A[i+j], ref A[end+j]);
        }
        t = i+j-1;

        //------ Create the output and return the result
        output = Tuple.Create<int, int>(q, t);
        return output;
    }
```



Appendix – Verification Source Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class CS255_HW2
    {

        private static int MIN_INTEGER = System.Int32.MinValue;
        private static int MAX_INTEGER = System.Int32.MaxValue;

        //-------------------------------------------------------------//
        //-------------------------------------------------------------//

        static void Main(string[] args)
        {
            int Q1_n, Q6_n, Q8_n, Q10_n;//----- n in T(n)
            int min;
            int max;
            int loc;
            int search_val;
            int Q9_numb_calls;
            int[] list_of_numbers;
            int[] Q1_array, Q10_array;
            Tuple<int, int, int> Q8_Answer;

            //-----------------------------------------------------------------------------------//
            //                                  Question #1                                      //
            //-----------------------------------------------------------------------------------//

            //----- Create unsorted array
            Q1_n = 15;
            Q1_array = Create_Int_Array(Q1_n, 10, 30);

            Console.WriteLine("Question #1 - Take the unsorted array below and sort it...");
            PrintArray(Q1_array, Q1_n);

            //----- Sort the array then print the results.
            Q1_Insertion_Sort(Q1_array, Q1_n-1);
            Console.WriteLine("The sorted array is: ");
            PrintArray(Q1_array, Q1_n);
            Console.Write("\n\n\n\n");

            //-----------------------------------------------------------------------------------//
            //                                  Question #6                                      //
            //-----------------------------------------------------------------------------------//

            Q6_n = 10;
            min = int.MaxValue; //----- Define so always overwritten
            max = int.MinValue; //----- Define so always overwritten
            list_of_numbers = Create_Int_Array(Q6_n, 0, 50);

            //----- Peform Divide and conquer step
            Q6_Find_Min_and_Max(list_of_numbers, 0, Q6_n - 1, ref min, ref max);

            //----- Print the results
            Console.WriteLine("Question #6: Define a recursive function to find the minimum and maximum
values in an array.  The array is:");
            PrintArray(list_of_numbers, Q6_n);
            Console.WriteLine("The minimum value in the array was: " + Convert.ToString(min));
            Console.WriteLine("The maximum value in the array was: " + Convert.ToString(max) + "\n\n\n");
```

```
            //-------------------------------------------------------------------------------------//
            //                                  Question #8                                        //
            //-------------------------------------------------------------------------------------//

            Q8_n = 20;
            list_of_numbers = Create_Int_Array(Q8_n, 0, 50);
            Q8_Answer = Q8_Brute_Force_Maximum_Subarray(list_of_numbers, Q8_n);

            //----- Print the results
            Console.WriteLine("Question #8: Create code to calculate maximum sub array using the brute force approach:");
            PrintArray(list_of_numbers, Q8_n);
            Console.WriteLine("The maximum subbarray in the array was: " + Convert.ToString(Q8_Answer.Item1) );
            Console.WriteLine("The maximum subarray started at index {0} and ended at index {1}.\n\n\n", Q8_Answer.Item2, Q8_Answer.Item3);


            //-------------------------------------------------------------------------------------//
            //                                  Question #9                                        //
            //-------------------------------------------------------------------------------------//

            search_val = 20;
            Q9_numb_calls = 0;
            Console.Write("Using the sorted array from question #1 (i.e. insertion sort).  Searching for the number \"" + Convert.ToString(search_val) + "\".\n");
            loc = Q9_Find_Element(Q1_array, search_val, 0, Q1_n - 1, ref Q9_numb_calls);

            if (loc == -1)
                Console.WriteLine(Convert.ToString(search_val) + " was not found in the array.");
            else
                Console.WriteLine(Convert.ToString(search_val) + " was in the array at position " + Convert.ToString(loc) + ".");

            Console.WriteLine("The number of calls to determine the position of " + Convert.ToString(search_val) + " was " + Convert.ToString(Q9_numb_calls) + ".\n\n");


            //-------------------------------------------------------------------------------------//
            //                                  Question #1                                        //
            //-------------------------------------------------------------------------------------//

            //----- Create unsorted array
            Q10_n = 15;
            Q10_array = Create_Int_Array(Q10_n, 30, 40);

            Console.WriteLine("Question #10 - Take the unsorted array below and sort it...");
            PrintArray(Q10_array, Q10_n);

            //----- Sort the array then print the results.
