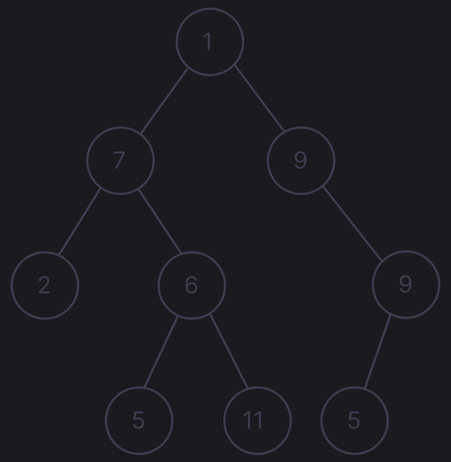
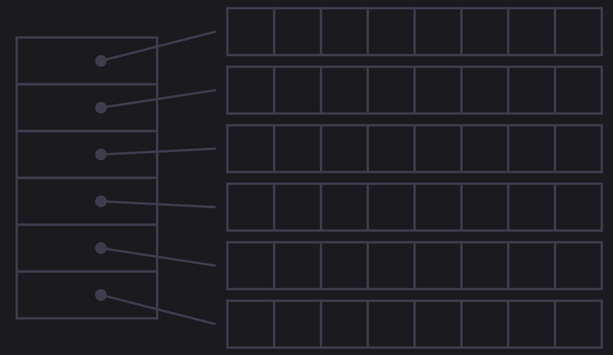
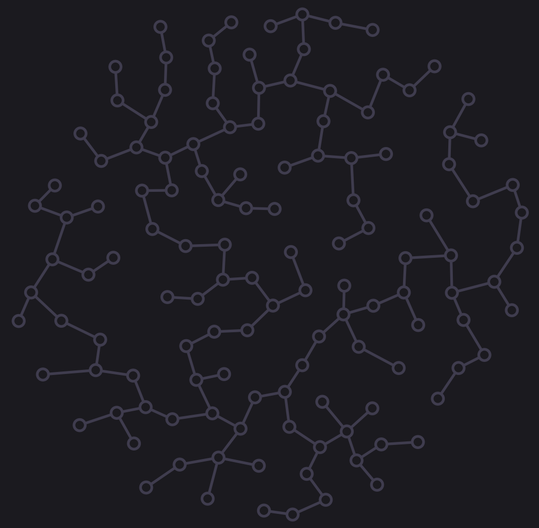
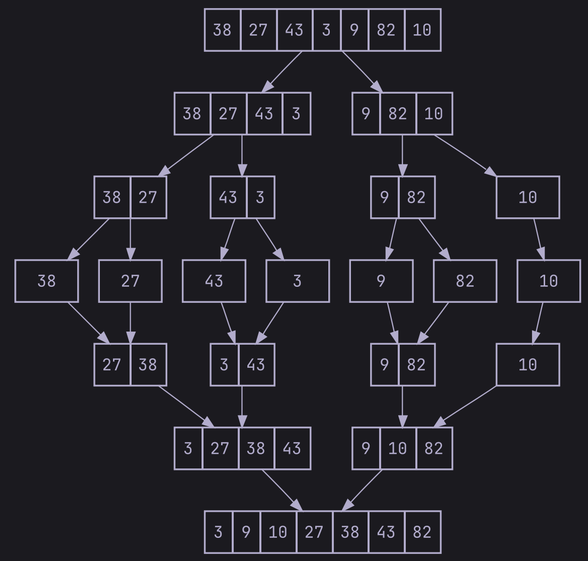




data structures & algorithms

Tutorial 5





we are learning about this thing today 👀

Burning questions from last week?

This week’s lab

Recursion (Continued)

Today we are learning about **Merge Sort**! One of the most famous sorting algorithms

- Recursion (revisited)
- Decrease and conquer
- Merge Sort
- Recursive Exponential

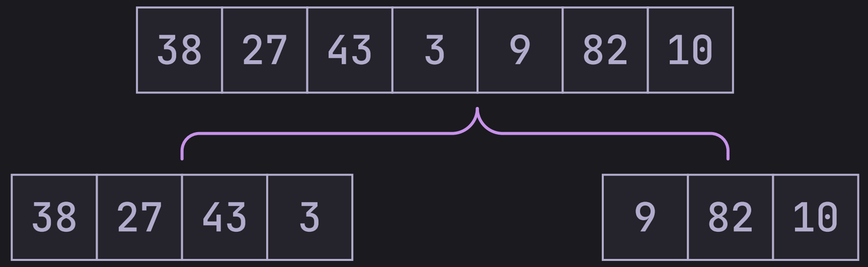
Divide and Conquer

1. **Divide:** Break the problem into smaller subproblems that are similar to the original problem but smaller in size
2. **Conquer:** Solve the subproblems recursively until the subproblems become simple enough to solve directly
3. **Combine:** Combine the solutions to the subproblems into a solution to the original problem

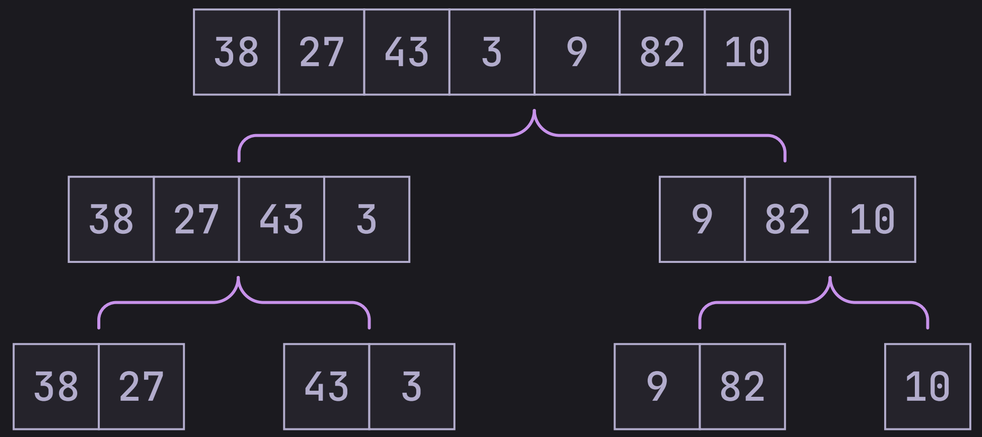
Merge Sort

Merge sort is a really efficient
divide and conquer sorting algorithm

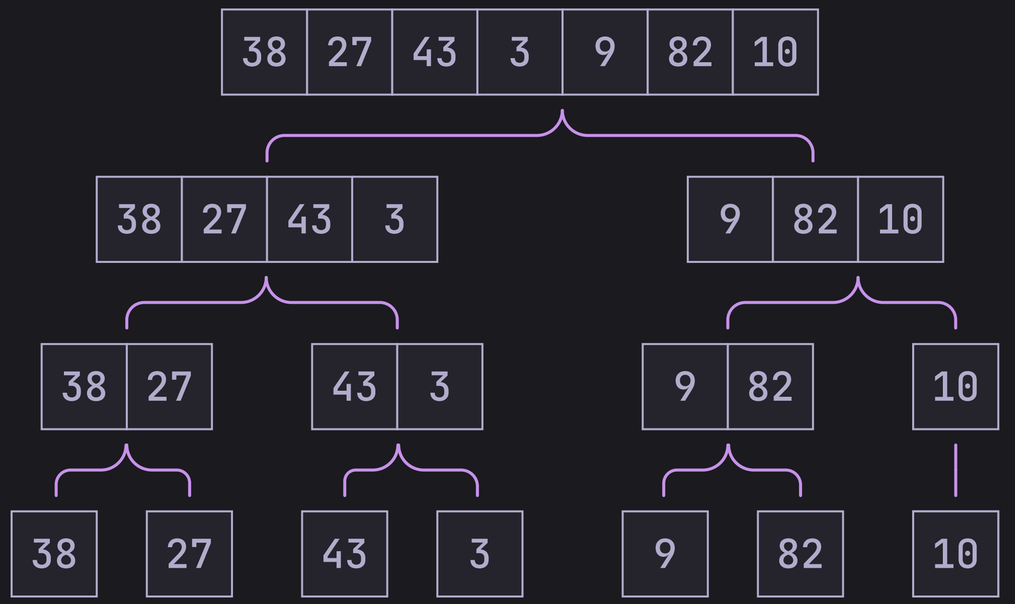
We first **divide** the problem into smaller problems. In this case we divide the vector into smaller vectors



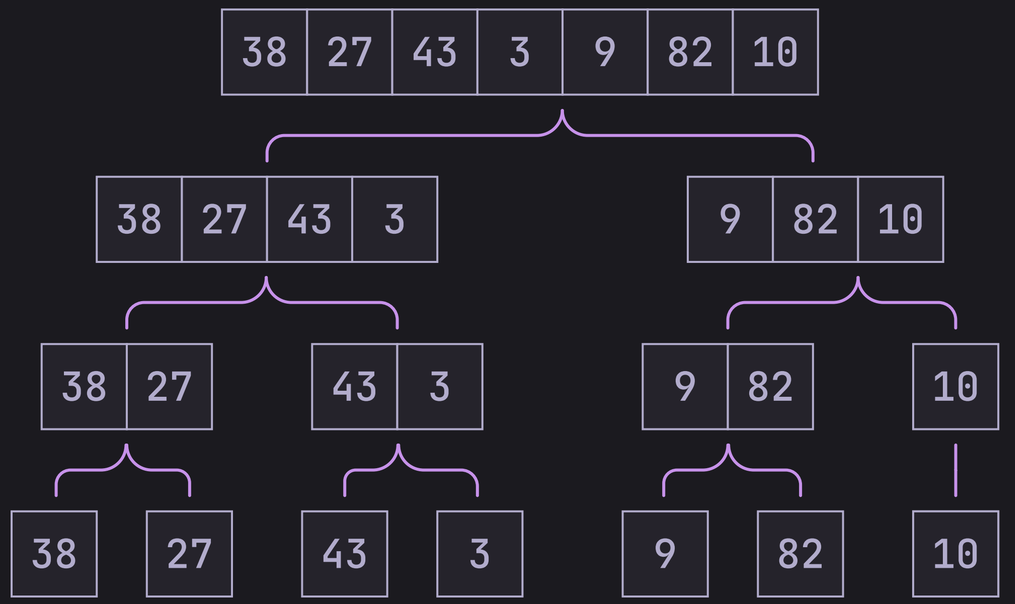
These subproblems are still too big, so we **divide** them again



Let’s **divide** them one last time

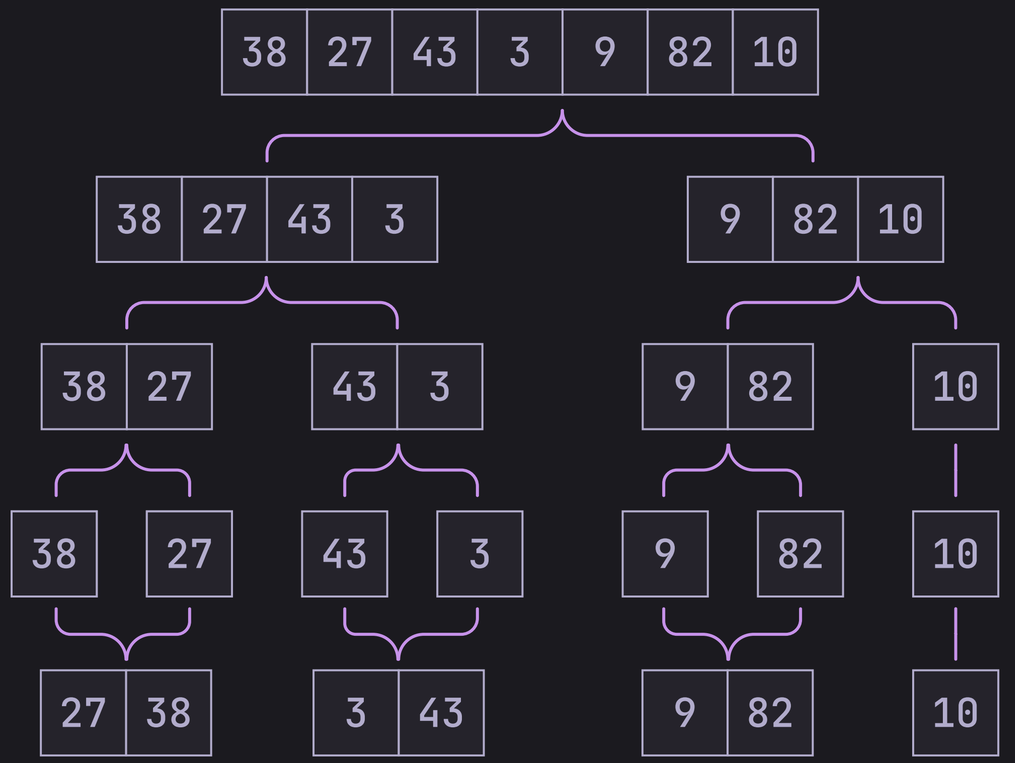


Now let’s **conquer** each subproblem!

