/*

Jason Wilson

Thursday, March 5th 2020

CMPT435L 111 20S

*/

Divide and Conquer

Please read turn-in checklist at the end of this document before you start doing exercises.

Section 1: Pen-and-paper Exercises

1. Given an array A of n elements, find out the maximum difference between any two elements such that larger element appears after the smaller element in A. In other words, find a pair of elements A[p], A[q] with q>p such that (A[q] – A[p]) is the maximum among all such pairs in A.

   For example, if array is [2, 3, 10, 6, 4, 8, 1] then the maximum difference should be 8 (Diff between 10 and 2).

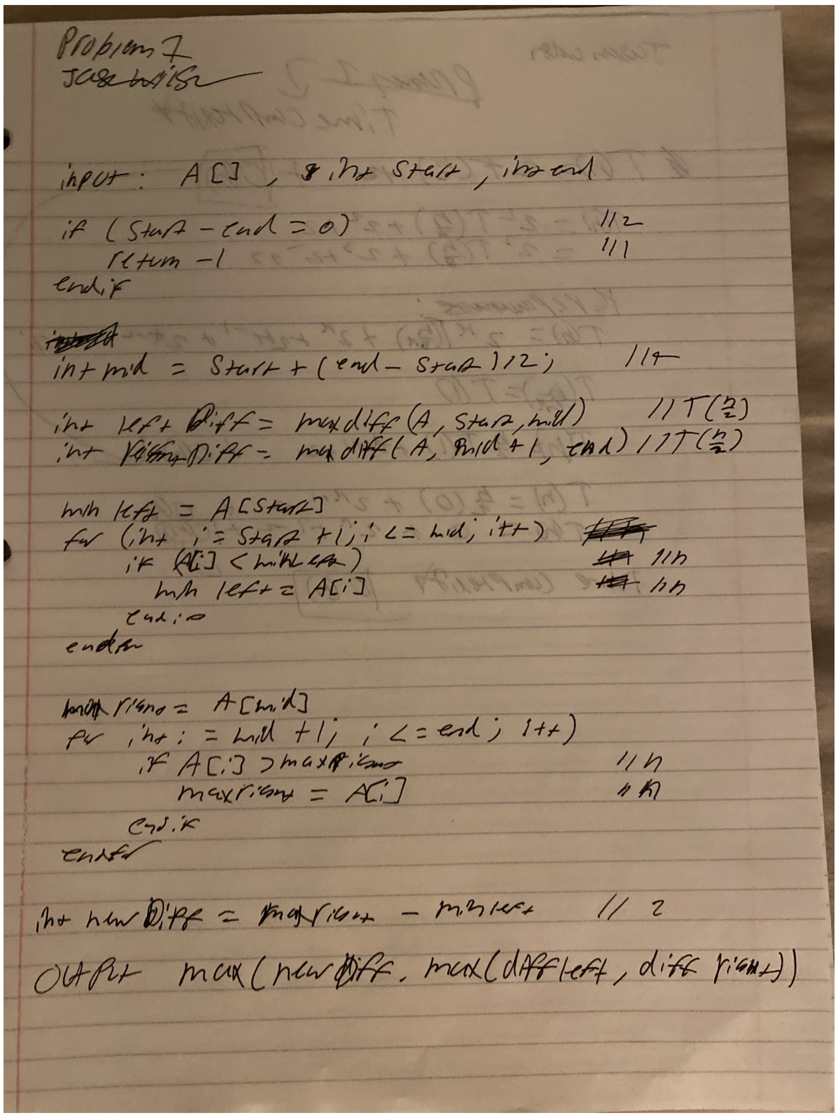
   If array is [7, 9, 1, 6, 3, 2] then the maximum difference should be 5 (Diff between 1 and 6).

   Design a **divide-and-conquer algorithm** of O(n) to solve this problem.

