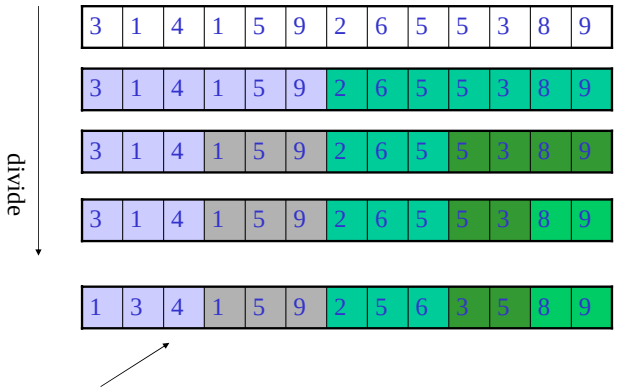
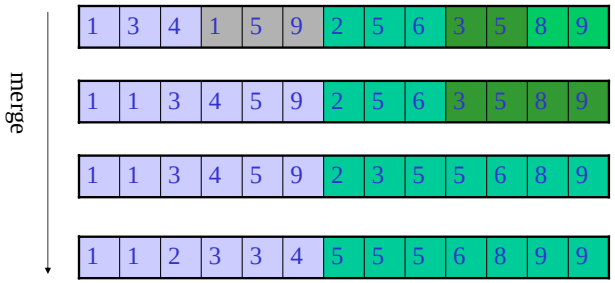
Merge Sort

- Sort two subarrays
- Merge the sorted array into one sorted array
  - Need additional storage
    - Compare with insertion and selection sort

Merge sort: top down



Merge sort: top down

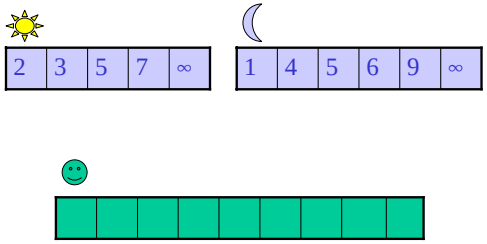


Merge two sorted arrays

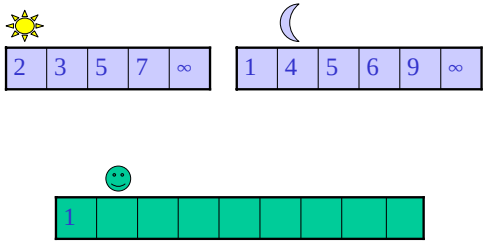
```
Merge(U[m], V[n], T[m+n])
// merge sorted arrays U and V into T
{
   u = 0;   // cursor for U
   v = 0;   // cursor for V
   U[m] = V[n] = +∞; // sentinels
   for (t=0; t<m+n; t++) {  // t is cursor for T
     if (U[u] < V[v]) {
          T[t] = U[u];
          u++;
     } else {
         T[t] = V[v];
         v++;
     }
   }
}
```

What to do if we do not use the two sentinels?