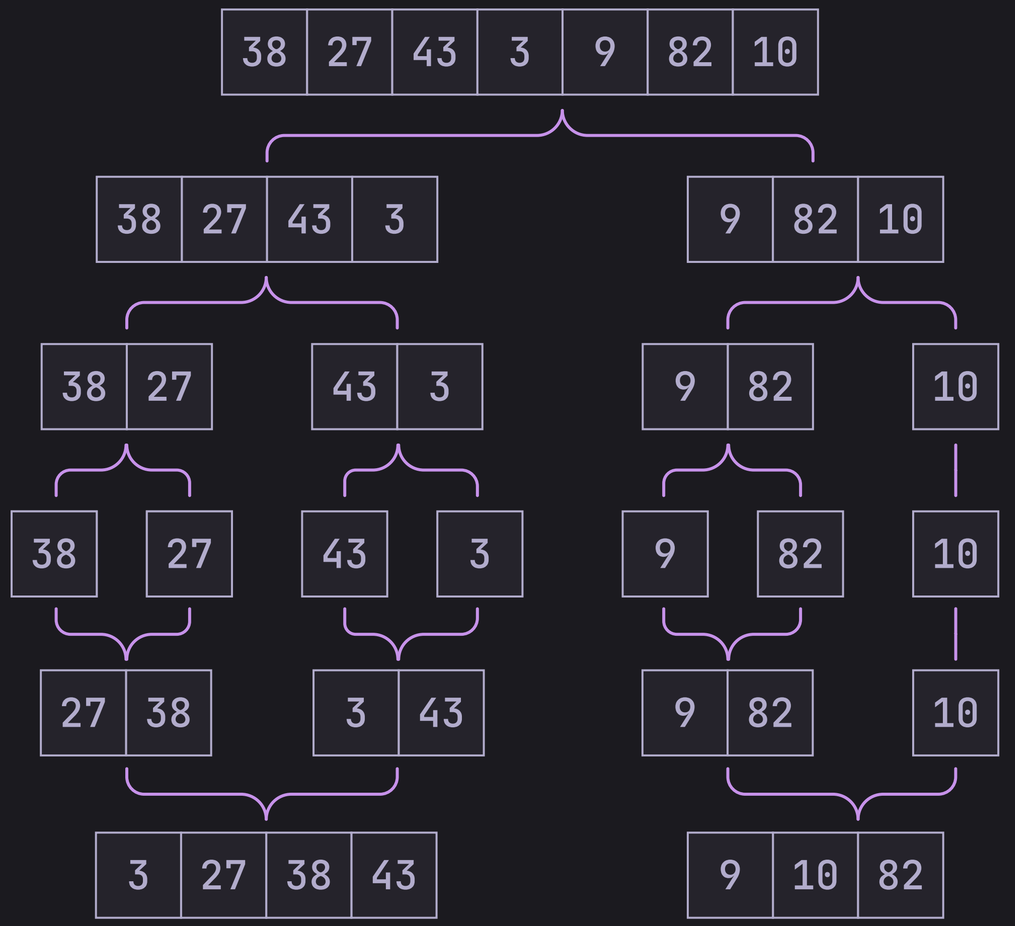


Now let’s **conquer** each subproblem.

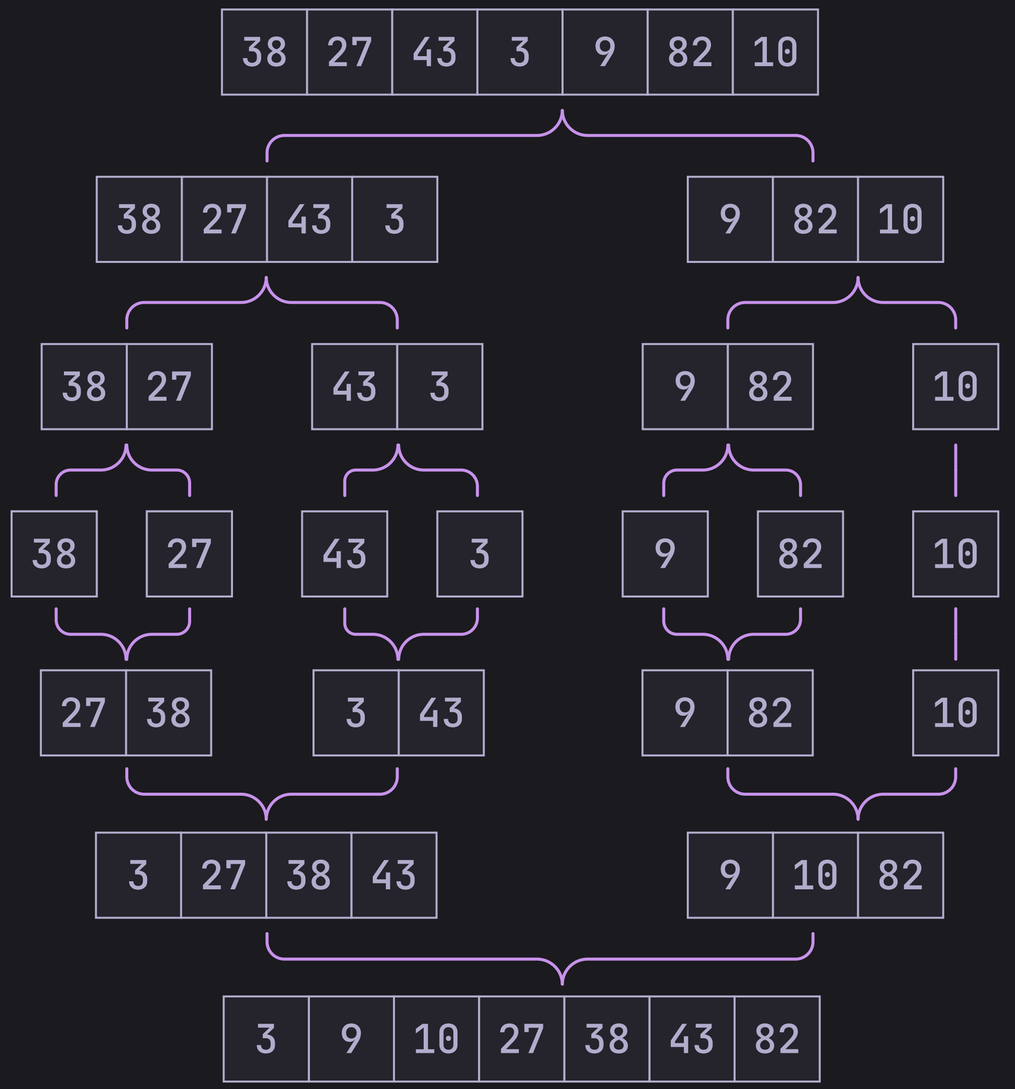
Do you notice each of the sub arrays are already sorted because they only contain one element!



Now let’s **combine** each of the conquered subproblems



Now let’s **combine** each of the conquered subproblems

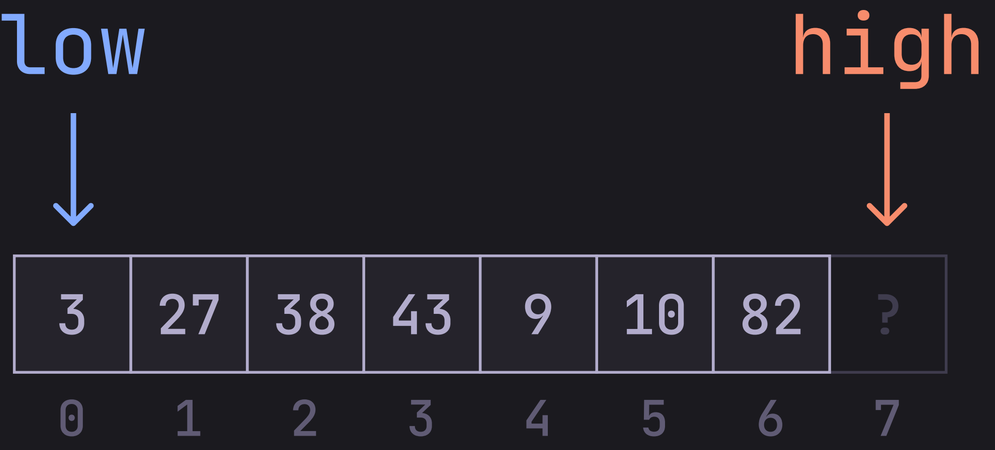


Now let’s **combine** each of the conquered subproblems

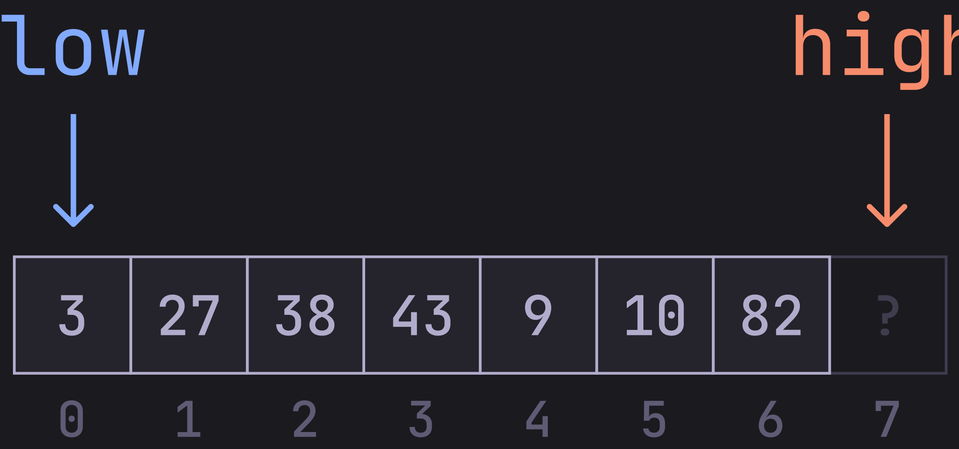
To better understand merge sort we will break down each step

1. Dividing the vectors using iterators (Divide)
2. The Recursive Step
3. Base Case (Conquer)
4. Merging Function (Combine)

Dividing the vectors: Finding the mid point



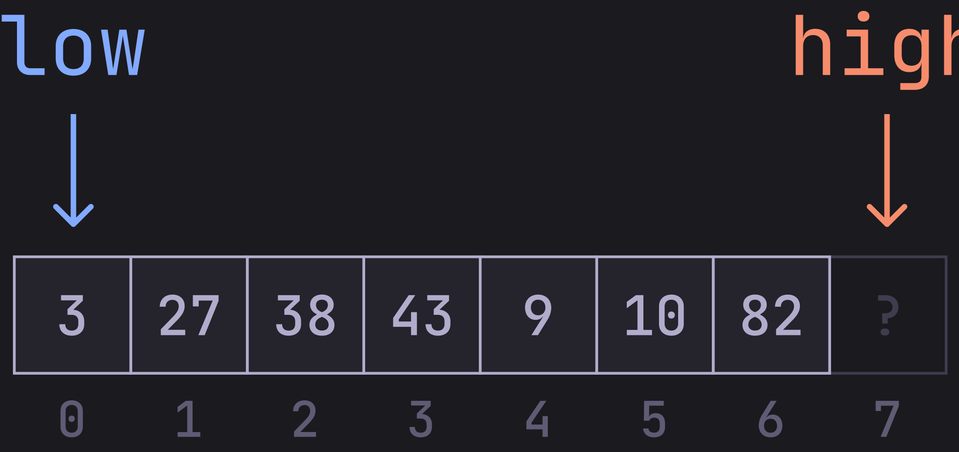
Dividing the vectors: Finding the mid point



high - low = 7

a vector iterator minus another iterator returns a int which is the number of elements between the two iterators

