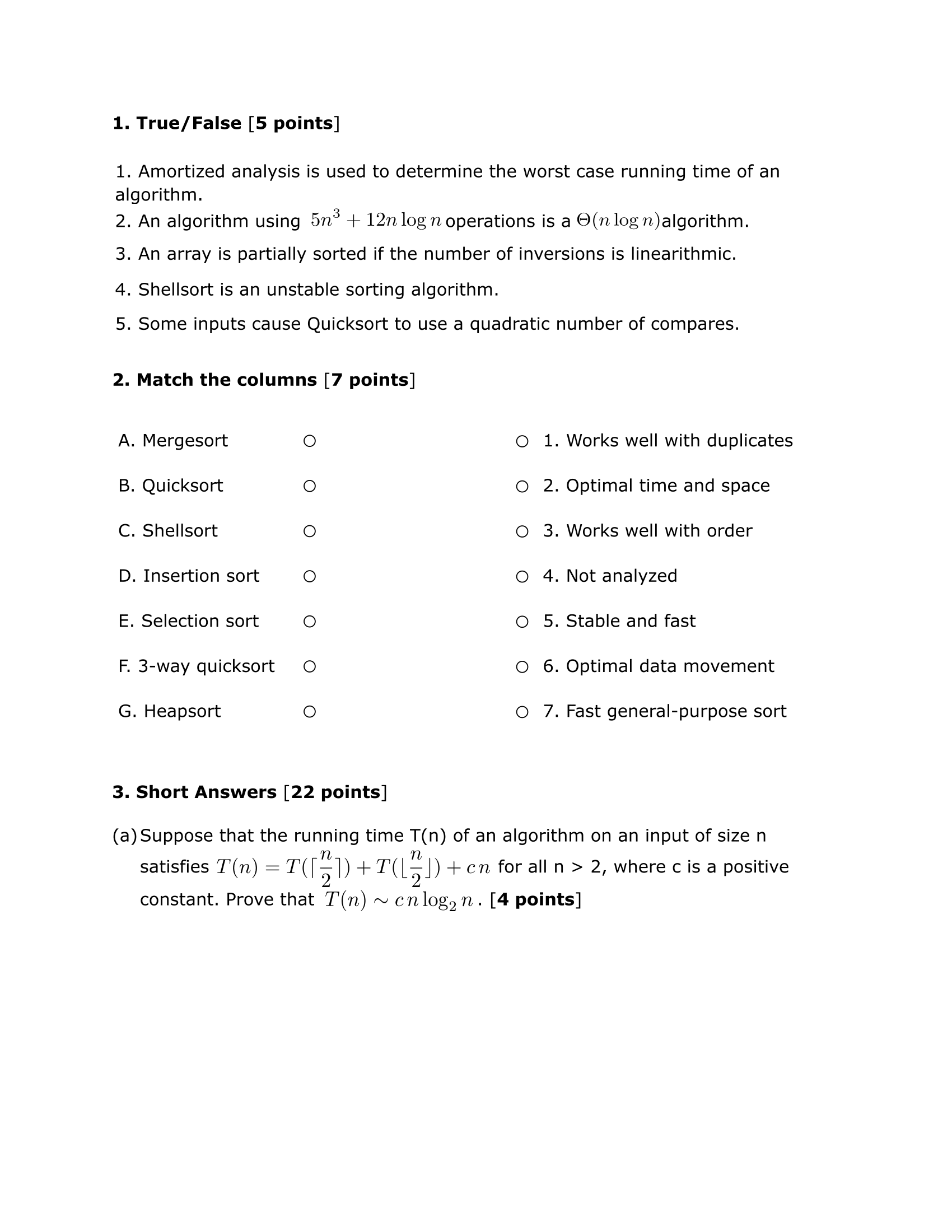


F

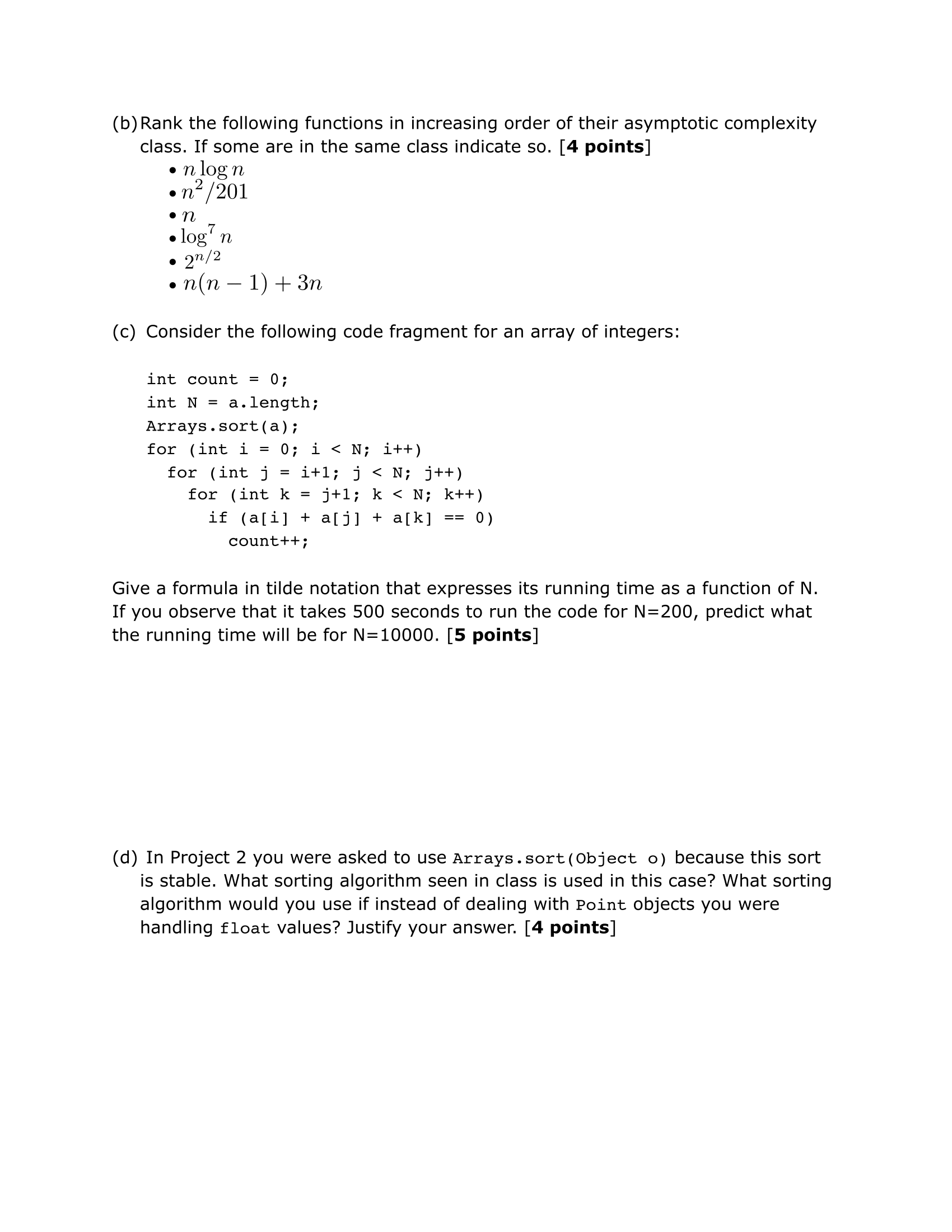


F

T

T

F



1. Same as
2. Best Complexity
3. Worst Complexity
4. Same as

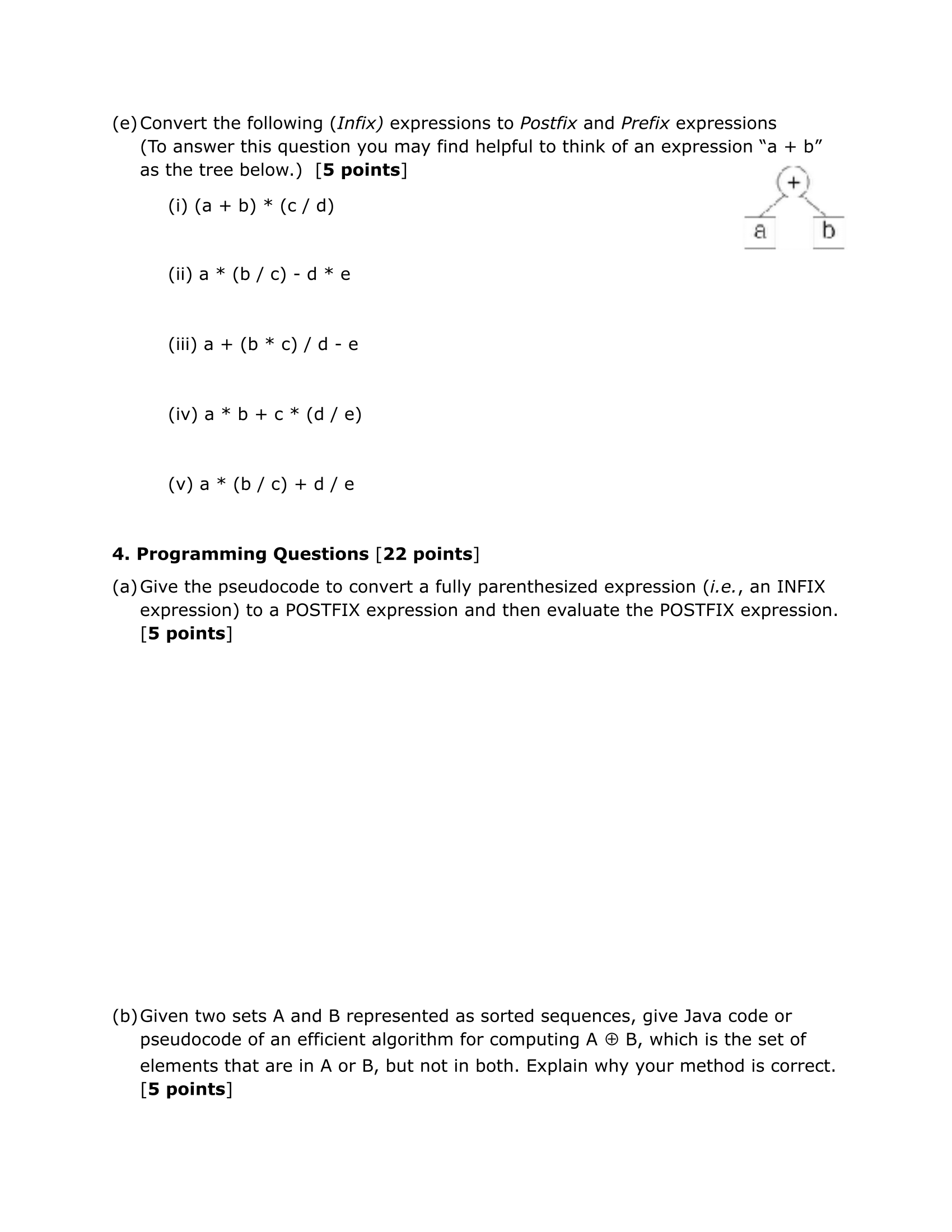
=

Using n=10,000 with a constant of s would take about 1,250,000 seconds or 14.47 days.

ANSWER

Mergesort is used for Arrays.sort(Object) since its stability is required when dealing with objects. The only other stable sort, Insertion sort, should only be used for a small N value whereas Mergesort guarantees Nlog(n) performance.

If handling float values, or some other primitive type where stability is not as much of a concern, you would use quick sort which is the fastest sort in practice.



**function** toPostfix**(char[]** s**)**

**Stack** stack**;**

**StringBuilder** postfix**;**

**for** (i **→** s.length **where** c **:** s[i])

**if** (c **is** operator)

postfix += c;

**while** (**importance**(c) <=

**importance**(stack.top))

postfix += **pop**(stack);

**push**(c);

**while** (stack not empty)

postfix += **pop**(stack);

**return** postfix;

**function** evalPostfix(string s)

**Stack** stack;

**for** (i **→** s.length **where** c **:** s[i])

**if** (c **is** operator)

**var** obj1 = **pop**(stack);

**var** obj2 = **pop**(stack);

**var** obj3 = **operation**(obj1,

obj2,

c);

**push**(obj3);

**else**

**push**(c)

**print** **pop**(stack);

Postfix: a b \* c + (d e /) \*

Prefix: \* ab + c \* (/ d e)

