COMP 3270 Assignment 1 12 problems 100 points 10% Credit
Due before 11:59 PM Thursday September 2

Instructions:

1. This is an individual assignment. You should do your own work. Any evidence of copying will result in a zero grade and additional penalties/actions.
2. Enter your answers in this Word file. Submissions must be uploaded **as a single file** (Word or PDF preferred, but other formats acceptable as long as your work is LEGIBLE) to Canvas before the due date and time. Don't turn in photos of illegible sheets. **If an answer is unreadable, it will earn zero points.** Cleanly handwritten submissions (print out this assignment and write answers in the space provided, with additional sheets used if needed) scanned in as PDF and uploaded to Canvas are acceptable.
3. **Submissions by email or late submissions (even by minutes) will receive a zero grade.** No makeup will be offered unless prior permission to skip the assignment has been granted, or there is a valid and verifiable excuse.
4. Think carefully; formulate your answers, and then write them out concisely using English, logic, mathematics and pseudocode (no programming language syntax).

1. Computational Problem Solving: Problem specification & Strategy design: Circle of Friends (8 points)
Spacebook (an intergalactic social network company) has hired you as a highly paid software engineer. Its CEO, Lieutenant Worf, wants to implement a privacy policy he calls "Friend Circles." All your immediate Spacebook friends are in your circle of friends of radius 1. Their friends are in your friend circle of radius 2. Their friends are in your friend circle of radius 3. And so on. Any member of Spacebook, if he/she/it can only be on at most one friend circle – i.e., if someone can be a member of multiple friend circles by the above logic, then that being can only be assigned to that friend circle which has the smallest radius. E.g., if Kirk is friends with you and he is also friends with Uhura and Sulu, and if Uhura is friends with Sulu then Sulu will be in your friend circle of radius 2 (not radius 3). Worf's privacy policy is that he would allow each Spacebook member to specify which friend circle(s) can view an item he/she/it posts. To implement such a policy, Spacebook's computers need to be able to compute all members of a particular member's friend circle of radius m, m > 0. Lt. Worf asks you (assume you speak Klingon fluently) to come up with a way to solve this problem.

So of course, you start by developing a well-defined problem specification! Some parts of such a specification is below. Complete the missing part – Correctness criteria.

Input(s):

1. For each Spacebook member, an alphabetically ordered linked list of his/her/its immediate Spacebook friends.

2. A member x.

3. A radius value m, m>0.

Output(s): An alphabetically ordered linked list of all members in x's friend circle of radius m.

Correctness Criteria: If $m = \lambda$, where $\lambda > 0$, then an alphabetically ordered linked list should be returned, containing every member within member X's friend circle of radius m at most once.

Given the problem specification above, come up with a computational strategy to solve it. Explain it below. Your explanation should be such that we can understand your strategy well enough to turn it into an algorithm, but not so detailed that what you provide becomes an algorithm and not a strategy.

1. Initialize a new linked list C
2. Beginning at the left of the input list, for each immediate friend in the list, add them and their own friend circle of radius m-1 to the end of list C.
3. Once every friend and their circle is added, scan the list C from left to right and remove any duplicates.
4. Sort list C into alphabetical order

2. CPS: Problem specification & inherent problem complexity: Data Reordering (5 points)

Problem Statement: Given an array containing some numbers, reorder those numbers in that array so that any negative numbers appear before any zeroes and any zeroes appear before any positive numbers. Come up with a well-defined problem specification for this.

Input(s): An array A of n numbers arranged in some order $[a_1, a_2, \ldots a_n]$

Output(s): An array B of n numbers $[b_1, b_2, b_3 \ldots, b_n]$ from A arranged with any negatives coming before any zeroes and any zeroes coming before any positives

Correctness Criteria: The array B must contain all the numbers in A, ordered with any negatives before any zeroes and any zeroes before any positives

The inherent complexity of this problem, if the input array has n, n>0, numbers, is **Ω(** n **)**

3. CPS: Strategy design, correctness & efficiency: Data Reordering (13 points)
Consider the three computational strategies to solve the above data reordering problem:

Strategy 1: Sort the array in the descending order.
Is this strategy correct? (Circle one) **yes** **no**
Will this strategy preserve the original relative orders of the negative, zero and positive numbers? (Circle one) **yes** **no** **maybe**

Strategy 2:

Scan the input array A left to right and copy all negative numbers to a new array B of the same size starting from its first index.

Then repeat the above, this time copying zeroes into B, starting with the index to the right of the last negative number copied into B.

Then repeat the above once more, this time copying positive numbers into B, starting with the index to the right of the last zero copied into B. Output array B.