            Q10_Quick_Sort_Prime(Q10_array, 0, Q10_n - 1);
            Console.WriteLine("The sorted array is through quick sort is: ");
            PrintArray(Q10_array, Q10_n);
            Console.Write("\n\n\n\n");


            //----- Cleanup and prevent memory leaks
            Q1_array = null;
            Q10_array = null;
            list_of_numbers = null;

        }
```

```
//-----------------------------------------------------------------------//
//  Homework #2 Question #1 - Exercise 2.3-4, page 39 from the textbook. //
//  Express insertion sort as a recursive problem as follows: in  order  //
//  to sort A[1..n], we recursively sort A[1..n-1] then insert // A[n]   //
//  into the sort array A[1..n-1].                                       //
//-----------------------------------------------------------------------//


static void Q1_Insertion_Sort(int[] search_array, int right)
{

    //----- Recursion stop condition
    if (right == 0) return;

    //----- Divide Step
    Q1_Insertion_Sort(search_array, right - 1);

    //---- Combine (i.e. insertion step)
    while (right > 0 && search_array[right] < search_array[right - 1])
    {
        Swap<int>(ref search_array[right], ref search_array[right - 1]);
        right--;
    }


}



static void Q6_Find_Min_and_Max(int[] search_array, int left, int right, ref int min, ref int max)
{
    int mid;

    //----- Recursive stop condition
    if (right < left) return;

    //----- Divide step
    mid = (right + left) / 2;
    Q6_Find_Min_and_Max(search_array, left, mid - 1, ref min, ref max); //----- Left split
    Q6_Find_Min_and_Max(search_array, mid + 1, right, ref min, ref max); //---- Right split

    //----- Combine step
    if (search_array[mid] < min) min = search_array[mid];
    if (search_array[mid] > max) max = search_array[mid];

    return;
}

static Tuple<int, int, int> Q8_Brute_Force_Maximum_Subarray(int[] search_array, int n)
{
    int i, j;
    int max_value = System.Int32.MinValue;
    int max_start =-1;
    int max_end = -1;
    int cur_sum;
    int[] diff_array;
    Tuple<int, int, int> output;


    //---- Calculate the difference on a daily basis (O(n)).
    diff_array = new int[n];
    for(i=1; i<n; i++)
        diff_array[i] = search_array[i]-search_array[i-1];
```

```
        //----- Embedded for loop for loops to calculate maximum subarray
        for(i=0; i < n-1; i++){
            cur_sum = 0; //---- Reset the counter

            //----- inner loop and check each min.
            for (j = i + 1; j < n; j++)
            {
                cur_sum += diff_array[j];
                if(cur_sum > max_value){
                    max_value = cur_sum;
                    max_start = i;
                    max_end = j;
                }
            }
        }

        output = new Tuple<int, int, int>(max_value, max_start, max_end);
        return output;

    }

    static int Q9_Find_Element(int[] search_array, int search_element, int left, int right, ref int
    numb_calls)
    {
        int mid;
        //----- Recursion halt condition
        if (right < left) return -1;

        numb_calls++;

        mid = (left + right) / 2;
        if (search_element == search_array[mid])
            return mid;
        else if(search_element > search_array[mid])
            return Q9_Find_Element(search_array, search_element, mid + 1, right, ref numb_calls);
        else //if(search_element < search_array[mid])
            return Q9_Find_Element(search_array, search_element, left, mid - 1, ref numb_calls);

    }


    static void Q10_Quick_Sort_Prime(int[] A, int p, int r)
    {
        Tuple<int, int> partition_locations;

        //----- Recursion halt condition
        if(p>=r) return;

        //----- Divide
        partition_locations = Q10_Randomized_Partition_Prime(A, p, r);


        //---- Conquer
        Q10_Quick_Sort_Prime(A, p, partition_locations.Item1 - 1);
        Q10_Quick_Sort_Prime(A, partition_locations.Item2 + 1, r);

    }


    static Tuple<int, int> Q10_Randomized_Partition_Prime(int[] A, int p, int r)
    {
        Random rand = new Random();
        Tuple<int, int> Q10_partition_locations;
```

```
            int i;

            i = rand.Next(p,r+1);//---- Need to add 1 to r to make it inclusive
            Swap<int>(ref A[i], ref A[r]);

            //----- Perform partition and return the tuple
            Q10_partition_locations = Q10_Partition_Prime(A, p, r);
            return Q10_partition_locations;

        }

        static Tuple<int, int> Q10_Partition_Prime(int[] A, int p, int r)
        {
            int q, t, i, j, end, numb_swapped;
            Tuple<int, int> output;


            i = p - 1;
            end = r;
            j = p;
            while(j<end)
            {
                if( A[j] == A[r] )
                {
                    end--;
                    Swap<int>(ref A[j], ref A[end]);
                    continue;//----- Go to top of the loop to prevent an increment
                }

                if (A[j] < A[r])
                {
                    i++;
                    Swap<int>(ref A[i], ref A[j]);
                }
                j++;
            }

            //----- Move q into its place split between the two halfs
            q = ++i;
            numb_swapped = r - end + 1;
            for (j = 0; j < numb_swapped; j++)
            {
                Swap<int>(ref A[i+j], ref A[end+j]);
            }
            t = i+j-1;

            //------ Create the output and return the result
            output = Tuple.Create<int, int>(q, t);
            return output;
        }

        static int[] Create_Int_Array(int n, int min_value, int max_value)
        {
            int cnt;
            int[] list_of_numbers;
            Random rand = new Random();

            //------ Populate Memory
            list_of_numbers = new int[n];
            for (cnt = 0; cnt < n; cnt++)
                list_of_numbers[cnt] = rand.Next(min_value, max_value - 1);

            return list_of_numbers;
```

```
        }


        static void Swap<T>(ref T left, ref T right)
        {
            T temp_var;
            temp_var = left;
            left = right;
            right = temp_var;
        }

        static void PrintArray(int[] print_array, int n)
        {
            int cnt;

            Console.Write("[ ");
            for (cnt = 0; cnt < n; cnt++)
            {
                Console.Write(Convert.ToString(print_array[cnt]));
                if (cnt + 1 != n) Console.Write(", ");
                else Console.WriteLine(" ]");
            }
        }

    }
}
```