Postfix: a (b c /) \* d e / +

Prefix: + / a (/ b c) \* de

Postfix: a (b c \*) d / + e -

Prefix: - / + a (\* b c) d e

Postfix: (b c /) a \* (d e \*) -

Prefix: - \* (/ b c) a (\* d e)

Postfix (RPN): (a b +) (c d /) \*

Prefix: \* (+ a b) (/ c d)

**function unionMinusIntersection** (**Obj[]** a**, Obj[]** b)

Dictionary<int, Boolean> stored = new Dictionary(a.length

+ b. length);

removeDuplicates(a);

removeDuplicates(b);

for (i **→** a.length **where** x **:** a[i])

if (stored.contains(x))

stored.add(x, true)

else

stored.add(x, false)

for (i **→** b.length **where** x **:** b[i])

if (stored.contains(x))

stored.add(x, true)

else

stored.add(x, false)

for (KeyValuePair<int, Boolean> pair : stored)

if (pair.value == false)

print pair.key

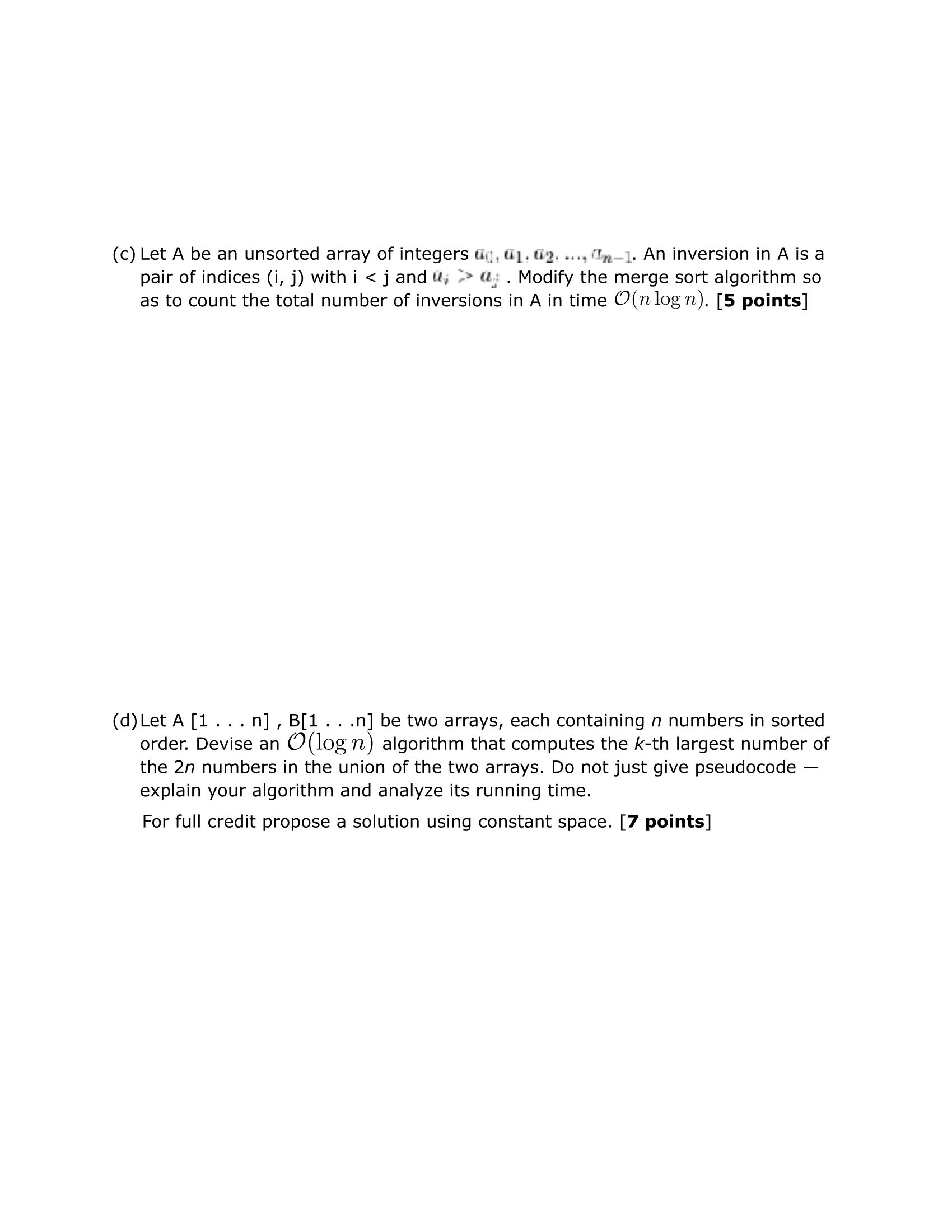
ANSWER TO (b)