Dividing the vectors: Finding the mid point

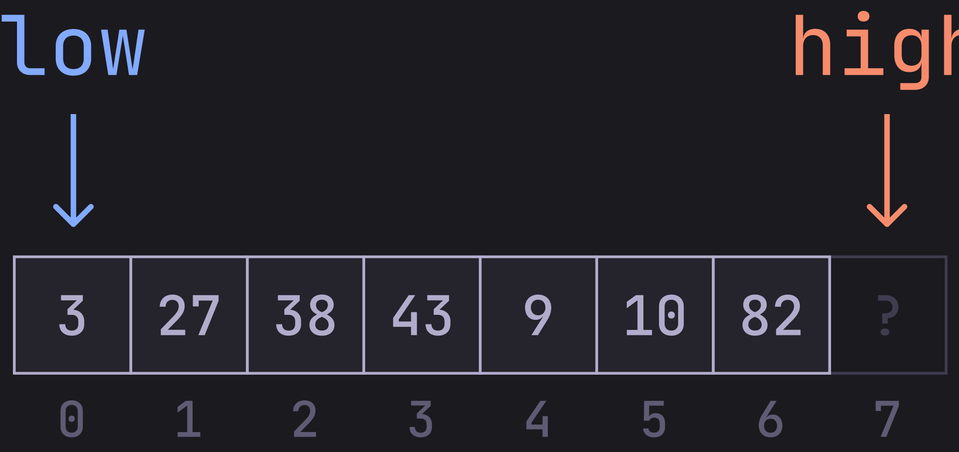



```
(high - low) / 2 = 3.5
```

int

We can divide an int which will return another int

Dividing the vectors: Finding the mid point




```
(high - low) / 2 = 3
```

int

We can divide an int which will return another int

(rounded down if decimal result)

Dividing the vectors: Finding the mid point


```
low + (high - low) / 2
```

iterator — int

Adding an int to and iterator will return an iterator offset by the amount of the int

Dividing the vectors: Finding the mid point

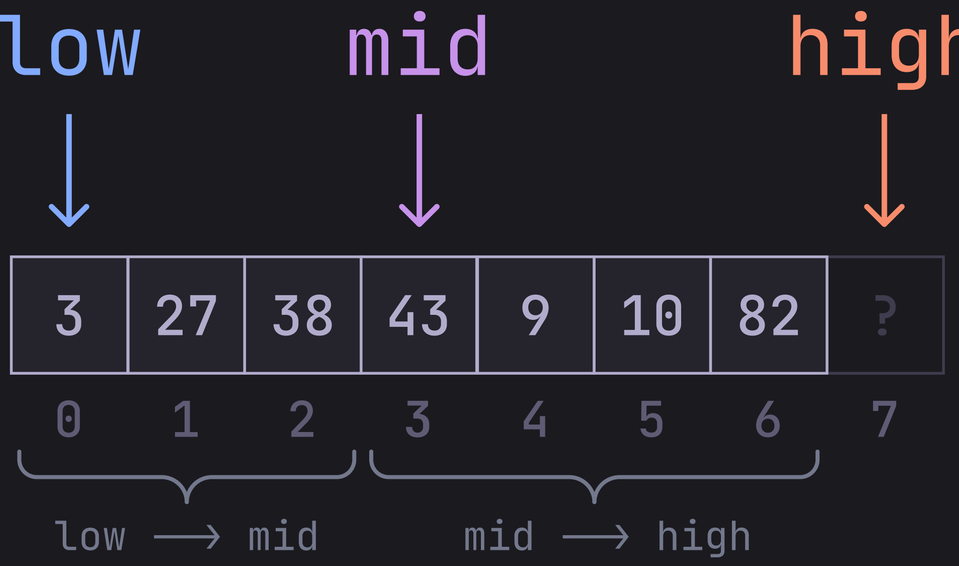

```
auto mid = low + (high - low) / 2
```

iterator

int

we can then save this result in a variable

Dividing the vectors: Iterator Ranges



These iterators define two sections of the vector

Dividing the vectors: Iterator Ranges

Now imagine splitting the vector at the mid point

Dividing the vectors: Iterator Ranges

Now imagine splitting the array at the mid point

Dividing the vectors: Passing Ranges to Functions


```
template <typename Iter>
void printElements(Iter start, Iter end) {
    for (Iter it = start; it != end; it++) {
        std::cout << *it << " ";
    }
}
```

Dividing the vectors: Passing Ranges to Functions


```
template <typename Iter>
void printElements(Iter start, Iter end) {
    for (Iter it = start; it != end; it++) {
        std::cout << *it << " ";
    }
}
```

```
printElements(low, mid)
// outputs: 3 27 38
```

Dividing the vectors: Passing Ranges to Functions


```
template <typename Iter>
void printElements(Iter start, Iter end) {
    for (Iter it = start; it != end; it++) {
        std::cout << *it << " ";
    }
}
```

```
printElements(low, mid)
// outputs: 3 27 38

printElements(mid, high)
// outputs: 43 9 10 82
```

Dividing the vectors: Passing Ranges to Functions

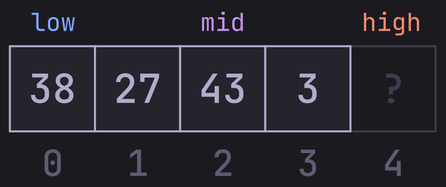

```
mergesort(low, mid); // Sorts the left half
mergesort(mid, high); // Sorts the right half
```

The Recursive Step

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```

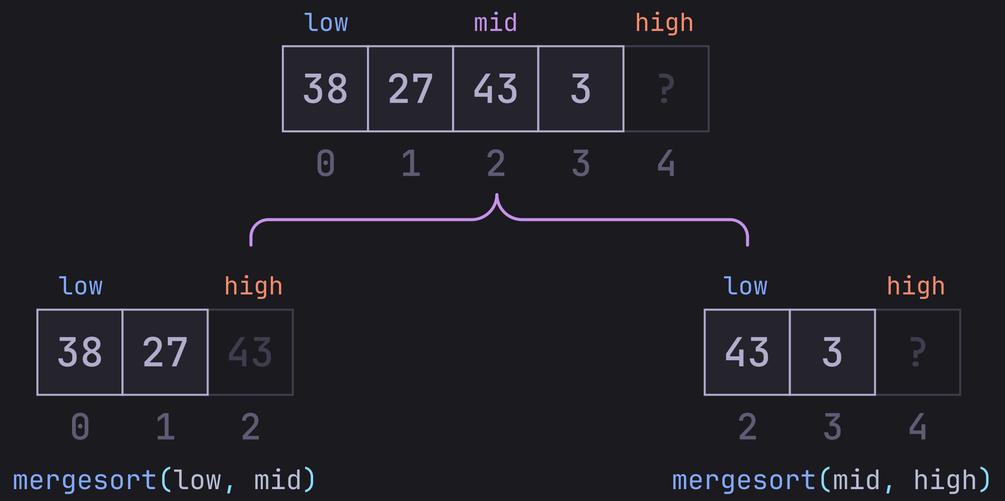
The Recursive Step (Divide)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



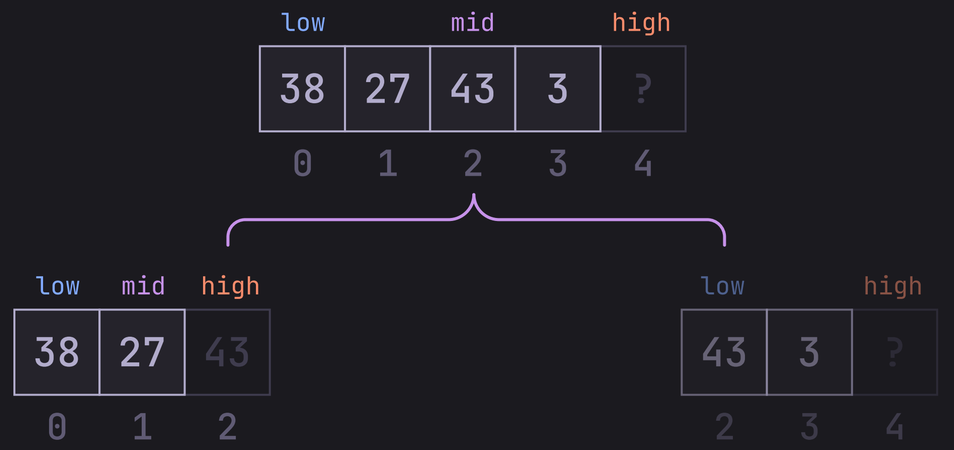
The Recursive Step (Divide)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



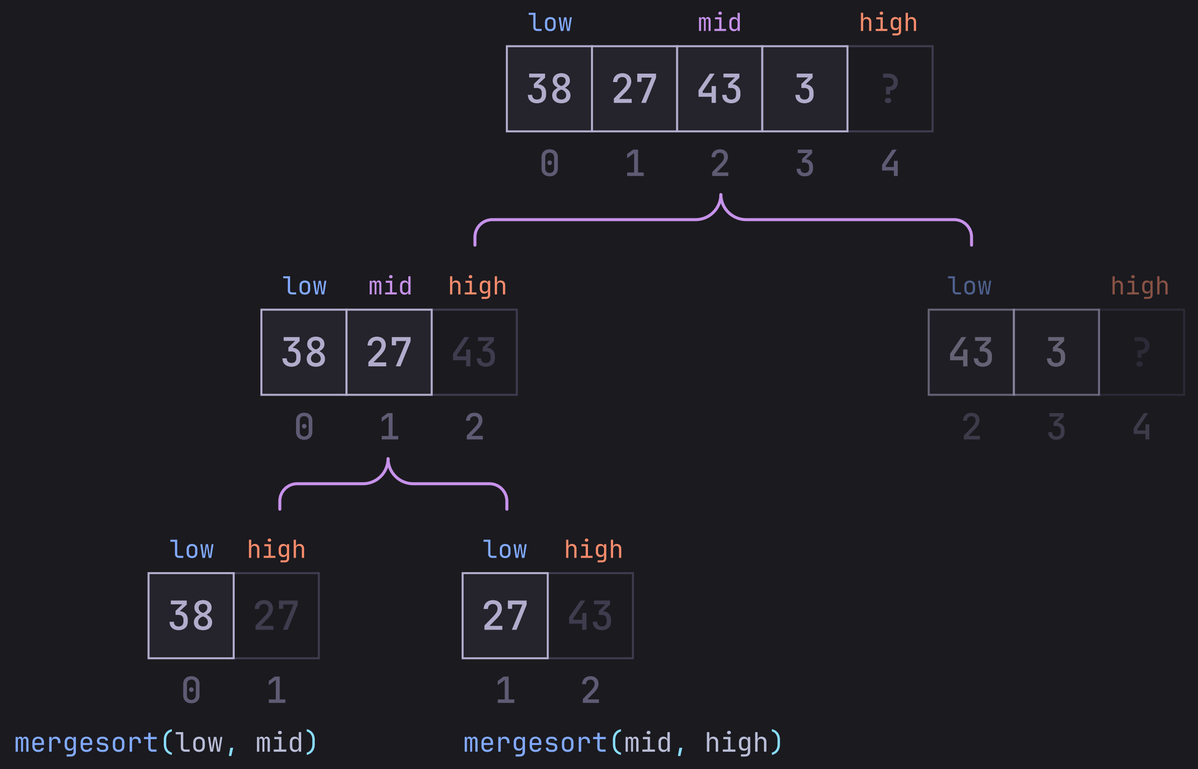
The Recursive Step (Divide)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



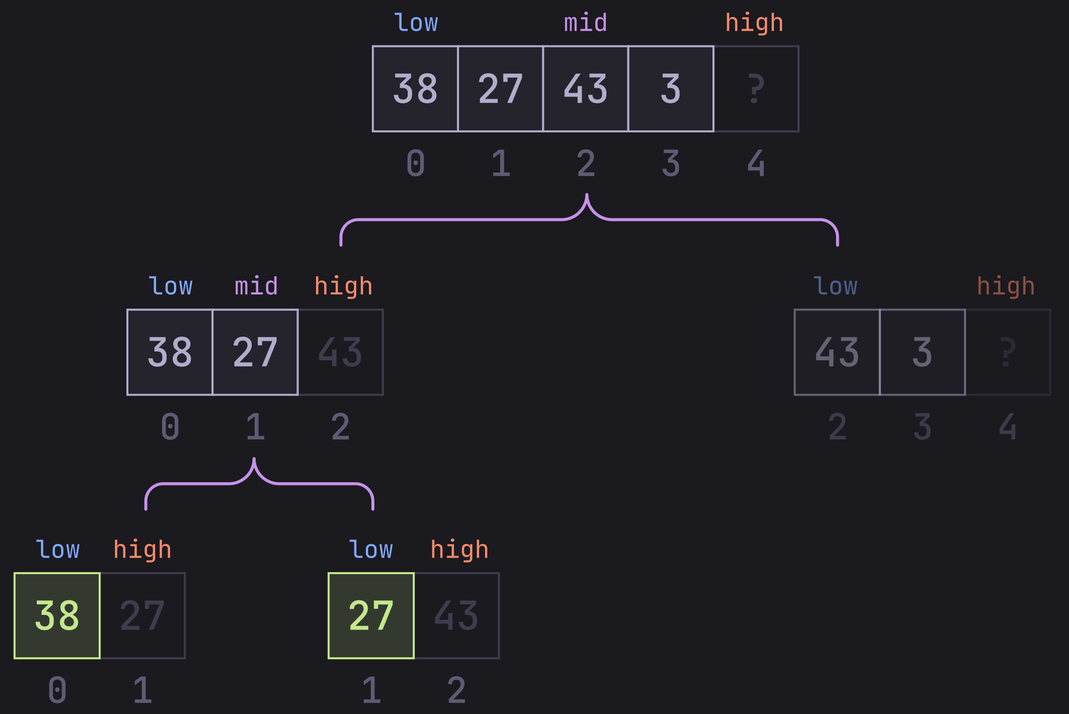
The Recursive Step (Divide)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