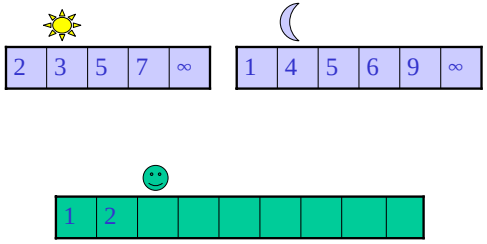
Merge two arrays



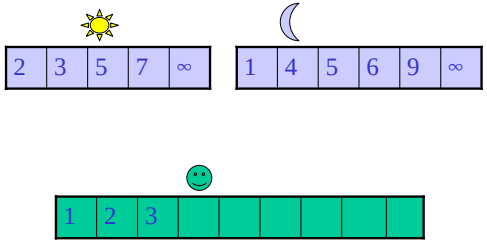
Merge two arrays



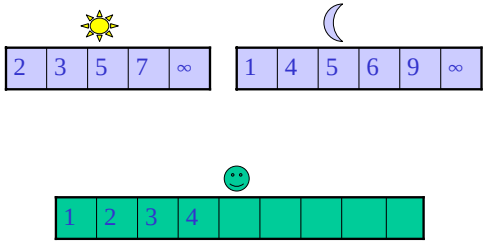
Merge two arrays



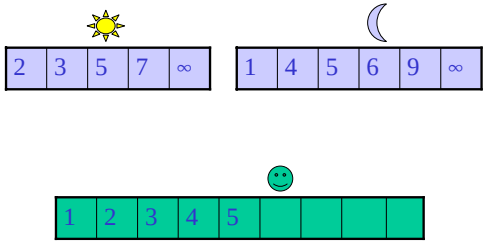
Merge two arrays



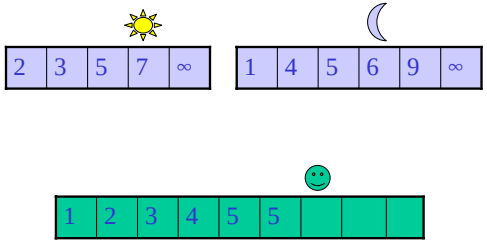
Merge two arrays



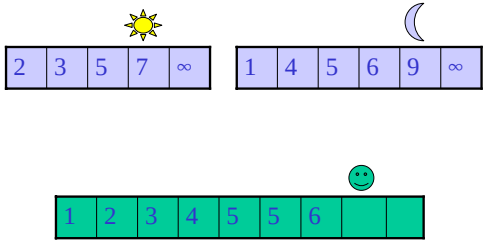
Merge two arrays



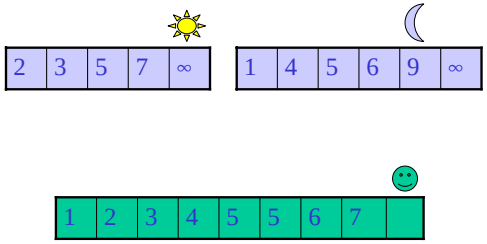
Merge two arrays



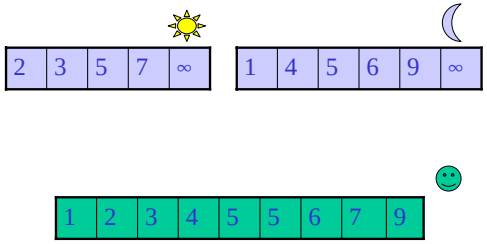
Merge two arrays



Merge two arrays



Merge two arrays



Merge sort

```
mergeSort(int T[n])
{
   if (n is sufficiently small)
      insertionSort(T);
   else {
      int U[⌊ n/2 ⌋], V[⌈ n/2 ⌉];
      copy T[1.. ⌊ n/2 ⌋] to U[1.. ⌊ n/2 ⌋];
      copy T[⌊ n/2 ⌋+1..n] to V[1..⌈ n/2 ⌉];
      mergeSort(U[1.. ⌊ n/2 ⌋]);
      mergeSort(V[1..⌈ n/2 ⌉);
      merge(U,V,T);
   }
}
```

Merge sort

```
mergeSort(int T[n])
{
   if (n is sufficiently small)
      insertionSort(T);
   else {
      int U[n/2], V[(n+1)/2];
      copy T[1..n/2] to U[1..n/2];
      copy T[n/2+1..n] to V[1.. (n+1)/2];
      mergeSort(U[n/2]);
      mergeSort(V[(n+1)/2);
      merge(U,V,T);
   }
}
```

Note: $\lfloor (n+1)/2 \rfloor = \lceil n/2 \rceil$

Cost

- Storage
- Execution time

$$t(n) = t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + \Theta(n)$$

A bottom up algorithm

```
mergeSort(T[n])
{   int len = 1;
    boolean direction = true;
    int U[n];

    while (len < n) {
       for (i=0; i+len<n; i+=2*len) {
            if (direction)
              merge(T[i..i+len-1], T[i+len, min(n-1, i+2*len-1)], U[i]);
            else
              merge(U[i..i+len-1], U[i+len, min(n-1, i+2*len-1)], T[i]);
       }
       direction = !direction;
       len * = 2;
    }
    if (direction)
       copy(U,T);
}
```


Cost

- Storage
- Execution time

$$t(n) = t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + \Theta(n)$$