Is this strategy correct? (Circle one) **yes** **no**
Will this strategy preserve the original relative orders of the negative, zero and positive numbers? (Circle one) **yes** **no** **maybe**
Can this strategy (as is with no modifications) be translated into an in-place algorithm? (Circle one)
yes **no**
Can this strategy (as is with no modifications) be translated into an on-line algorithm? (Circle one)
yes **no**

Strategy 3:

Scan the input array A, counting the number of negative, zero and positive numbers. Let these counts be x, y and z respectively.

Create a new array B of the same size and set a local variable neg to 1, zero to x+1, and pos to x+y+1.

Scan the input array from left to right, and if the current number being looked at is negative, copy it to B[neg] and increment neg, if it is a zero, copy it to B[zero] and increment zero, and if it is positive, copy it to B[pos] and increment pos.

Output array B.
Is this strategy correct? (Circle one) **yes** **no**
Will this strategy preserve the original relative orders of the negative, zero and positive numbers? (Circle one) **yes** **no** **maybe**
Can this strategy (as is with no modifications) be translated into an in-place algorithm? (Circle one)
yes **no**

Can this strategy (as is with no modifications) be translated into an on-line algorithm? (Circle one)

yes **no**

Considering the memory needed to store the input, output and any local variables, which of the two strategies strategy-1 and strategy-2 is more space-efficient? (Circle one)

strategy-2 **strategy-3** **both are equally space-efficient**

Considering the number of basic operations executed, which of the two strategies strategy-1 and strategy-2 is more time-efficient? (Circle one)

strategy-2 **strategy-3** **both are equally time-efficient**

Is it possible to modify strategy-3 so it can be turned into an **in-place** algorithm? (Circle one)

yes **no**

4. Strategy efficiency (4 points)

Two versions of a strategy to solve the "10 largest trades on a given day in the NYSE" problem are given below:

Strategy a: Updating an unsorted output array:

a.1. Let B be an array of size 10.

a.2. Copy the first 10 trade values from A into B.

a.3. Scan B to locate the smallest trade value L in it.

a.4. Repeat steps a.4.1-a.4.3for each of the remaining (10 million –ten) trade values in A:

a.4.1 Get the next trade value from A and compare it with L.

a.4.2 If L is larger or equal, do nothing

a.4.3 Otherwise

a.4.3.1 Replace L in B with this next trade value from A.

a.4.3 2 Scan B to locate the smallest trade value L in it.

a.5 Output the trade values in B.

The second version maintains B in a sorted state so that the largest trade value in it could be directly located:

Strategy a': Updating a sorted output array:

a'.1. Let B be an array of size 10.

a'.2. Copy the first 10 trade values from A into B.

a'.3. sort B in ascending or descending order

a'.4. Repeat steps 3b.4.1-3b.4.2for each of the remaining (10 million –ten) trade values in A:

a'.4.1 Get the next trade value from A and compare it with the smallest value in B

a'.4.2 If the smallest value in B is larger or equal, do nothing

a'.4.3 Otherwise

a'.4.3.1 Replace the smallest value in B with this next trade value from A.

a'.4.3 2 sort B in the same order

a'.4. Output the trade values in B.

Which version is more efficient? Circle one: **Strategy a** **Strategy a'** **both are equally efficient**

Provide a technical justification or argument (short and precise) to support your claim:

Strategy A, because it does not require that the program must continuously re-sort itself. A simply scans which is more efficient than sorting, which would require both scanning the array and then sorting based off that scan. So A is more efficient than B.

5. The string search problem (6 points)

We discussed the "sliding window" strategy for string search in class, in which, if you are searching for a string S of length m in a text of length n, m≤n, you compare S with a m-length substring of T starting with its first character (at index 1), and if a match is not found sliding that m-length "window" one character to the right on T and comparing the m-length substring at indexes 2...(m+1) of T with S, and repeating this until the last m characters of T are compared with S (unless S is found in T earlier, in which case the search ends).

Given S and T, where |S|=m and |T|=n and m≤n, what is the **minimum** number of character-to-character comparisons that an algorithm implementing this strategy has do <u>to find S in T</u>? m

If S="cat" provide a T of length 6 for which this minimum number applies: catabc

Given S and T, where |S|=m and |T|=n and m≤n, what is the **maximum** number of character-to-character comparisons that an algorithm implementing this strategy has do <u>to find S in T</u>? n

If S="cat" provide a T of length 6 for which this maximum number applies: abacat

Given S and T, where |S|=m and |T|=n and m≤n, what is the **maximum** number of character-to-character comparisons that an algorithm implementing this strategy has do to <u>if S is not in T</u>? n

