

ANSWER: Consider the following given recurrence relation: T(n) = T([n/2]upper) + T([n/2]lower) + cn

In the substitution method a solution is guessed every time and checked whether its working or not.

Now solve the recurrence relation using substitution method, by making a guess that T(n) = O(cnlog2n).

Since substitution method is using, it is required to prove that T(n) <= cnlog2n for C > 0.

Thus, T([n/2]upper) <= c([n/2]upper)log2([n/2]upper), same for the T([n/2]lower).

Then, T(n) <= c([n/2]upper)log2([n/2]upper)) + c([n/2]lower)log2([n/2]lower)) + cn

=(cn/2c)log2(n/2) + (cn/2c)log2(n/2) + cn

=2(cn/2c)log2(n/2) + cn = nlog2(n/2) + cn which is ~ to cnlog2n

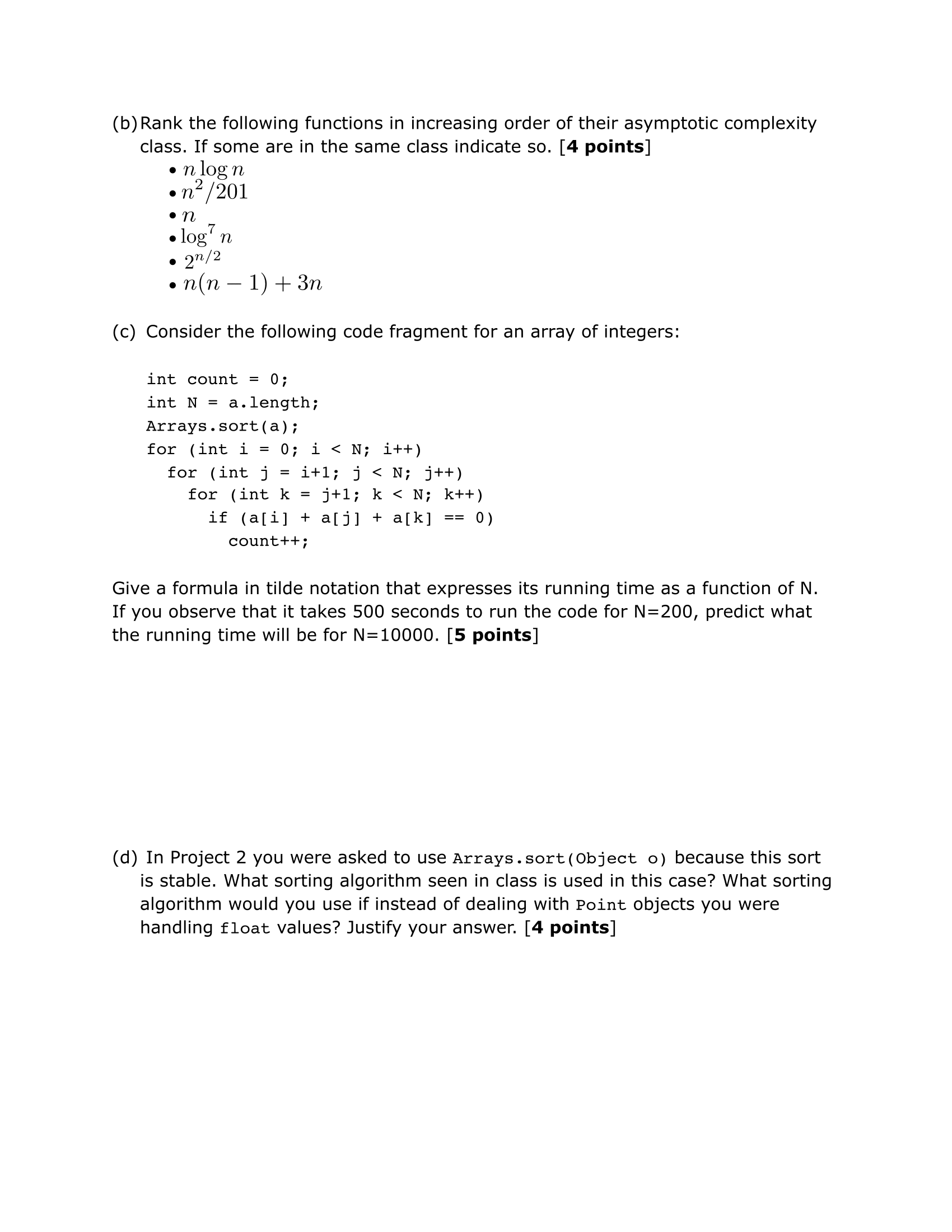
√

√

√

X

X



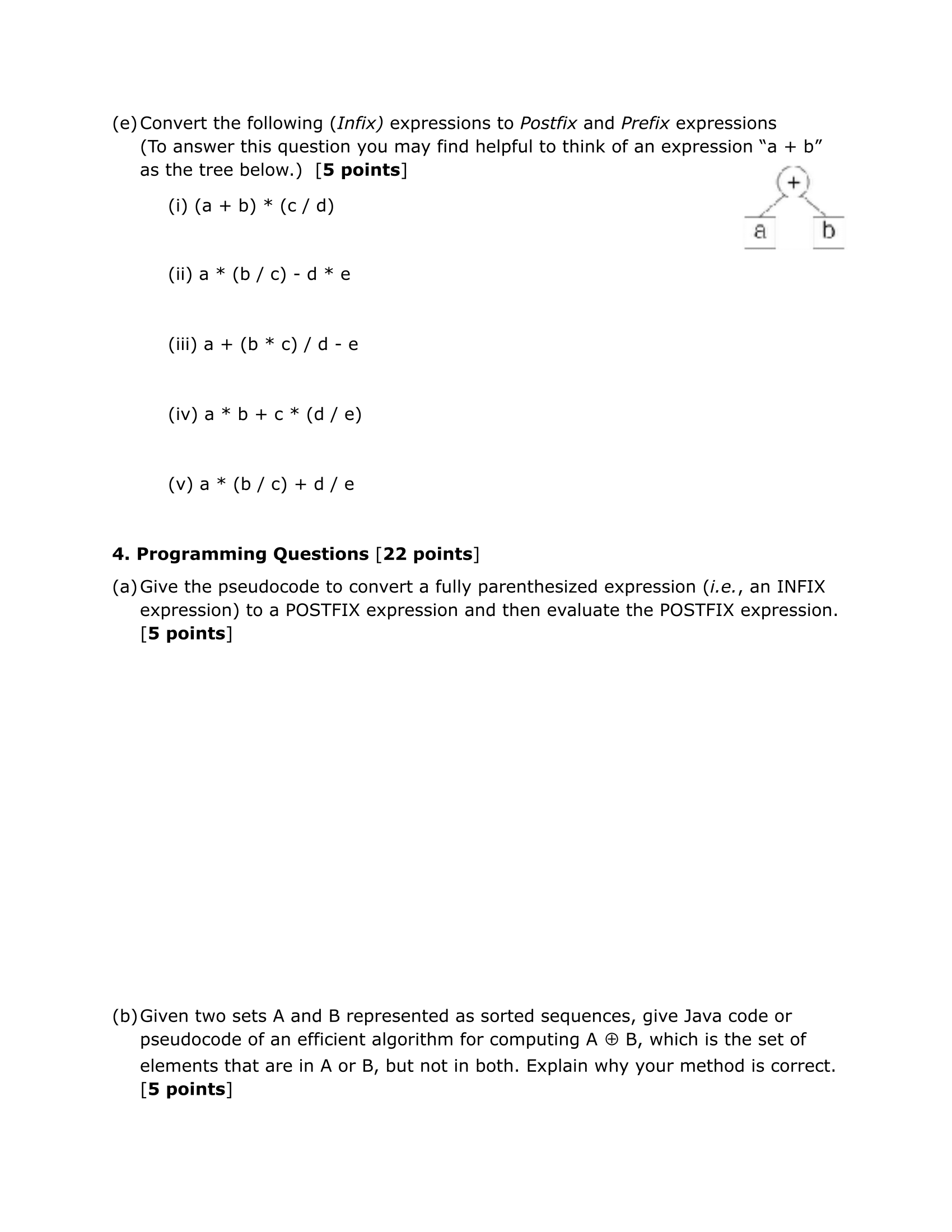
ANSWER: Array.sort uses MergeSort. For sorting float values, I would prefer Radix sort, because it works in O(n) and use its property of IEEE floats being sorted when their bit pattern is interpreted as ints, in another word, as sign-magnitude integers rather than two’s complement integers.

Another sort I would use is a combination of Heapsort and quicksort.