   (i) describe the idea behind your algorithm in English (2 points);

   1. If the size of the array = 0 then return - 1 , base case
   2. Divide the array into two halves, left and right and call the function recursively - general
   3. Get the min element from the left half and the lax element from the right half and compare with the left and right min and max difference
   4. Return the max difference

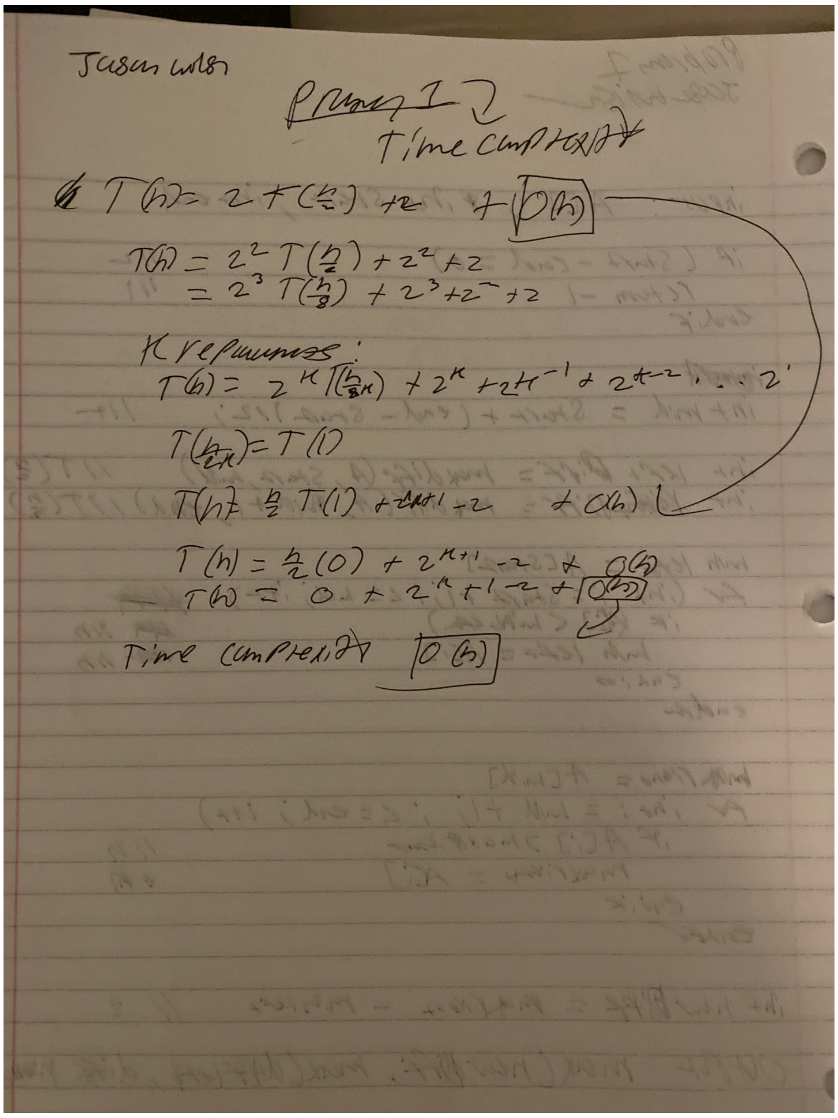
(ii) provide pseudocode (5 points);



Problem 7
Jase [illegible]

```
input: A[], int start, int end

if (start - end = 0)                          //2
    return -1                                 //1
endif

int mid = start + (end - start)/2;            //4

int leftDiff = maxdiff(A, start, mid)         //T($\frac{n}{2}$)
int rightDiff = maxdiff(A, mid+1, end)        //T($\frac{n}{2}$)

min left = A[start]
for (int i = start+1; i <= mid; i++)
    if (A[i] < minleft)                       //n
        min left = A[i]                       //n
    endif
endfor

max right = A[mid]
for int i = mid+1; i <= end; i++)
    if A[i] > maxright                        //n
        maxright = A[i]                       //n
    endif
endfor

int newDiff = maxright - minleft              // 2

Output max(newDiff, max(diffleft, diffright))
```

(iii) analyze its running time (3 points).



