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);

For every increment of the for loop either the max difference and/or minimum element will be updated. The value of the max difference by the end of the loop will fill the requirement of the question. The minimum element will get updated to the current element if the current element is less than the previous min element.

(ii) provide pseudocode (5 points);

maxDiff = A[1] - A[0];

minElem = A[0];

Asize = A.length;

for (i = 1; i < Asize; i++)

{

if (A[i] - minElem > maxDiff)

maxDiff = A[i] - minElem;

if (A[i] < minElem)

minElem = A[i];

}

return maxDiff;

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

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);

Basically, every value in the first array will be added to every value in the second array and compared to the test value to see if the sum equals the target value. Adjust the element position in the array accordingly.

(ii) provide pseudocode (5 points);

i = 0;

j = s2.length-1;

while(i<s1.length && j>=0) {

if(s1[i]+s2[j]==x) {

return true;

} else if(s1[i]+s2[j]<x) {

i++;

} else if(s1[i]+s2[j]>x) {

j--;

}

}

return false;

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

O(nlogn)

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).

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

Increment the according position of the array depending one which element is higher and lower or if they are equal, eventually comparing each number from one array to the other and returning each value that appears at least one time in both arrays.

(ii) provide pseudocode (5 points);

i, j = 0;

// either increment one array, the other array, or both if the element in each is equal.

while(i < s1.length && j < s2.length) {

if(i != 0 && s1[i] == s1[i-1]) {

i++;

} else if (j != 0 && s2[j] == s2[j-1]) {

j++;

} else if (s1[i] == s2[j]) {

System.out.print(s1[i] + " ");

i++;

j++;

} else if (s1[i]>s2[j]) {

j++;

} else {

i++;

}

}

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

O(nlogn)

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}

* 1768025
* 1768 | 025
* 17|68 | 025
* 1|7|68 |
* 17|68 |
* 17|6|8 |
* 17|68 |
* 16|78 |
* 1678 | 02|5
* 1678 | 0|2 |5
* 1678 | 02|5
* 1678 | 025
* 0125678

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

**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.**

**TURN-IN CHECKLIST:**

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.**

1. **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.**