Matt Ronan 3/2/20

Algorithms Assignment 5

Section 1 Question 1:

English: To find the max difference where the subtractor is a smaller index than the subtracted, we must divide the array in two and find the min and max on each side then compare the index and size of each element and determine which of the mins is the smallest and which is behind the larger max and return their difference.

Pseudocode:

**public** **static** MinMaxDiff findmaxdiff (**double**[] A)

{

MinMaxDiff maxDiff = **new** MinMaxDiff();

**if**(A.length == 1) {

maxDiff.min = A[0];

maxDiff.max = A[0];

maxDiff.diff = 0;

}

**else** **if**(A.length == 2) {

**if**(A[0] < A[1]) {

maxDiff.min = A[0];

maxDiff.max = A[1];

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

}

**else** {

maxDiff.min = A[1];

maxDiff.max = A[0];

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

}

}

**else** {

**int** length = A.length;

**double**[] L = Arrays.*copyOfRange*(A, 0, (length + 1)/2);

**double**[] R = Arrays.*copyOfRange*(A, ((length + 1)/2), length);

MinMaxDiff left = **new** MinMaxDiff();

MinMaxDiff right = **new** MinMaxDiff();

left = *findmaxdiff*(L);

right = *findmaxdiff*(R);

**double** otherDiff = right.max - left.min;

**if**(left.max > right.max) {

maxDiff.max = left.max;

}

**else** {

maxDiff.max = right.max;

}

**if**(left.min < right.min) {

maxDiff.min = left.min;

}

**else** {

maxDiff.min = right.min;

}

**if**(left.diff > right.diff) {

maxDiff.diff = left.diff;

}

**else** {

maxDiff.diff = right.diff;

}

**if**(otherDiff > maxDiff.diff) {

maxDiff.diff = otherDiff;

}

}

**return** maxDiff;

}

Running Time: To get the min and max of both sides you must loop through half of the array 4 times

4 \* (n/2) = 2n = O(n)

Section 1 Question 2:

English: To find if elements of A and B add up to input x we loop through both arrays at once keep track of their position. If we find that the sum of two elements is equal to x we return true. Otherwise we check to see which element is greater A or B and increment the position of the lesser element. Once the elements of the arrays are exhausted the method returns false.

Pseudocode:

**int** i = 0;

**int** j = 0;

**double** answer = 0;

**while**(i < A.length && j < B.length) {

**if**(A[i] + B[j] == x) {

**return** **true**;

}

**else** **if**(A[i] > B[j]) {

j++;

}

**else** {

i++;

}

}

**return** **false**;

Running Time: The worst case this algorithm will run n + m times. N being the length of array A, m being the length of array B. If we consider them to be very large numbers close to infinity, we can consider them as equals and say n + n = 2n = O(n)

Section 1 Question 3:

English: To find common elements of A and B we use a while loop to loop through both arrays keeping track of their position. If the elements match we put the element in array C and increment all of the arrays and keep track of the previous element to make sure there are no repeats. If A > B at the position we only increment B’s position otherwise we increment A’s position

Pseudocode:

Int I = j = k = 0;

Int Prev;

While(I < A.size && j < B.size)

If(A[i] == B[j] && A[i] != prev)

C[k] = A[i];

K++; i++; j++; prev = C[k];

Else if(A[i] > B[j])

J++;

Else

I++

End if

End while

Running Time: The worst case this algorithm will run n + m times. N being the length of array A, m being the length of array B. If we consider them to be very large numbers close to infinity, we can consider them as equals and say n + n = 2n = O(n)

Section 1 Question 4:

Pivot = element

1. {1, 7, 6, 8, 0, 2, 5}
2. {0}, {1}, {7,6,8,2,5}
3. {0}, {1}, {6,2,5}, {7}, {8}
4. {0}, {1}, {2,5}, {6}, {7}, {8}
5. {0}, {1}, {2}, {5}, {6}, {7}, {8}
6. {0,1,2,5,6,7,8}