If S="cat" provide a T of length 6 for which this maximum number applies: ababab

6. Writing algorithms (10 points)

Convert Problem 3 Strategy 3 into an algorithm and write it below. remember to provide an algorithm header, number the algorithm steps and to use pseudocode conventions.

```
Strategy3 ( A: Array [1...n] of numbers)
1. x = y = z = Ø
2. for i = 1 to n
3.     if A[i] < Ø
4.         then x = x + 1
5.     else if A[i] == Ø
6.         then y = y + 1
7.     else if A[i] > Ø
8.         then z = z + 1
9. let B [1...n] be a new array
10. neg = 1
11. zero = x + 1
12. pos = x + y + 1
13. for j = 1 to n
14.     if A[j] < Ø
15.         then B[neg] = A[j]
16.         neg = neg + 1
17.     else if A[j] == Ø
18.         then B[zero] = A[j]
19.         zero = zero + 1
20,     else if A[j] > Ø
21.         then B[pos] = A[j]
22.         pos = pos + 1
23. return B
```

7. Importance of algorithm efficiency (5 points)

This is a variation on problem 1-1 from the text (p. 14-15). Suppose you have a computer that can execute $2*10^7$ computational steps per second. If there are five algorithms with efficiencies O(lgn) [lgn = log of n to the base 2], O(square-root(n)), O(n), O(n^2) and O(n^3), and you need the algorithm to produce an answer in no more than one second, what is the range of input sizes from 1 to n for which each of these algorithms will be able to produce an answer in at most one second?

O(lgn) algorithm for input sizes 1 to $2^{(2 \times 10^7)}$

O(square-root(n)) algorithm for input sizes 1 to $(2 \times 10^7)^2 = 4 \times 10^{14}$

O(n) algorithm for input sizes 1 to 2×10^7

O(n^2) algorithm for input sizes 1 to $\sqrt{2 \times 10^7} \approx 4472$

O(n^3) algorithm for input sizes 1 to $\sqrt[3]{2 \times 10^7} \approx 271$

8. Iterative algorithm understanding (13 points)

Mystery(y, z: positive integer)

```
x=0
while z > 0
        if z mod 2 ==1 then
                x = x + y
        y = 2y
        z = floor(z/2)            //floor is the rounding down operation
return x
```

Simulate this algorithm for y=4 and z=7 and answer the following questions:

(3 points) At the end of the first execution of the while loop, x= 4 , y= 8 and z= 3 .

(3 points) At the end of the second execution of the while loop, x= 12 , y= 16 and z= 1 .

(3 points) At the end of the third execution of the while loop, x= 28 , y= 32 and z= 0 .

(2 points) Value returned by the algorithm = 28

(2 points) What does this algorithm compute? Y x Z

9. Understanding and modifying algorithms: Online/Anytime Sorting (10 points)

An online/anytime sorting algorithm is one that reads its input numbers one at a time, and maintains a sorted array of all the inputs it has seen so far, so that if it is interrupted at any time, it will still output the correct answer for the inputs that it has processed. Not all sorting algorithms are amenable to modification to make them anytime. But one sorting algorithm, Bubble Sort, is easy to so modify.

First, understand the ascending order Bubble Sort algorithm below:

```
Bubble-sort (A: Array [1...n] of number)
    for i=1 to (n-1)
        for j=1 to (n-i)
            if A[j]>A[j+1] then
                temp=A[j]
                A[j]=A[j+1]
                A[j+1]=temp
```

(4 points) If input A=[12,5,11,6,10,7,9,8], what will A be after the **3rd** <u>iteration of the outermost for loop</u> of the Bubble-sort algorithm completes? **A=[**5, 6, 7, 9, 8, 10, 11, 12**]**

(6 points) Modify the algorithm above to obtain an online/anytime sorting algorithm. Assume that "quit" is a global Boolean that is set external to the algorithm indicating that it should terminate. Some parts are given. Fill in the blanks:

```
Anytime-Bubble-Sort ()
 Let A be an empty dynamic (variable length) array with a large enough capacity
array-length=0
if quit is false
    next= read next number from the input stream
    array-length= array-length+1
    A[array-length]=next
    for j=array-length-1 down to 1
        if A[j + 1]<A[j]
            swap A[j] and A[j+1]
        else
            exit the for loop
return A
```

10. Understanding and modifying algorithms: Selection problem (12 points)

First, understand the Selection-sort algorithm below:

```
Selection-sort(A: Array [1..n] of numbers)
    for i=n down to 2
        position=i
        for j=1 to (i–1)
```

```
            if A[j]>A[position] then position=j
      if position ≠ i then
            temp=A[i]
            A[i]=A[position]
            A[position]=temp
```

