Not sure about any of these

1a)

Condition = if (y > 0 && x < 0)

Signed integer overflow is undefined

∆(x) = (true -> ¬(INT\_MAX – 2 + y ∉ [-2^n-1, 2^(n-1)-1]))

= INT\_MAX – 2 + y ∈ [-2^n-1, 2^(n-1)-1]

Elimination query = (y > 0 ∧ x < 0) ∧ ∆(x)

Alternate answer?

Condition = if (y > 2)

Because signed integer overflow is undefined, y > 2 would trigger signed integer overflow and in a well defined program, y <= 2 at this point.

Elimination query = (y > 2) and (true -> negate(y > 2))

Alternate answer #2?

Condition = if (x < 0)

Because signed integer overflow not defined, then optimized compiler assumes x will always be > 0 (right?)

Elimination query = (x < 0) and (true -> negate(x < 0))

Alternate answer #3?

Condition = if (x < y)

Elimination query = (x < y) and (true -> INT\_MAX – 2 + y ∉ [-2^n-1, 2^(n-1)-1]))

Alternate answer #4?

Condition = if (y > 2 )

Elimination query = (y > 2) and (true -> x ∉ [INT\_MIN, INT\_MAX]))

B)

I) length(argv) >= 2, argv[1] is numeric and within range of [INT\_MIN, INT\_MAX]

Ii) x = -2^(n-1) = INT\_MIN.

Iii) x = -2^n-1 ∧ (true -> ¬(x = -2^n-1))

Iv) Yes because it is trivially unreachable.

V) No because it is trivially unreachable.

Vi) x != INT\_MIN

Vii) (-x < 0 & x < 0) ∧ x != INT\_MIN

Viii) Yes because the bit that makes it trivially unreachable is outside the function. With the approximation, Stack only looks at it from the start of the function and so the other conditions are missed and this is unstable code.

2)

A) I) yes ii) yes iii) no iv) no

B)

I) General – is the array index in range (needs to hold for any program)

Ii) Functional – Is the algorithm correct (does if obey the defined spec)

Iii) Non-functional – Why is the performance so bad? (non-spec constraints e.g. power, performance)

C)

I)

char \*a = new char[100];

char \*b = new char[1000];

a[500] = 0;

Assuming a small redzone, this access will likely point into b and will completely skip over the redzone so Asan won’t be able to detect it. The J&K compiler on the other hand keeps track of the bounds of each array and will be able to capture it.

Ii) ASan – Just need to check if the address is in a poisoned area

J&K – Need to search through the object table and check if the address is within range

D)

I) No errors because it is legal to hold a pointer to one past and q is within range

Ii) The dereference of q is an error, as q = (a+200) - 201 = a-1 which is a buffer overflow

Iii) No J&K will not report an error because there is a 1 byte padding between objects to accommodate the one past the end pointer thing. So both p and q are valid. I think this is only the case here because a and b are both heap allocated.

Iv) J&K could throw an error here because arrays on the stack passed to a function is an edge case where there is no padding in between. A pointer 1 past the end is brought back in so it J&K would think you derived it from a pointer to b instead of a.

V) Yes J&K will report an error in h because q is derived from p which will be marked as invalid so q will also be marked as invalid.