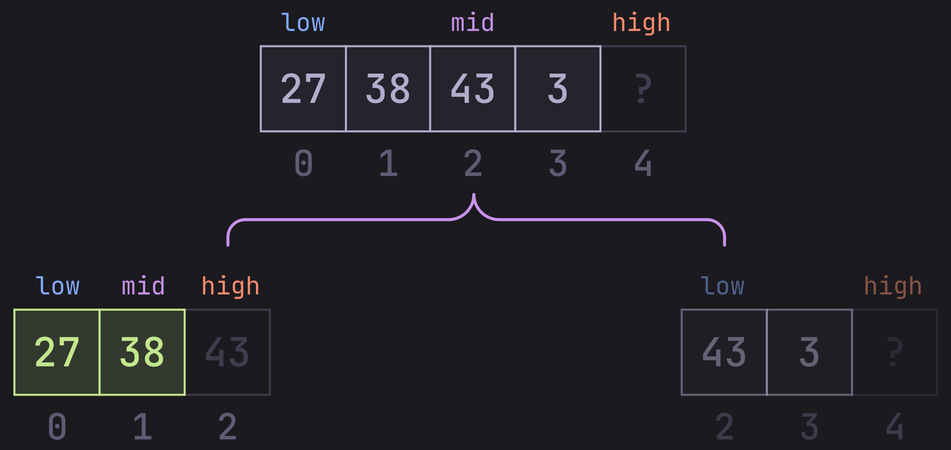
The Recursive Step (Base Case)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



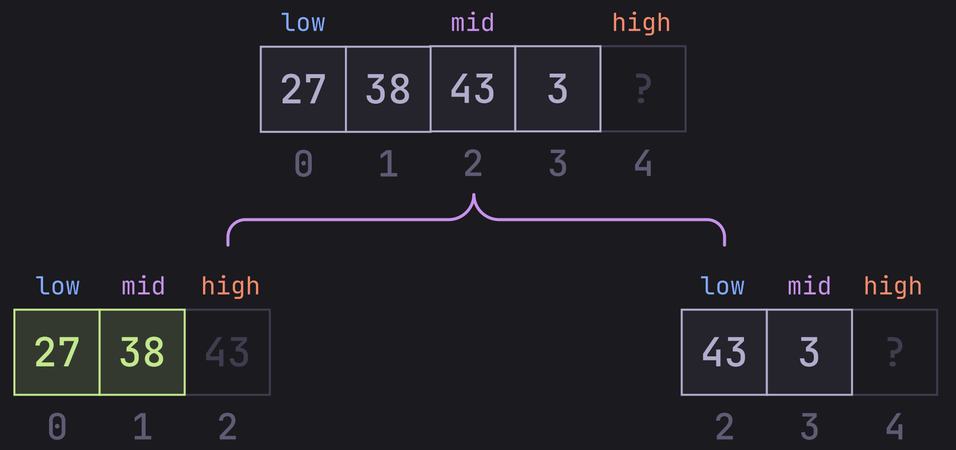
The Recursive Step (Combine)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



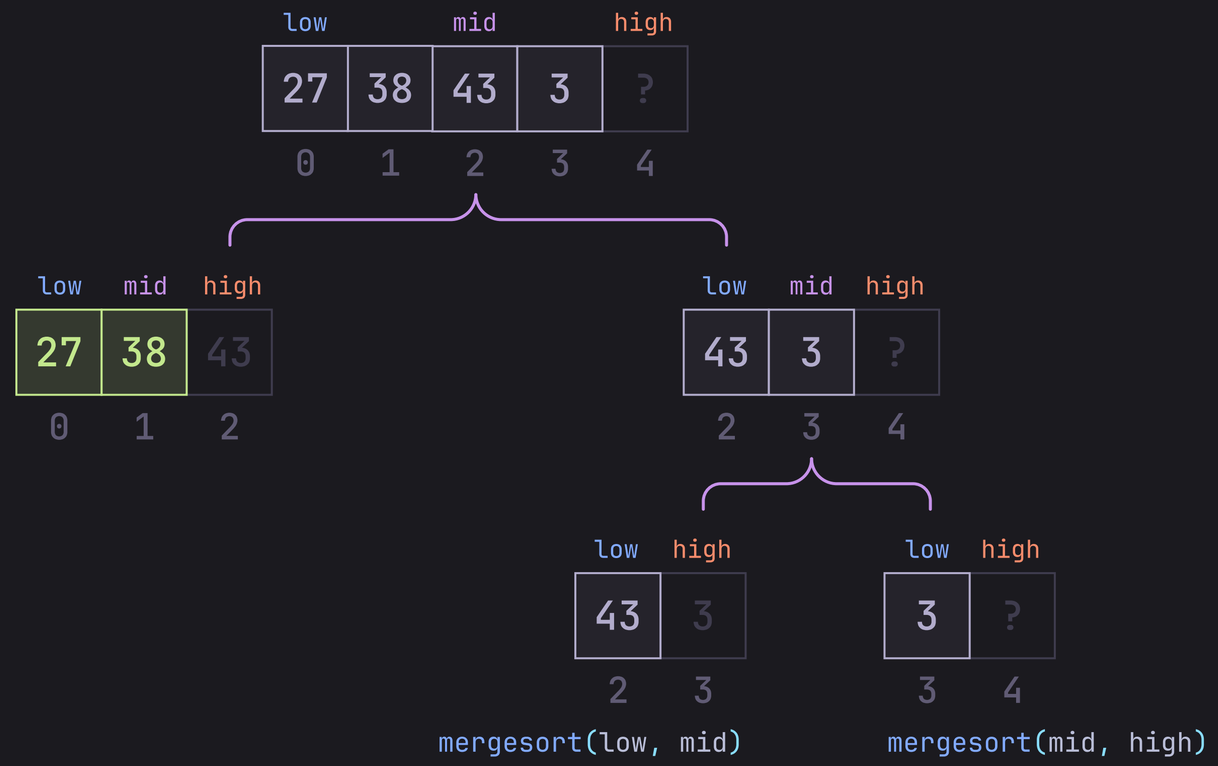
The Recursive Step (Divide)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



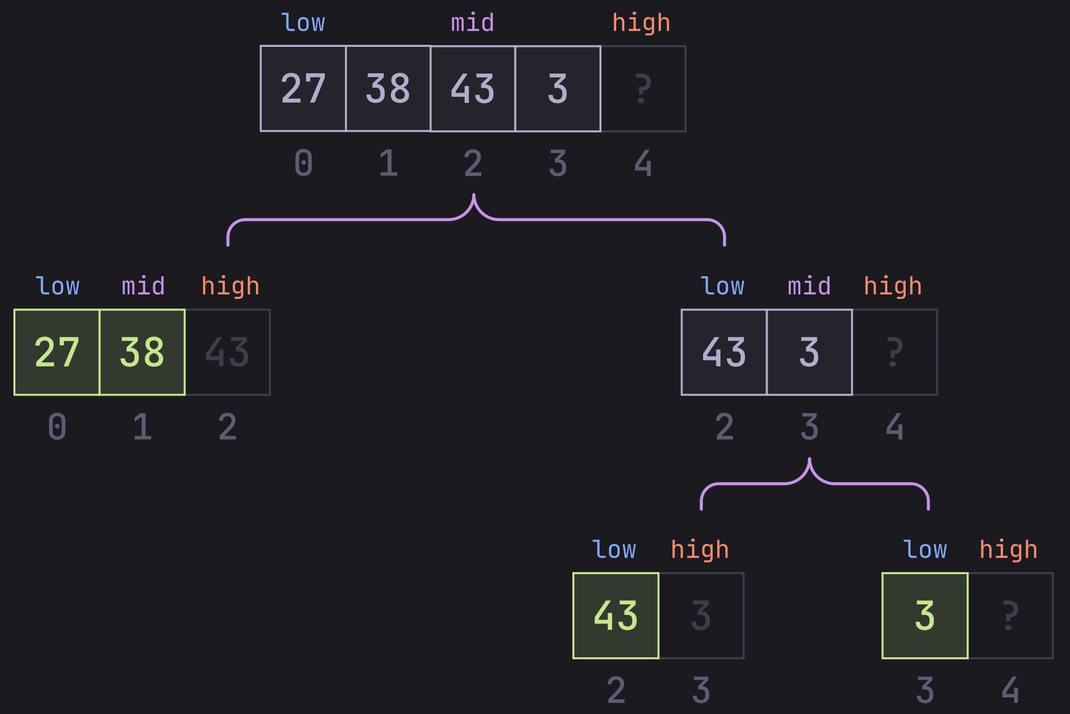
The Recursive Step (Divide)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



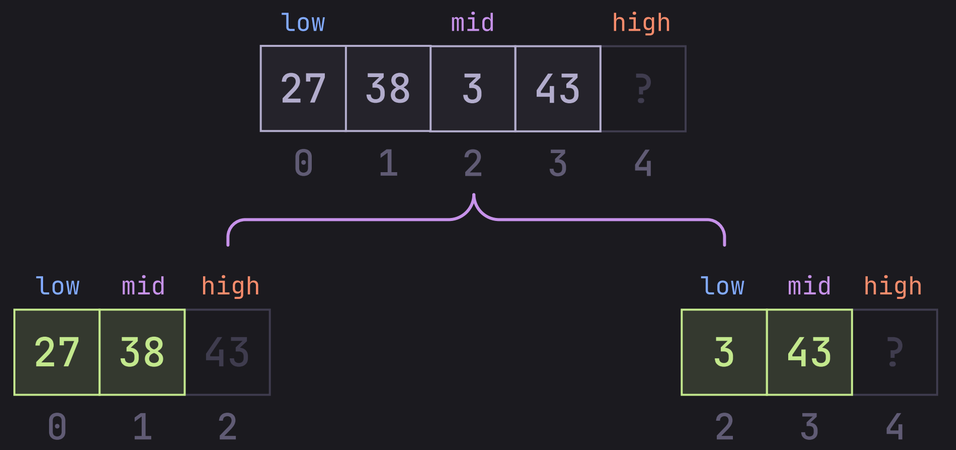
The Recursive Step (Base Case)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



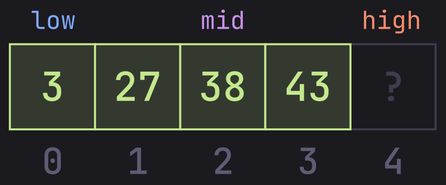
The Recursive Step (Combine)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



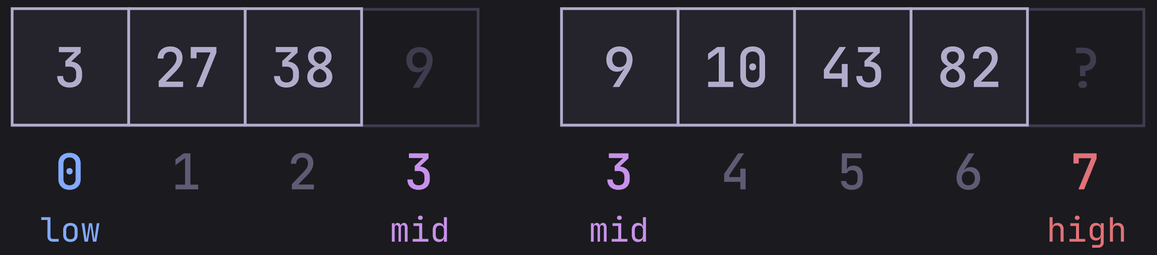
The Recursive Step (Combine)

```
template <typename Iter>
void mergesort(Iter low, Iter high) {
  if (high - low < 2) {
    return;
  }

  Iter mid = low + (high - low)/2;
  mergesort(low, mid);
  mergesort(mid, high);
  merge(low, mid, high);
}
```



Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```

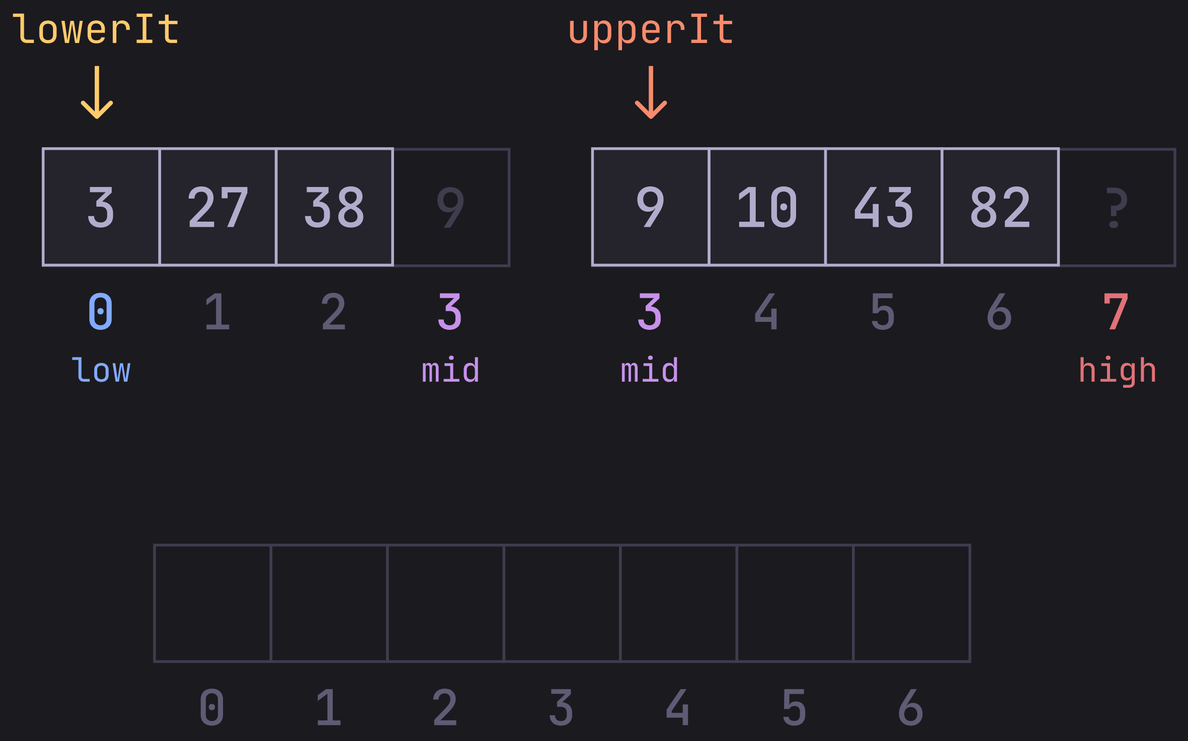


Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```

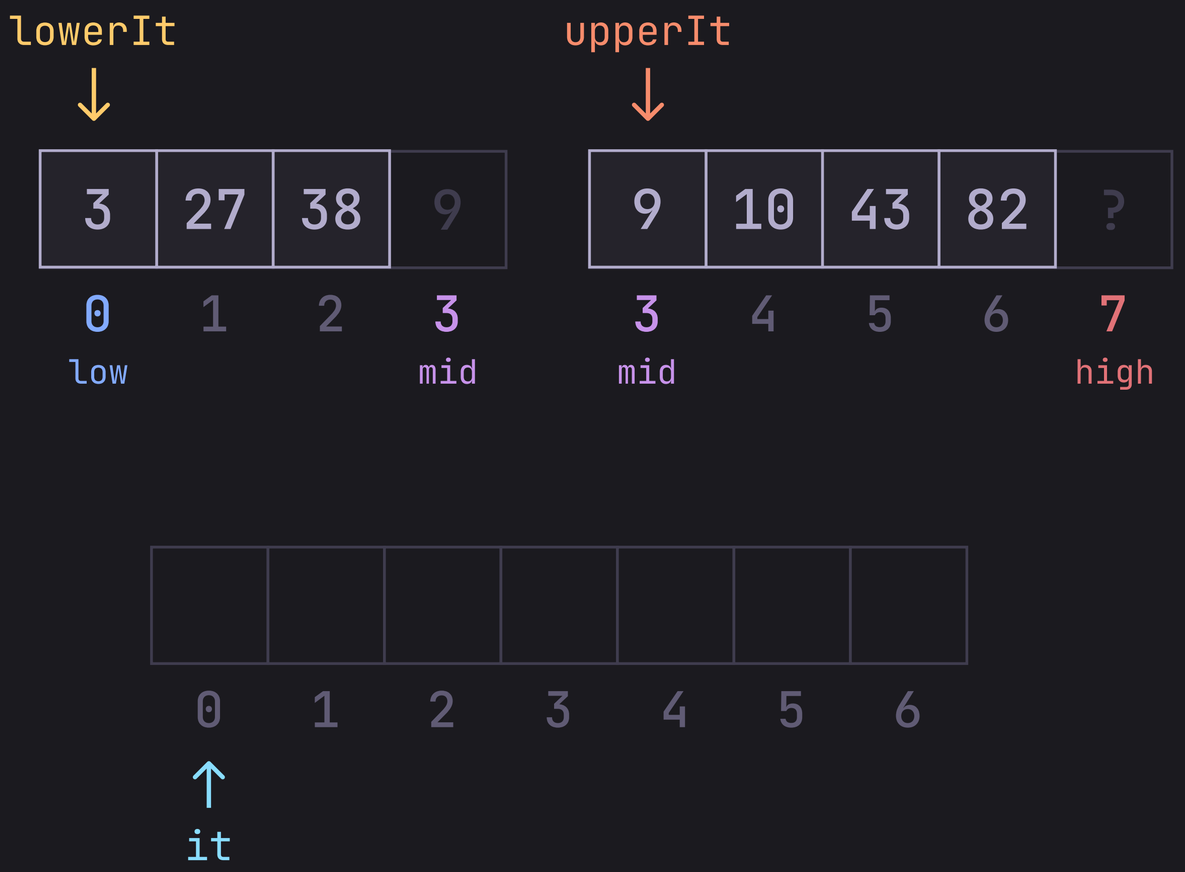
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



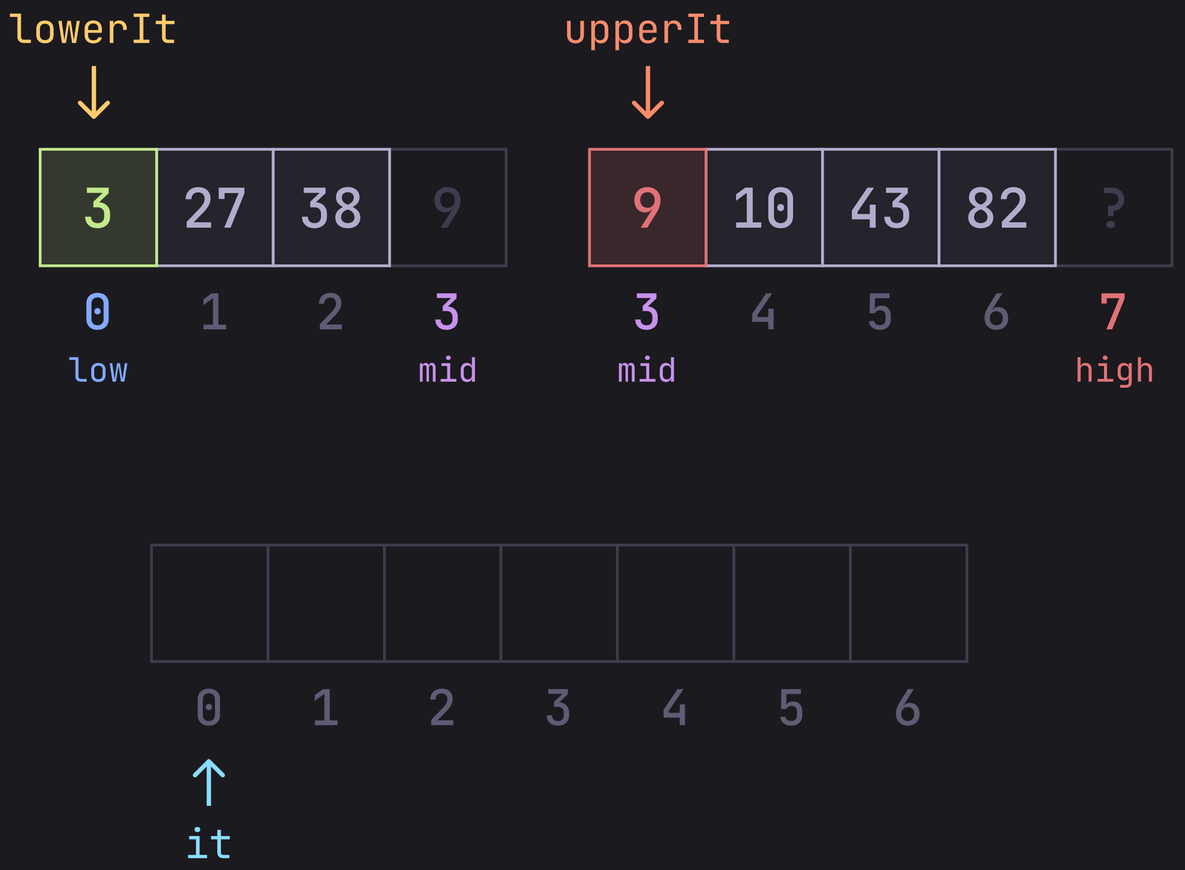
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



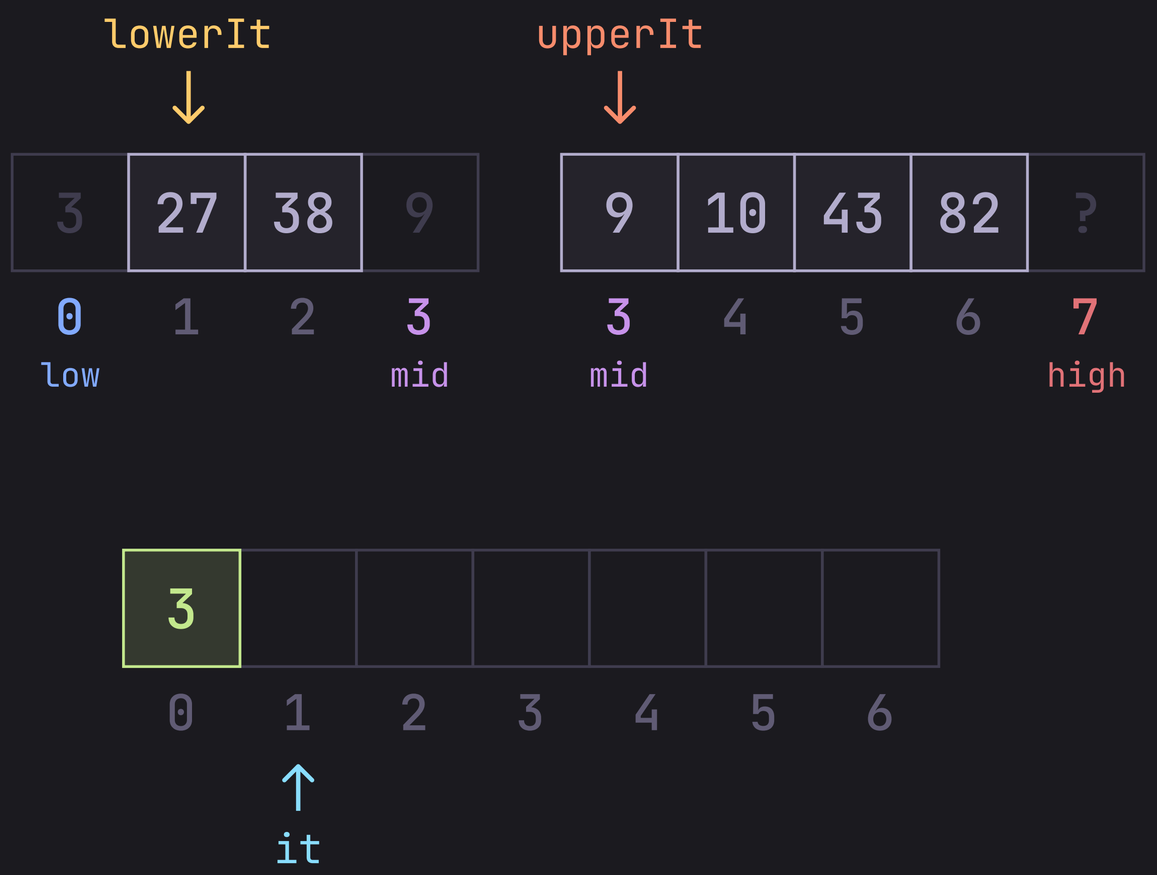
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



