# Background

In this problem, we will verify the correctness of operations that work with finite sets of real numbers stored in a sorted array. We will also include the number -∞ as the first element in the array and +∞ as the last element in the array. This removes some special cases that would otherwise need to be handled in the code.

In summary, the array S represents a set in this manner iff it satisfies the constraint that

$$-\infty = S[0] < S[1] < S[2] < \ldots < S[n] < S[n+1] = +\infty$$

where n is the number of elements in the set (which is 2 less than the length of S due to the extra elements at the front and back). This array represents the set {S[1], S[2], .., S[n]}.

In other words, the constraint above is the *representation invariant*, and for an array S satisfying this invariant, its *abstract value* is {S[1], S[2], .., S[n]}.

We will represent real numbers using Java's `float` type. In addition to being able to represent a wide range of numbers (e.g., $-10^{100}$, $+10^{100}$, and $1/10^{100}$), a `float` can also represent -∞ and +∞. Although we will not need it in the code below, there is also a special `float` value called NaN ("not a number"), which is produced for invalid operations. For example, the calculation 1.0 / 0.0 produces +∞, while the calculation 0.0 / 0.0 produces NaN.

In these problems, we will not require you to *show* every line of reasoning, only the most important ones. However, you will likely want to fill in the important assertions by working through the code line-by-line as before.

**Hints**: The following facts may be useful:

- The assertion "a < min(b, c)" is equivalent to (another way of writing) "a < b and a < c".
- The assertion "max(a, b) < c" is equivalent to "a < c and b < c".
- Together, "max(a, b) < min(c, d)" is equivalent to "a < c and a < d and b < c and b < d".
- These equivalences also hold with "<" replaced by "≤".

## Verifying Correctness of Union

Fill in the indicated assertions by reasoning in the direction indicated by the arrows. We will address the places where "?"s appear on the *page 4*.

Notation: we will use "n" and "m" to refer to the number of points in sets S and T, respectively.

```
  {{ Pre: -∞ = S[0] < S[1] < … < S[n] < S[n+1] = ∞ and -∞ = T[0] < T[1] < … < T[m] < T[m+1] = ∞ and
         U is an array containing at least n+m+2 elements }}
↓ U[0] = Float.NEGATIVE_INFINITY;
↓ int i = 1, j = 1, k = 1;

  {{ Pre and U[0] = -∞ and i=1 and j=1 and k=1 }}

  ? answer 1 on a separate page

  {{ Inv: Pre and -∞ = U[0] < U[1] < … < U[k-1] < min(S[i], T[j]) and
        {U[1], U[2], …, U[k-1]} = {S[1], S[2], …, S[i-1]} ∪ {T[1], T[2], …, T[j-1]} }}
  while (S[i] < Float.POSITIVE_INFINITY || T[j] < Float.POSITIVE_INFINITY) {
↓   if (S[i] < T[j]) {
      {{ Inv and S[i] < T[j] }}

      U[k] = S[i];  // ? answer 2 on the page 4

      {{ Pre and -∞ = U[0] < U[1] < … < U[k] < min(S[i+1], T[j]) and {U[1], …, U[k]}
        = {S[1], …, S[i]} ∪ {T[1], …, T[j-1]} }}
↑     i = i + 1;
↑     k = k + 1;
    } else if (S[i] > T[j]) {
      {{ Inv and S[i] > T[j] }}

      U[k] = T[j];  // ? answer 3 on the page 4

      {{ Pre and -∞ = U[0] < U[1] < … < U[k] < min{S[i], T[j+1]) and {U[1], …, U[k]}
        = {S[1], …, S[i-1]} ∪ {T[1], …, T[j]} }}
↑     j = j + 1;
↑     k = k + 1;
    } else {
      {{ Inv and S[i] = T[j] }}

      U[k] = S[i];  // ? answer 4 on the page 4

      {{ Pre and -∞ = U[0] < U[1] < … < U[k] < min{S[i+1], T[j+1]) and {U[1], …, U[k]}
        = {S[1], …, S[i]} ∪ {T[1], …, T[j]} }}

      // continued on the next page…
```

```
↑     i = i + 1;
↑     j = j + 1;
↑     k = k + 1;
    }
  }

↓ U[k] = Float.POSITIVE_INFINITY;
```