Jason Walsh

Problem 1

Time complexity

$T(n) = 2T(\frac{n}{2}) + 2 \quad + \boxed{O(n)}$

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

$= 2^3 T(\frac{n}{8}) + 2^3 + 2^2 + 2$

K replacements:

$T(n) = 2^k T(\frac{n}{2^k}) + 2^k + 2^{k-1} + 2^{k-2} \dots 2^1$

$T(\frac{n}{2^k}) = T(1)$

$T(n) = \frac{n}{2} T(1) + 2^{k+1} - 2 \quad + O(n)$

$T(n) = \frac{n}{2}(0) + 2^{k+1} - 2 + O(n)$

$T(n) = 0 + 2^{k+1} - 2 + \boxed{O(n)}$

Time complexity $\boxed{O(n)}$

Regarding requirement (iii): Unless otherwise specified, show the steps of your analysis and present your result using big-O.

Note: Full credit (10 points) will be awarded for a divide-and-conquer algorithm that is O(n). Algorithms that are NOT divide-and-conquer or slower than O(n) will be scored out of 5 points.

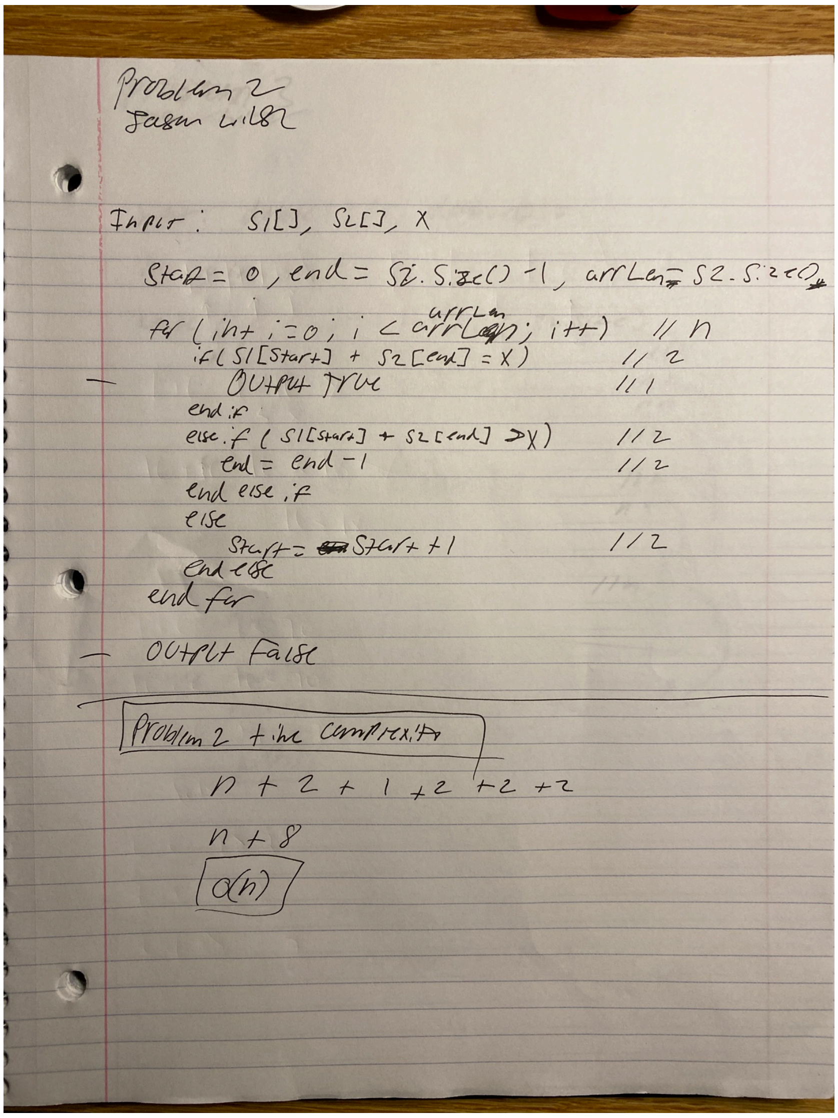
2. Consider the following problem:
   Let S1 and S2 be lists of elements. Each contains n elements in an increasing order.