ANSWER T(N) = N^3. If 500 seconds to run N = 200, if N = 10000, 6.25 \* 10^-5 \* (N^3) = 6.25 \* 10^-5 \* 10000^3 = 62500000 seconds.

ANSWER:

Log^7n < n < nlogn < n^2/201 < n(n-1) +3n < 2^n/2



ANSWER: Postfix:b c / a \* d e / + Prefix + \* a / b c / d e

ANSWER: Postfix:a b \* d e / c \* + Prefix + \* a b \* c / d e

ANSWER: Postfix:a b c \* d / + e – Prefix - + a / \* d b c e

ANSWER: Postfix: a b c / \* d e \* - Prefix: - \* a / b c \* d e

ANSWER: Postfix: a b + c d / \*, Prefix: \* + a b / c d

ANSWER: My method is build on mathematical methord A ⊕ B = ( A – B) ∪ ( B – A ) or = ( A ∪ B) – (A ∩ B). The pseudocode is as follow:

For ( loop through A. length as i increases) { for(loop through B. length as j increases) { Array “arraySame” will store all the elements that are same from A[i] with B[j]} NEXT PAGE

ANSWER:

For ( each char c in the in fix expression) {

Switch(c) {

Case operand: postfix exp = postfixexp + c; break;

Case ‘(‘: astack.push(c); break;

Case ‘ )’ : while( top of stack is not ‘(‘) { postfixexp = postfixexp + (top of aStack); aStack.pop(); } astack.pop; break;