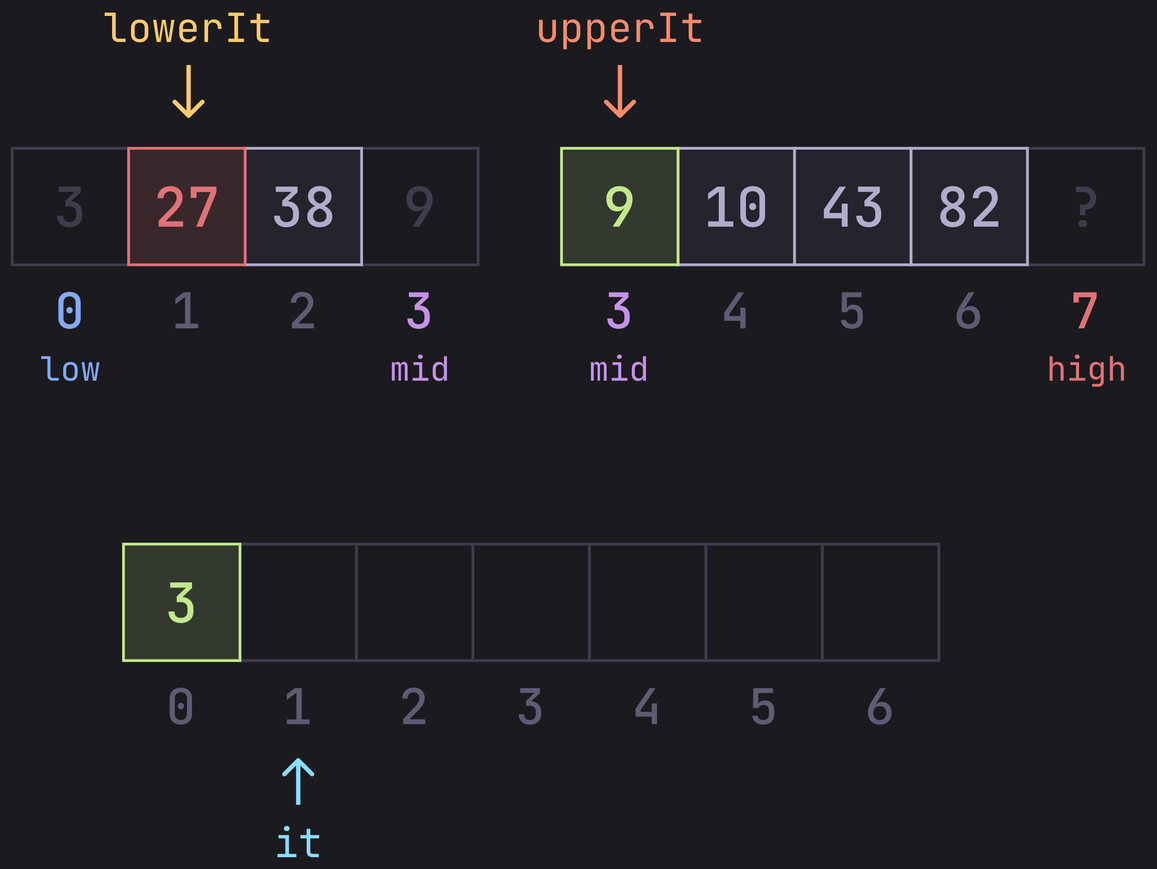
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



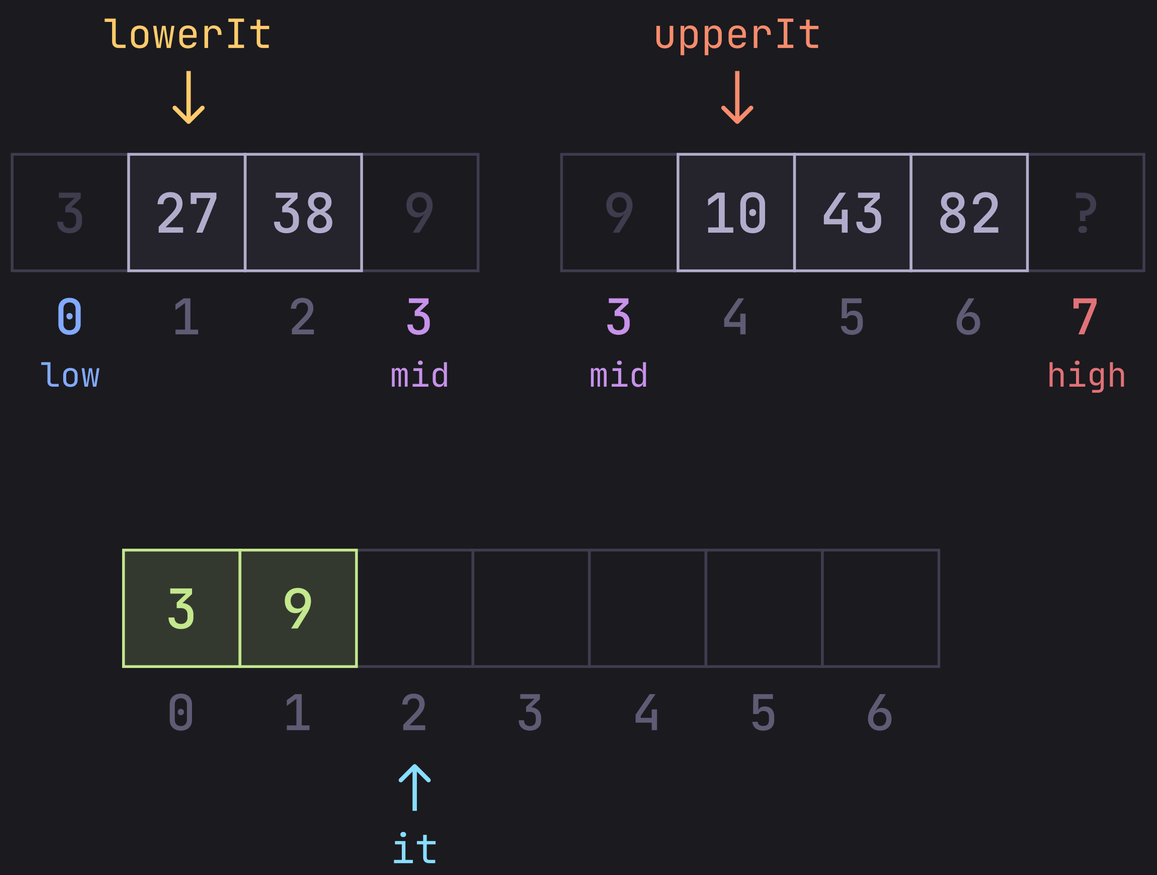
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



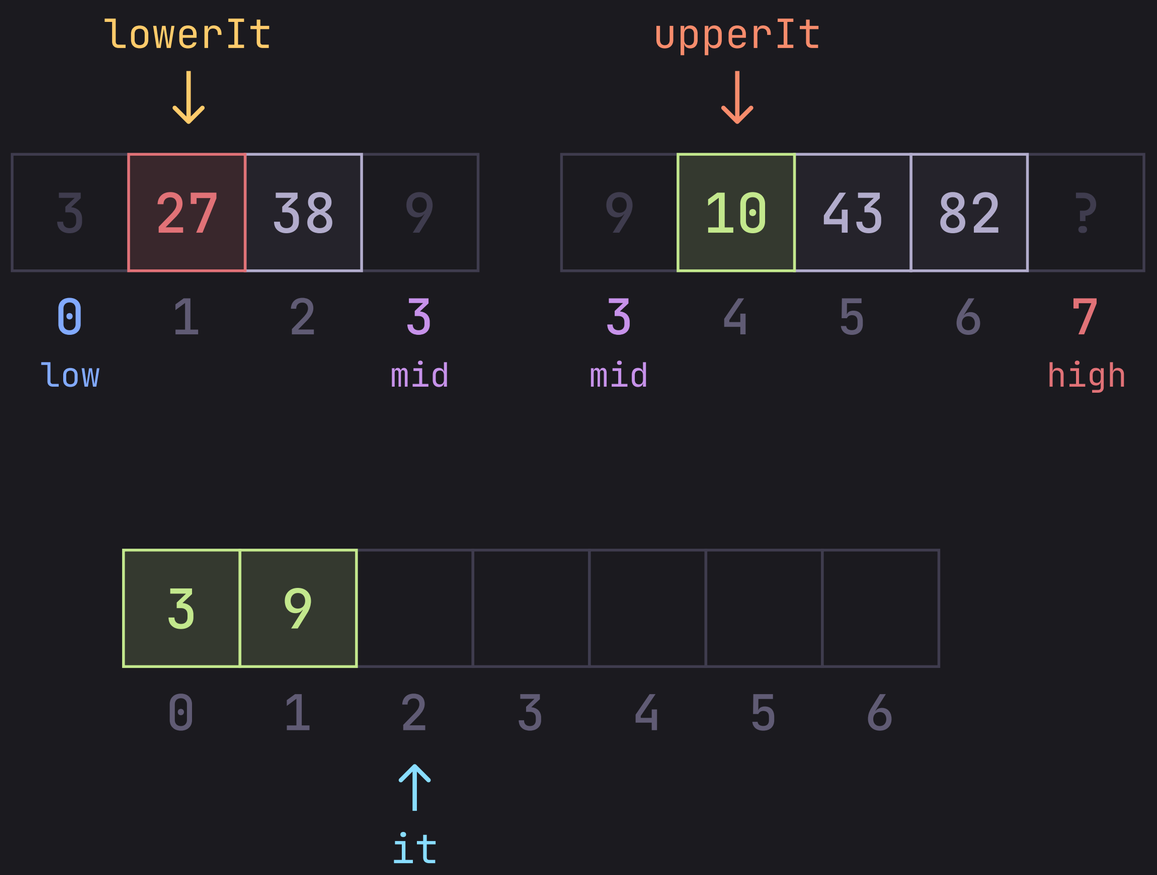
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



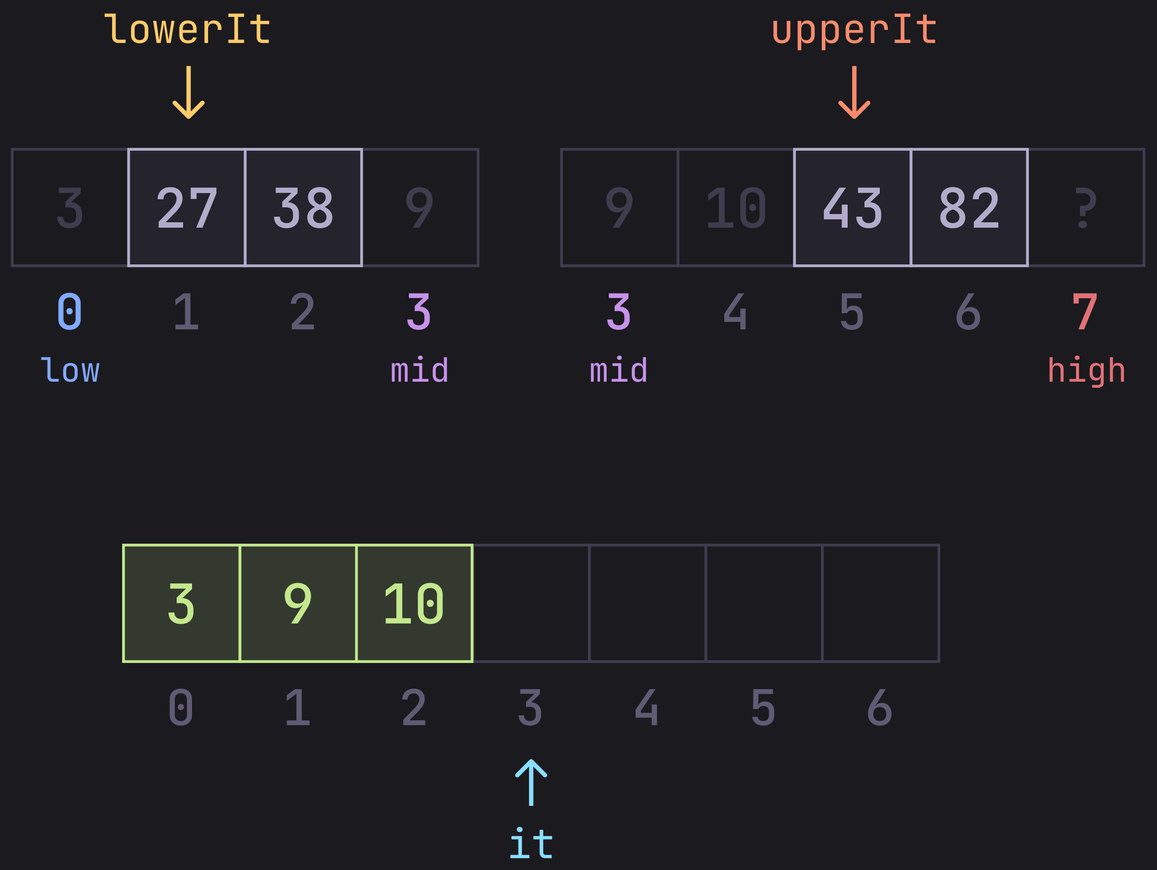
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