For a given x, the problem is to find whether there exist an element in S1 and an element in S2 whose sum is exactly equal to x. Give an algorithm that solves this problem in O(n) time in the worst case (10 points).

(i) describe the idea behind your algorithm in English (2 points);

1. If the sum of the elements at indexes s1[start] and s2[end] equal return
2. Else if the sum of the numbers is greater than x, decrease the index of the second array
3. Else If the sum is less than x, increase the index of the first array

(ii) provide pseudocode (5 points); and (iii) analyze its running time (3 points).



Problem 2
Jason Wilsz

```
Input:  S1[], S2[], X

  Start = 0, end = S2.Size() - 1, arrLen = S2.Size()

  for ( int i = 0; i < arrLen; i++)                // n
      if( S1[Start] + S2[end] = X )                // 2
          Output True                              // 1
      end if
      else if ( S1[Start] + S2[end] > X)           // 2
          end = end - 1                            // 2
      end else if
      else
          Start = Start + 1                        // 2
      end else
  end for

  Output False
```

Problem 2 time complexity

$n + 2 + 1 + 2 + 2 + 2$

$n + 8$

$\boxed{O(n)}$

Regarding requirement (iii): Unless otherwise specified, show the steps of your analysis and present your result using big-O.

Note: Algorithms that are O(nlogn) or slower will be scored out of 5 points.

3. Consider the following problem:
Input: Two **sorted** arrays, A, B.

A and B together contain **n** integers.

Output: Identify the elements that appear in both A and B. If an element appears in A AND also in B, it should appear in the output.

For example, if A and B are:

{0, 0, 0, 1, 2, 3, 97, 98}

{0, 1, 2, 3, 4, 4, 10, 98, 100, 100}

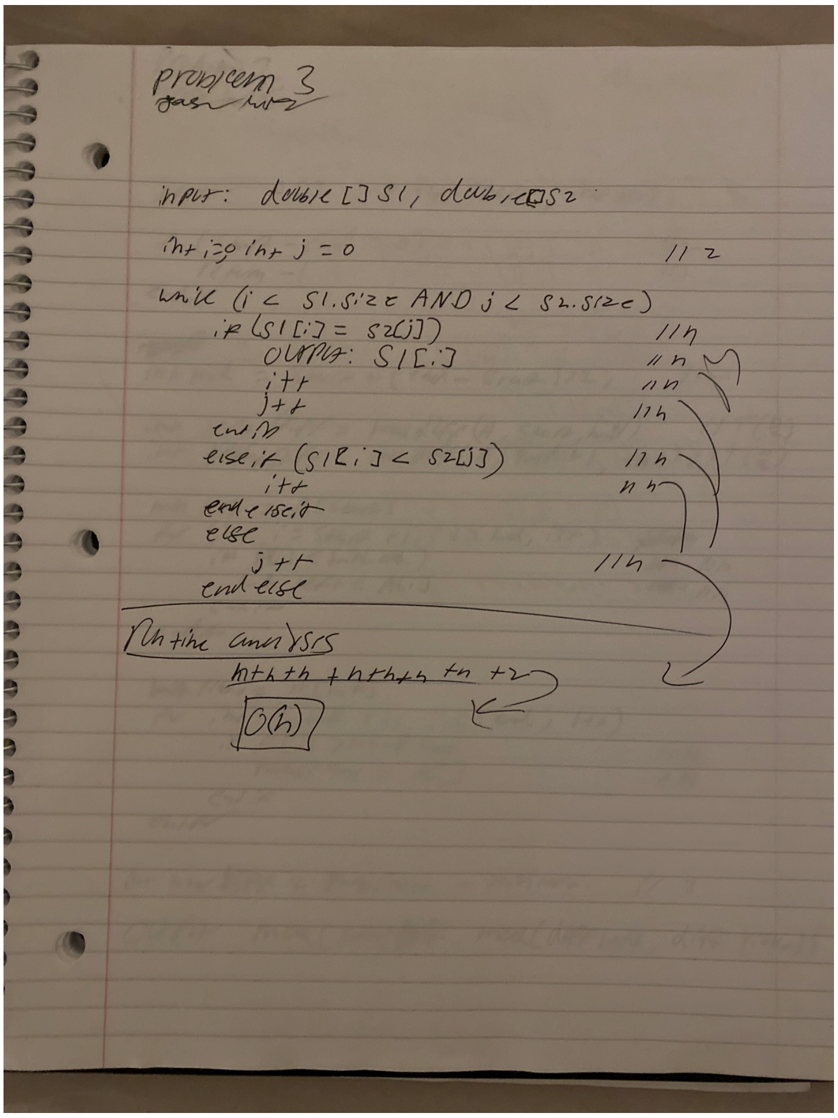
The output should be {0, 1, 2, 3, 98}, as these elements appears in both A and B.