{{ Inv and S[i] = +∞ and T[j] = +∞ and U[k] = +∞ }}

? answer 5 on the next page

{{ Post: -∞ = U[0] < U[1] < … < U[k] = ∞ and
{U[1], U[2], …, U[k-1]} = {S[1], S[2], ..., S[n]} ∪ {T[1], T[2], …, T[m]} }}

## Explanations

Explain why the invariant holds initially (answer 1) and why the postcondition holds (answer 5):

1. Since $i=1$, $j=1$, $k=1$, we have $k-1=0 \Rightarrow -\infty = U[0] < \min\{S[i], T[j]\}$

   Since $S[i] > -\infty$ and $T[j] > -\infty$

   Also, we have $\{U[i], \dots, U[k-1]\} = \emptyset$, $\{S[i], \dots, S[i-1]\} = \emptyset$, $\{T[i], \dots, T[j-1]\} = \emptyset$

   $\emptyset = \emptyset \cup \emptyset$ holds.

5. From Inv and $U[k] = +\infty$ we have $-\infty = U[0] < U[1] < \dots < U[k] = +\infty$

   From $S[i] = +\infty$ and $T[j] = +\infty$ we know $i = n+1$ and $j = m+1$.

   Then plug them into inv and we have

   $\{U[1], U[2], \dots, U[k-1]\} = \{S[1], S[2], \dots, S[n]\} \cup \{T[1], \dots, T[m]\}$

## Comparing Assertions

The three "?"s in the middle of the loop (answers 2-4) appear next to an assignment and between two assertions that you filled in on the previous page. In each case, compare the two assertions and explain which *additional facts*, not listed in the top assertion, are needed to establish that the bottom assertion holds. (Do not yet worry about whether these facts hold.)

2. $U[k-1] < U[k] < \min\{S[i+1], T[j]\}$ and $U[k] = S[i]$

3. $U[k-1] < U[k] < \min\{S[i], T[j+1]\}$ and $U[k] = T[j]$

4. $U[k-1] < U[k] < \min\{[S[i+1], T[j+1]\}$ and $U[k] = S[i] = T[j]$

## More Explanations

For the three cases above, explain why the additional facts must hold from what we know in the top assertion and the assignment statement in between them.

Since these facts can not be known easily from the top assertions.

From previous courses. we know that when two assertions meet, the top assertions can lead to the below.

Only in this way can we prove that our code is correct.

Take answer 2 as an example:

we know $U[k-1] < \min\{S[i], T[j]\}$, $S[i] < T[j]$, $U[k] = S[i]$

then $U[k-1] < U[k] = S[i]$ holds.

From $S[i] < S[i+1]$, $S[i] < T[j] \Rightarrow S[i] < \min\{S[i+1], T[j]\}$

$\because S[i] = U[k] \quad \therefore U[k] < \min\{S[i+1], T[j]\}$

## Addendum

Congratulations on checking the correctness of a complex algorithm! While arrays are familiar data structures, the constraints added to them with this representation are tricky.

As you continue programming, you will find that this example is **typical** of what it is like to check the correctness of complex algorithms. They typically have invariants with many facts to keep track of and loop bodies with several cases to consider. We check their correctness exactly as you did above: by reasoning forward and backward into each case, comparing the facts known before and needed after to identify the additional facts that need to hold, and then figuring out why the known facts in that specific case ensure that they do always hold.

If you were able to work through this example, I think you can check the correctness of many other complex algorithms. With that in mind, let's do some more practice…

## Verifying Correctness of Intersection

Fill in the indicated assertions by reasoning in the direction indicated by the arrows. We will address the places where "?"s appear on the *next page*.

{{ Pre: -∞ = S[0] < S[1] < … < S[n] < S[n+1] = ∞ and -∞ = T[0] < T[1] < … < T[m] < T[m+1] = ∞ and
U is an array containing at least n+m+2 elements }}

↓ `U[0] = Float.NEGATIVE_INFINITY;`
↓ `int i = 1, j = 1, k = 1;`

`? answer 1 on the next page`

{{ Inv: Pre and -∞ = U[0] < U[1] < … < U[k-1] < min(S[i], T[j]) and max(S[i-1], T[j-1]) < min(S[i], T[j]) and
{U[1], U[2], …, U[k-1]} = {S[1], S[2], …, S[i-1]} ∩ {T[1], T[2], …, T[j-1]} }}

```
while (S[i] < Float.POSITIVE_INFINITY || T[j] < Float.POSITIVE_INFINITY) {
```
↓
```
  if (S[i] < T[j]) {
```
{{ Inv and S[i] < T[j] }}