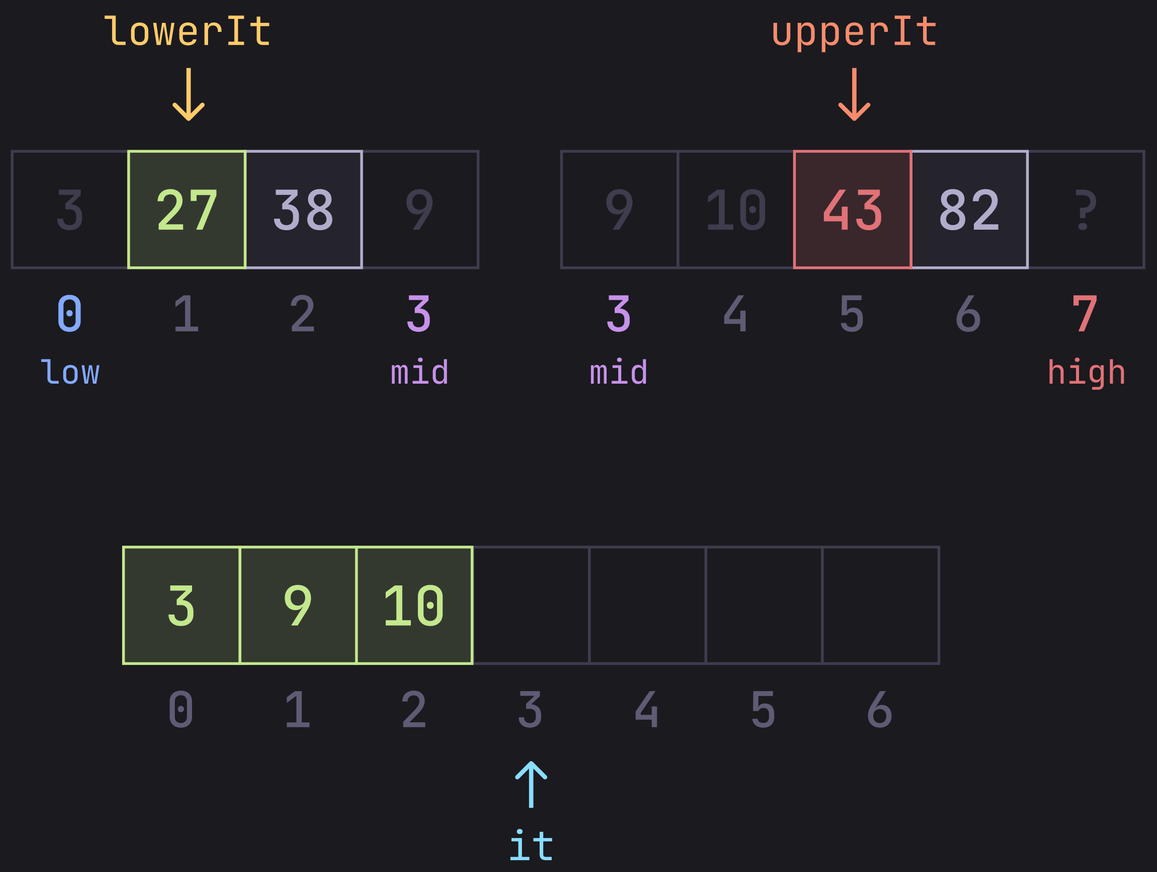
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



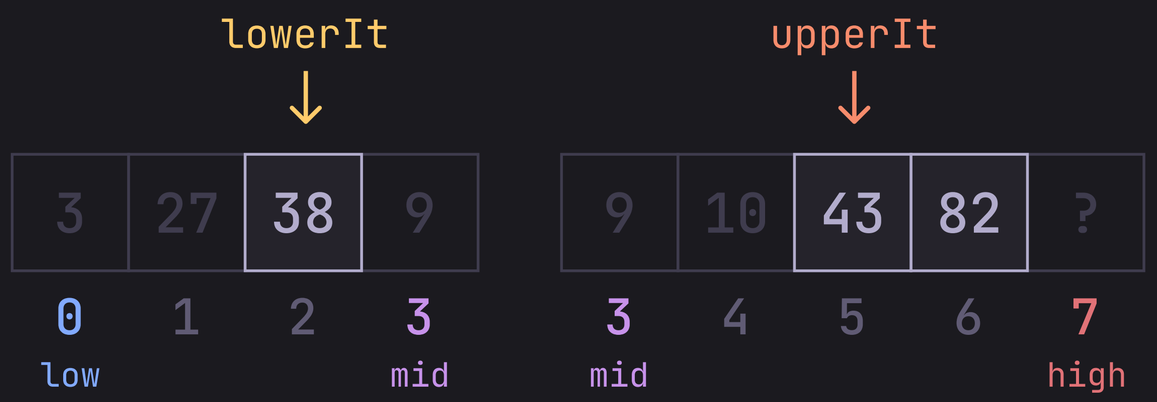
Merging Function

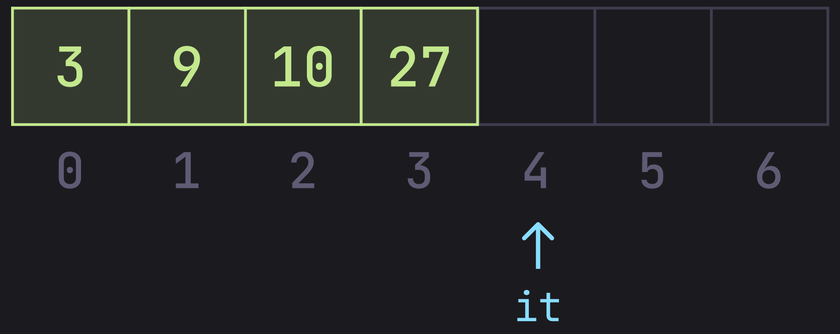
```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



Merging Function

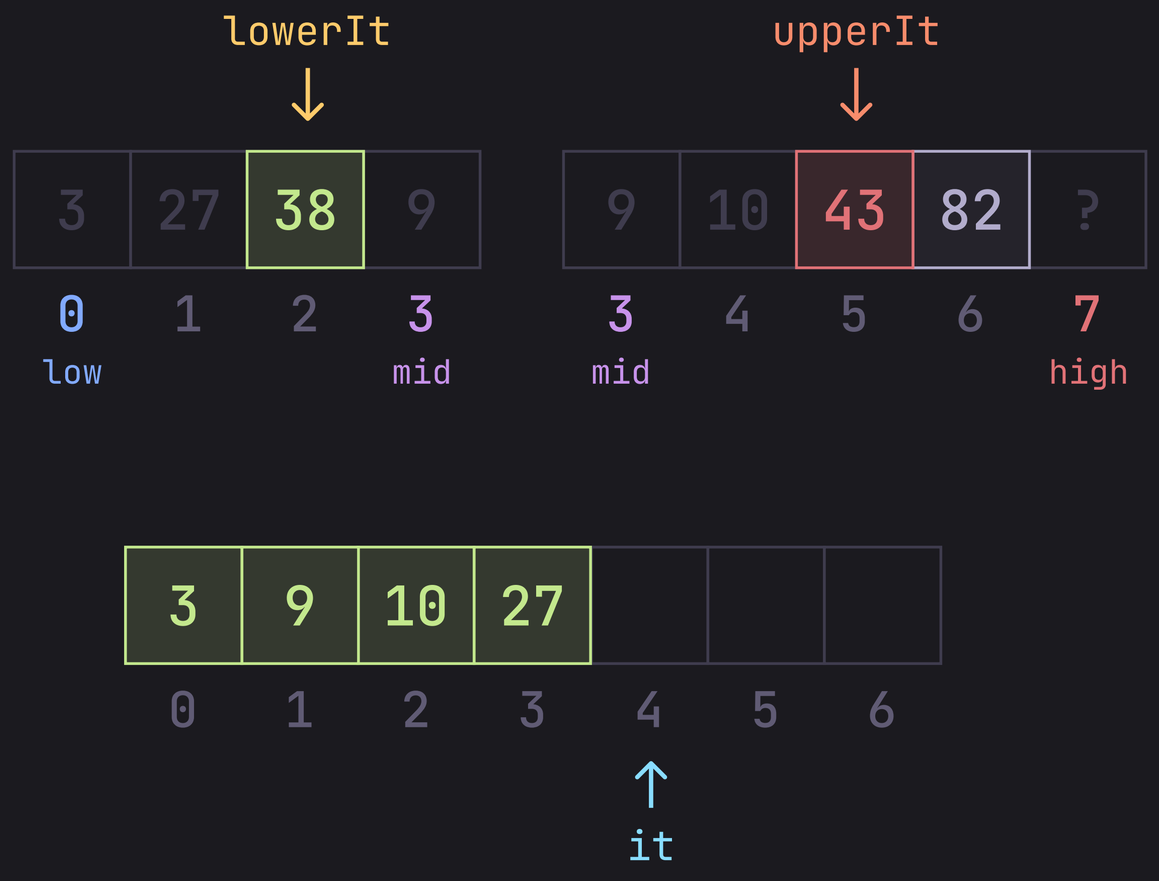
```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```