Case operator: while (!astack.isempty() && top of stack is not ‘(‘ && precedence(c) <= precedence(top of astack) { postfixecp = postfixexp + (top of stack); astack.pop();} astack,push©; break;

}}}

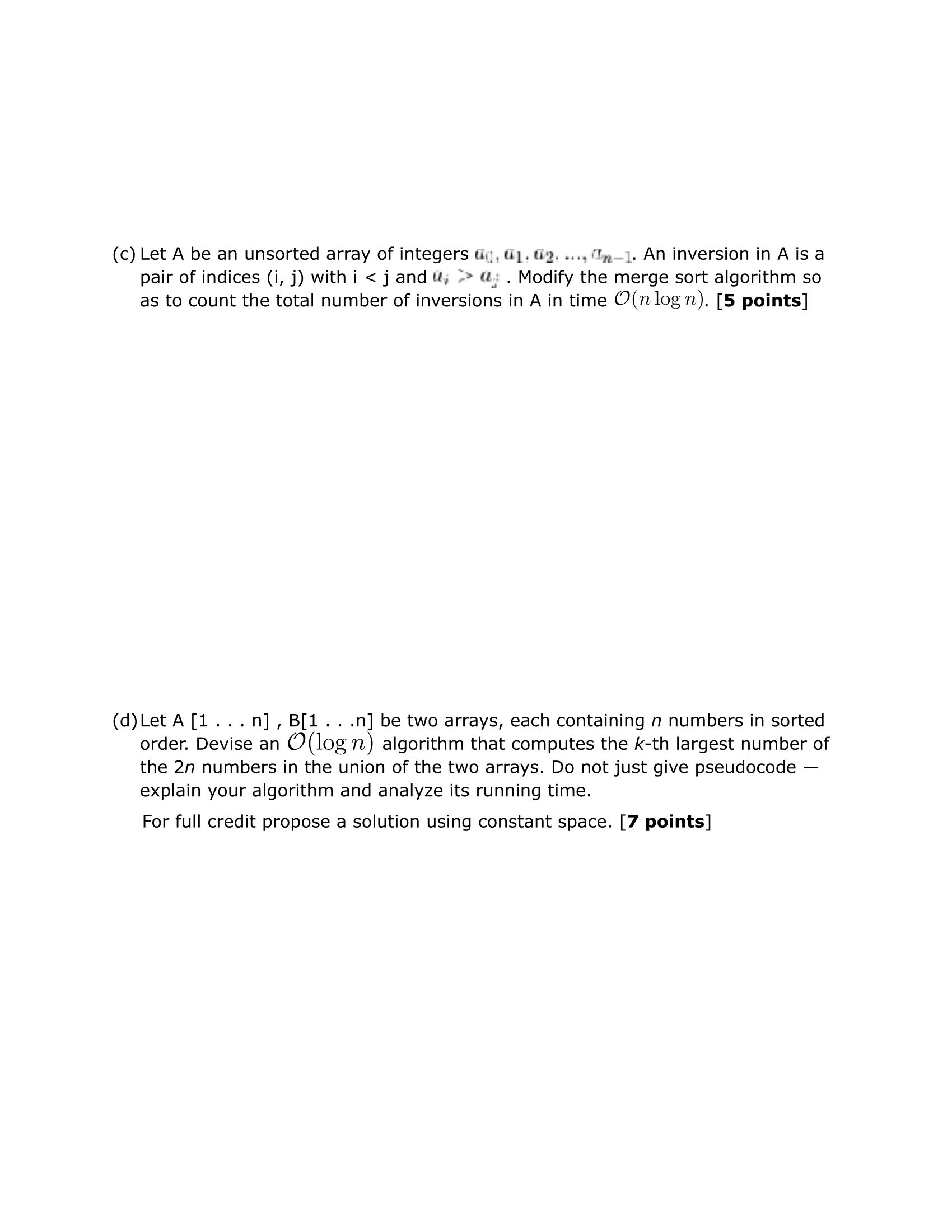
For ( loop through A. length as i increases) { for(loop through B. length as j increases) { Array “arrayDif” will store all the elements that are not the same from A[i] with B[j] }

Then Loop through arraySame and ArrayDif

If element arraysame and ArrayDif has a element that are only exist in either of them

Then store in arrayResult

Return ArrayResult;



ANSWER public static int mergeSort(int[] a, int start, int end, int[] aux) {

if (start >= end) {

return 0;

}

int invCount = 0;

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

int invCountLeft = mergeSort(a, start, mid, aux); // divide and conquer

int invCountRight = mergeSort(a, mid + 1, end, aux); // divide and conquer

invCount += (invCountLeft + invCountRight);

for (int i = start; i <= end; i++) {

aux[i] = a[i];

}

int left = start;

int right = mid + 1;

int index = start;

while (left <= mid && right <= end) {

if (aux[left] < aux[right]) {

a[index++] = aux[left++]; } else { a[index++] = aux[right++]; invCount += mid - left + 1; }}

while (left <= mid) { a[index++] = aux[left++];} return invCount;

ANSWER

ANSWER