`? answer 2 on the next page`

{{ Pre and -∞ = U[0] < U[1] < … < U[k-1] < min(S[i+1], T[j]) and max(S[i], T[j-1]) < min(S[i+1], T[j]) and {U[1], …, U[k-1]} = {S[1], …, S[i]} ∩ {T[1], …, T[j-1]} }}

↑
```
    i = i + 1;
  } else if (S[i] > T[j]) {
```
{{ Inv and S[i] > T[j] }}

`? answer 3 on the next page`

{{ Pre and -∞ = U[0] < … < U[k-1] < min(S[i], T[j+1]) and max(S[i-1], T[j]) < min(S[i], T[j+1]) and {U[1], …, U[k-1]} = {S[1], …, S[i-1]} ∩ {T[1], …, T[j]} }}

↑
```
    j = j + 1;
  } else {
```
{{ Inv and S[i] = T[j] }}

```
    U[k] = S[i];  // ? answer 4 on the next page
```
{{ Pre and -∞ = U[0] < … < U[k] = min(S[i+1], T[j+1]) and max(S[i], T[j]) < min(S[i+1], T[j+1]) and {U[1], …, U[k]} = {S[1], …, S[i]} ∩ {T[1], …, T[j]} }}

↑
```
    i = i + 1;
```
↑
```
    j = j + 1;
```
↑
```
    k = k + 1;
  }
}
```

↓ `U[k] = Float.POSITIVE_INFINITY;`

{{ Inv and S[i] = +∞ and T[j] = +∞ and U[k] = +∞ }}

`? answer 5 on the next page`

{{ Post: -∞ = U[0] < U[1] < … < U[k] = ∞ and
{U[1], U[2], …, U[k-1]} = {S[1], S[2], …, S[n]} ∩ {T[1], T[2], …, T[m]} }}

## Explanations

Explain why the invariant holds initially (answer 1) and why the postcondition holds (answer 5):

1. ∵ k=1 ∴ k-1=0 ⇒ $-\infty > U[0]$

∵ i=1, j=1, $S[1] > -\infty$, $T[1] > -\infty$ ⇒ $-\infty < U[0] < \min(S[1], T[1])$

$S[i-1] = -\infty$, $T[j-1] = -\infty$ ⇒ $\max(S[i-1], T[j-1]) < \min(S[i], T[j])$

$\phi = \{U[0], \ldots, U[k-1]\} = \phi \cap \phi = \{S[0], \ldots, S[i-1]\} \cap \{T[0], \ldots, T[j-1]\}$

5. From $S[i] = T[j] < +\infty$ ⇒ i=m+1, j<n+1

plug in i and j ⇒ $-\infty < U[0] < \cdots < U[k] = +\infty$

$\{U[1], U[2], \ldots, U[k-1]\} = \{S[1], \ldots, S[m]\} \cap \{T[1], \ldots, T[n]\}$

## Comparing Assertions

The three "?"s in the middle of the loop (answers 2-4) appear next to an assignment and between two assertions that you filled in on the previous page. In each case, compare the two assertions and explain which *additional facts*, not listed in the top assertion, are needed to establish that the bottom assertion holds. (Do not yet worry about whether these facts hold.)

2. $\{S[i]\} \cap \{T[1], \ldots, T[j-1]\} = \phi$

3. $\{S[1], \ldots, S[i-1]\} \cap \{T[j]\} = \phi$

4. $S[i] = T[j] = U[k]$

## More Explanations

For the three cases above, explain why the additional facts must hold from what we know in the top assertion and the assignment statement in between them.

2. $S[i] < T[j]$ , $S[i] > T[j-1]$

Since $T[1] < T[2] < \dots < T[j-1]$

$\Rightarrow S[i] \neq T[j']$ . $\forall j' \in \{1, 2, \dots, j-1\}$

$\Rightarrow \{S[i]\} \cap \{T[1], \dots T[j-1]\} = \emptyset$

3. Similar to Answer 2.

4. we can know that $S[i] = T[j]$ , $U[k] = S[i]$

## Code Review

For which of the five cases we considered earlier (before the loop, after the loop, and the three parts of the if/else if/else statement), do you think the author should have included comments?

Before the loop should state the Inv.