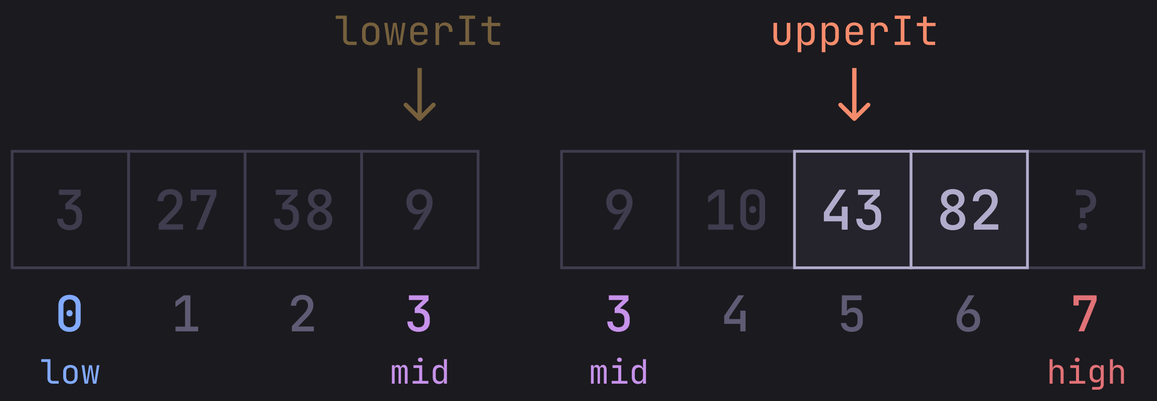
Merging Function

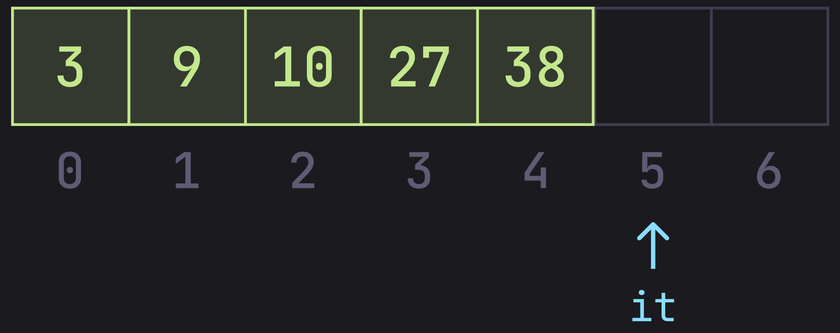
```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



Merging Function

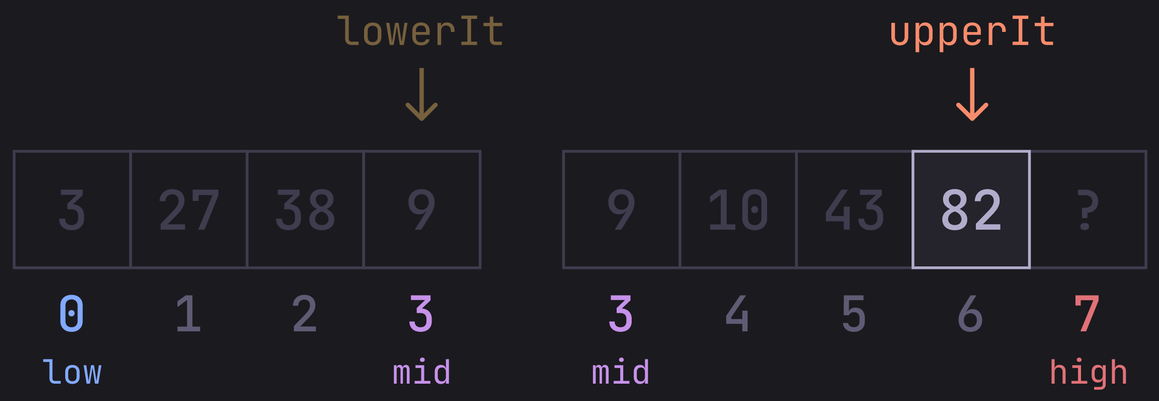
```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```

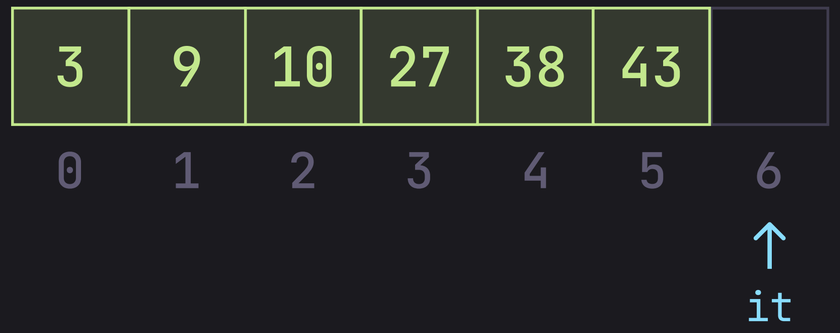




Merging Function

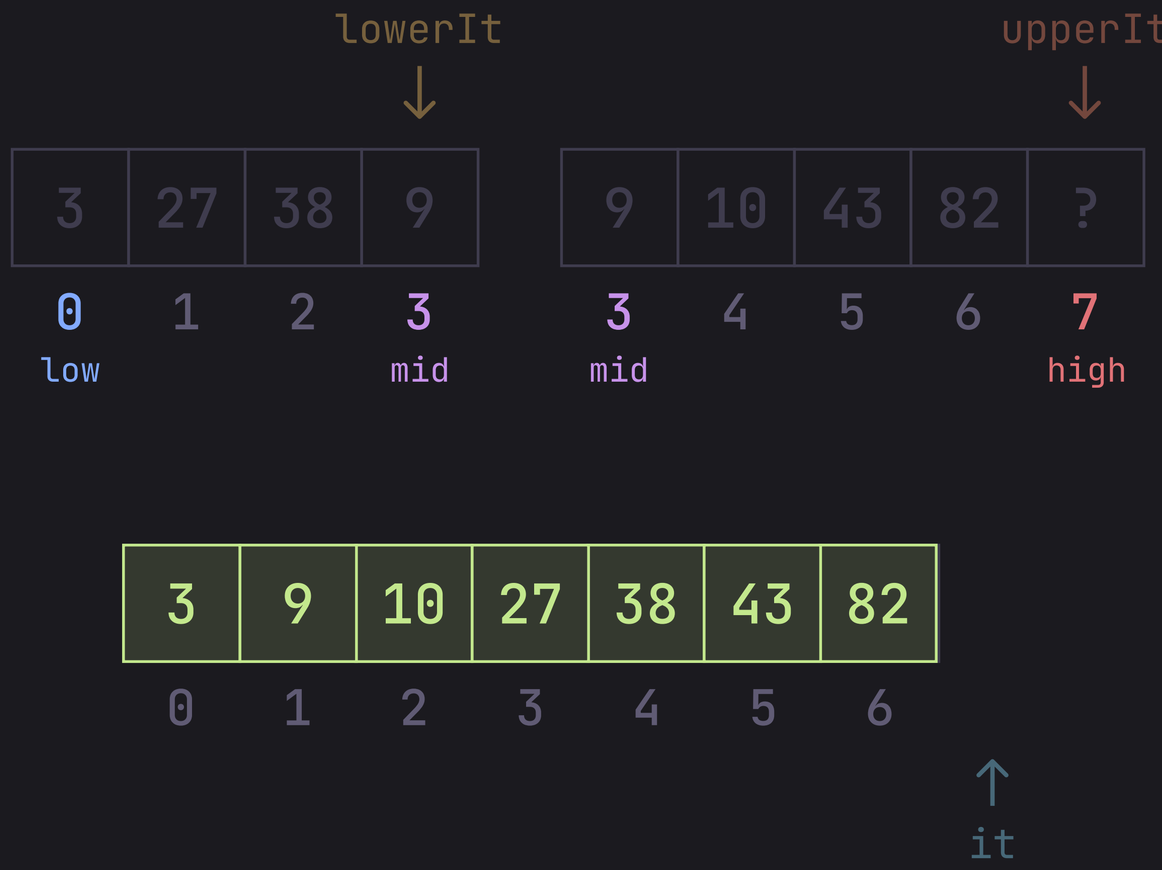
```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```





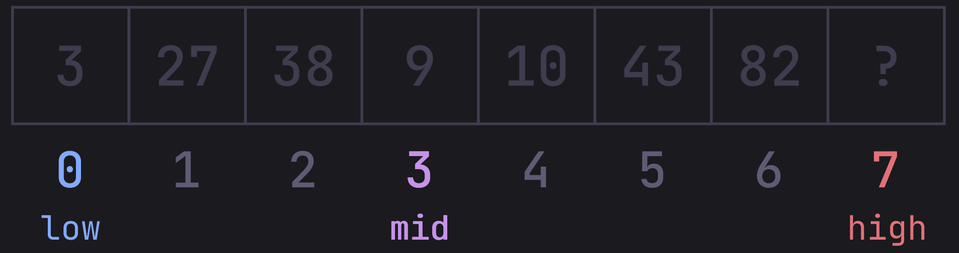
Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



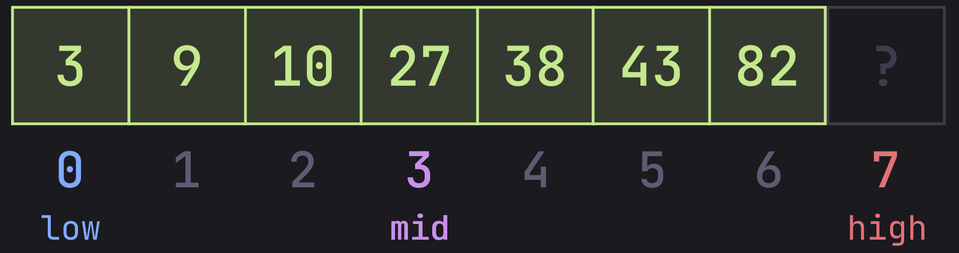
Merging Function

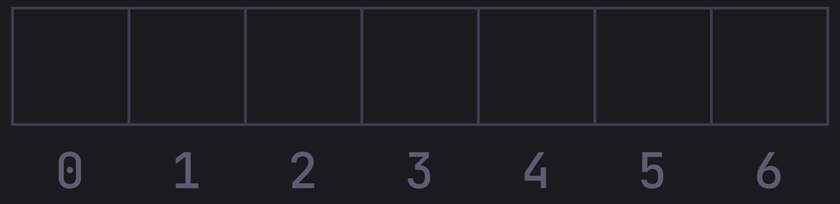
```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```



Merging Function

```cpp
template <typename Iter>
void merge(Iter low, Iter mid, Iter high) {
  using T = std::iter_value_t<Iter>;
  std::vector<T> mergedVec(high - low);

  Iter lowerIt = low;
  Iter upperIt = mid;
  for (Iter it = mergedVec.begin(); it != mergedVec.end(); it++) {
    if (upperIt == high) {
      *it = *lowerIt;
      lowerIt++;
    } else if (lowerIt == mid) {
      *it = *upperIt;
      upperIt++;
    } else if (*lowerIt <= *upperIt) {
      *it = *lowerIt;
      lowerIt++;
    } else {
      *it = *upperIt;
      upperIt++;
    }
  }
  std::copy(mergedVec.begin(), mergedVec.end(), low);
}
```





In Summary

Merge sort is basically **three simple steps**

```
mergesort(low, mid); // Sort the left half
mergesort(mid, high); // Sort the right half
merge(low, mid, high); // merge the two sorted halves
```

Lets give it a go in Ed!

Recursive Exponential

$$x^a = \underbrace{x \times x \times \cdots \times x}_{a \text{ times}}$$

```
int pow(int base, int exponent)
```

Recursive Exponential

$$3^5 = \underbrace{3 \times 3 \times 3 \times 3 \times 3}_{\text{5 times}}$$

```
pow(3, 5) = 3 * 3 * 3 * 3 * 3
```

Recursive Exponential

$$5^6 = \underbrace{5 \times 5 \times 5 \times 5 \times 5 \times 5}_{\text{6 times}}$$

```
pow(5, 6) = 5 * 5 * 5 * 5 * 5 * 5
```

Recursive Exponential

$$5^6 = (5 \times 5) \times (5 \times 5) \times (5 \times 5)$$

```
pow(5, 6) = (5 * 5) * (5 * 5) * (5 * 5)
```

Recursive Exponential

$$5^6 = \underbrace{5^2 \times 5^2 \times 5^2}_{\text{3 times}}$$

```
pow(5, 6) = (5 * 5) * (5 * 5) * (5 * 5)
```

Recursive Exponential

$$5^6 = (5^2)^3$$

```
pow(5, 6) == pow(5*5, 3)
```

Recursive Exponential

$$9^8 = (9^2)^4$$

```
pow(9, 8) == pow(9*9, 4)
```

Recursive Exponential

$$x^a = (x^2)^?$$

```
pow(x, a) == pow(x*x, ?)
```

assuming that a is even

Recursive Exponential

$$x^a = (x^2)^{(a/2)}$$

```
pow(x, a) == pow(x*x, a/2)
```

assuming that a is even

Recursive Exponential

But what if a is **odd**? 👀

Recursive Exponential

$$5^9 = 5 \times 5^8$$

```
pow(5, 9) = 5 * pow(5, 8)
```

But what if a is **odd**?

Recursive Exponential

$$5^9 = 5 \times 5^8$$

$$x^a = x \times x^{(a-1)}$$

```
pow(x, a) == x * pow(x, a-1)
```

And so, more generally we can write this

Recursive Exponential

$$x^a = \begin{cases} (x^2)^{(a/2)} & \text{if } a \text{ is even} \\ x \times x^{(a-1)} & \text{if } a \text{ is odd} \end{cases}$$

```
pow(x, a) = { pow(x*x, a/2)     // if a is even
            { x * pow(x, a-1)   // if a is odd
```

Recursive Exponential

Lastly what could we use as
a **base case**??

Recursive Exponential

$$x^0 = 1$$

```
pow(x, 0) == 1
```

Lets give it a go in Ed!

$$
\mathrm{pow}(x, a) = \begin{cases} 1 & \text{// if } a == 0 \\ \mathrm{pow}(x*x,\ a/2) & \text{// if } a \text{ is even} \\ x * \mathrm{pow}(x,\ a-1) & \text{// if } a \text{ is odd} \end{cases}
$$