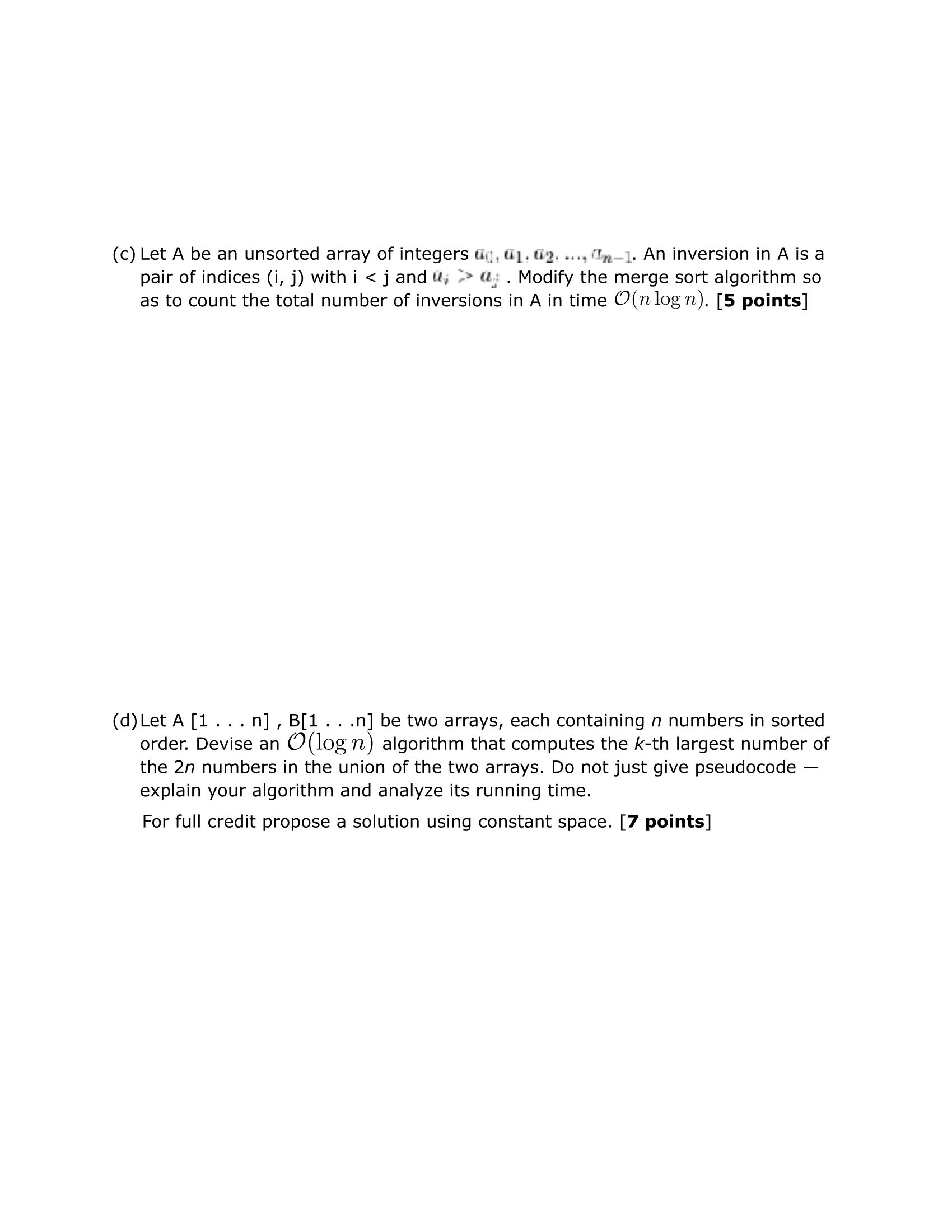
JUSTIFICATION

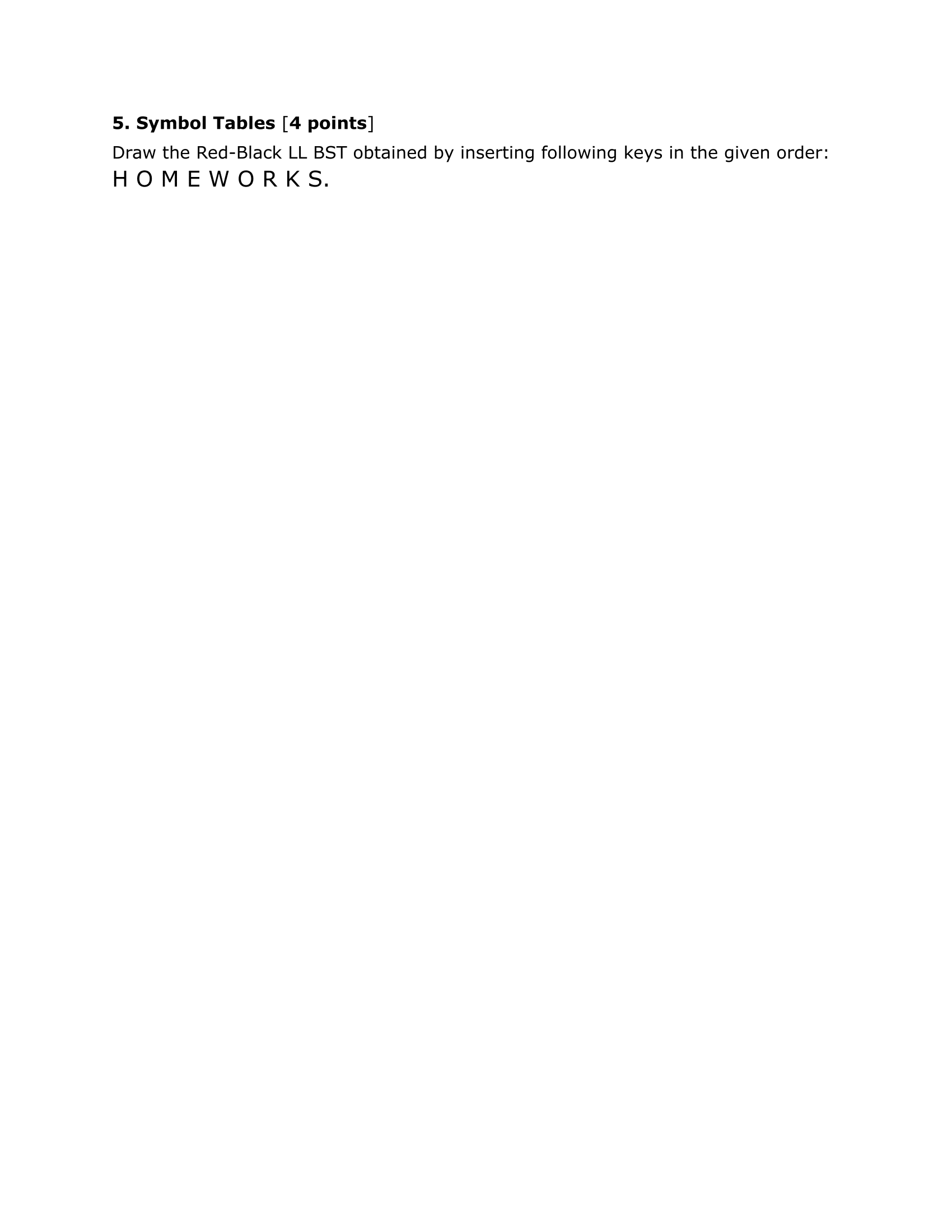
The above algorithm keeps track of whether a value is stored in A or B by simply updating a Boolean value indicating whether or not It’s a duplicate value. If it’s a duplicate value, specifically both in A and B, we ignore reading it entirely. Thus producing an output free of duplicates of any kind.

**function**

**function**







Assuming that duplicate keys are not allowed