Give an algorithm that solves this problem in O(n) time in the worst case (10 points).

(i) describe the idea behind your algorithm in English (2 points);

1. Iterate throught both arrays until each have been searched through entirely
2. If s1 = s2 then output s1[i]
3. Else if //s2 is bigger than s1 increase the s1 search index by 1
4. Else increase s2 search index by 1

(ii) provide pseudocode (5 points) and (iii) analyze its running time (3 points):



problem 3

[illegible]

```
input: double[] S1, double[] S2

int i=0 int j = 0                          // 2

while (i < S1.size AND j < S2.size)
    if (S1[i] = S2[j])                     // n
        output: S1[i]                      // n
        i++                                // n
        j++                                // n
    end if
    else if (S1[i] < S2[j])                // n
        i++                                // n
    end else if
    else
        j++                                // n
    end else
```

Run time analysis

$n+n+n+n+n+n+n+n+2$

$\boxed{O(n)}$

Regarding requirement (iii): Unless otherwise specified, show the steps of your analysis and present your result using big-O.

Note: Algorithms that are O(nlogn) or slower will be scored out of 5 points.

4. Trace Merge Sort on the following array:
   {1, 7, 6, 8, 0, 2, 5}

   Note: Write down the state of the array after each recursive call. (5 points).

```
                                 1, 7, 6, 8, 0, 2, 5

        1, 7, 6, 8                                0, 2, 5

     1, 7        6, 8

     1   7                                                                Merge

      1, 7

                 6       8                                                  Merge

                         6, 8

                   1, 7         6, 8                                        Merge

                 1, 6, 7, 8

                                                 0, 2           5

                                               0      2                     Merge

                                                 0, 2

                                                 0, 2           5           Merge

                                                      0, 2, 5

                    1, 6, 7, 8                        0, 2, 5               Merge

                                 0, 1, 2, 5, 6, 7, 8
```

Section 2: Java Implementation

5. Implement problem 1 in Java (30 points).
   Note:

   Find a file called Problem1.java in the folder.

   Complete the method of findmaxdiff().

   Test your method in the main method provided following the comments.

   Full credit (30 points) will be awarded for a divide-and-conquer algorithm that is O(n). Algorithms that are NOT divide-and-conquer or slower than O(n) will be scored out of 10 points.

6. Implement problem 2 in Java (30 points).
   Note:

   Find a file called Problem2.java in assignment 10 folder.

   Complete the method of checksum().

   Test your method in the main method provided following the comments.

   Full credit (30 points) will be awarded for an algorithm that is O(n). Algorithms that are O(nlogn) or slower will be scored out of 10 points.

7. Implement problem 3 in Java (30 points).
   Note:

   Find a file called Problem3.java in assignment 10 folder.

   Complete the method of commonelements().

   Test your method in the main method provided following the comments.

   Full credit (30 points) will be awarded for an algorithm that is O(n). Algorithms that are O(nlogn) or slower will be scored out of 10 points.

<u>TURN-IN CHECKLIST:</u>

1. **Answers to Section 1 (.doc/.txt), and to Section 2 (all your source Code (.java files)). Remember to include your name, the date, and the course number in comments near the beginning of your code/report.**

2. **Create a folder and name it 'FirstName_LastName_assignment_5'. In the newly created folder copy and paste your files (.doc/.txt/.java files). Then compress the folder, and submit to iLearn.**