(4 points) If input A=[12,5,11,6,10,7,9,8], what will A be after the 3rd iteration of the outermost for loop of the Selection-sort algorithm completes? **A=[**8, 5, 9, 6, 7, 10, 11, 12**]**

(8 points) Modify the algorithm to solve the problem of finding the k-th largest number in array A, 1≤k≤n, without sorting the entire array. Parts of the algorithm are given below. Fill in the blanks.

```
Select-k-th-largest(A: Array [1..n] of numbers; k: integer, 1≤k≤n)
for i = n down to (n - k + 1)
      position = i
      for j = 1 to (i - 1)
            if A[j] > A[position] then position = j
      if position ≠ i then
            temp=A[i]
            A[i]=A[position]
            A[position]=temp
return A[n-k+1]
```

11. Recursive algorithm understanding (8 points)

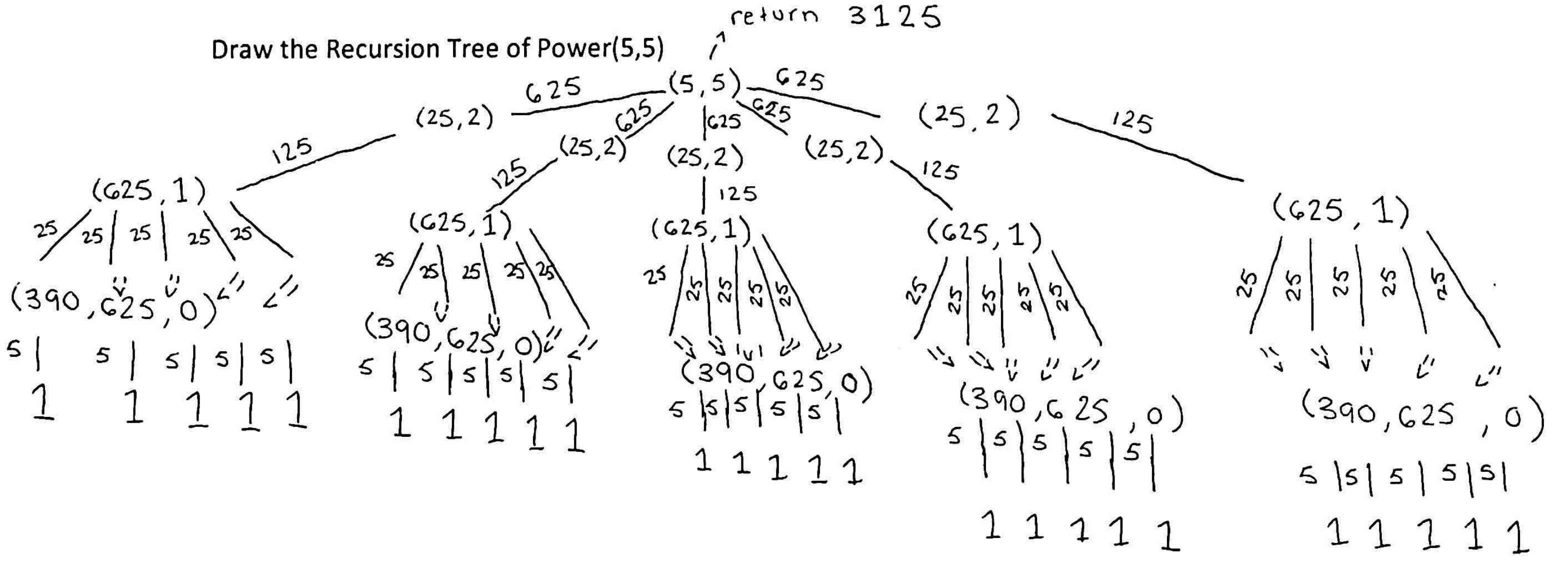
Power(y: number; z: non-negative integer)

1. if z==0 then return 1
2. if z is odd then
3. return (Power(y*y, z/2)*y) comment: z/2 is integer division; note the parentheses

else

4. return Power(y*y, z/2) comment: z/2 is integer division

Draw the Recursion Tree of Power(5,5)



12. Converting recursive algorithms to iterative algorithms (6 points)

If you understand how the above recursive algorithm to compute y^z works, you can turn it into a more efficient iterative algorithm that basically uses the same strategy (though it is not a tail recursive algorithm). Some parts of this iterative algorithm is given below. Fill in the blanks:

Power-iterative(y: number; z: non-negative integer)

1. answer=1
2. while z > 0
3. if z is odd then answer= answer * y
4. z = z / 2
5. y = y * y
6. return answer