CE2101/ CZ2101: Algorithm Design and Analysis

Mergesort

Ke Yiping, Kelly

Learning Objectives

At the end of this lecture, students should be able to:

- Explain the approach of Divide and Conquer
- Describe how Mergesort works by:
  - Recalling the pseudo code
  - Manually executing the algorithm on a toy input array
- Analyse the time complexity of Mergesort, by using:
  - Recurrence equation
  - Recursion tree

NANYANG TECHNOLOGICAL UNIVERSITY

通常用递归 recursive function 来解决

The Divide and Conquer approach

usually, k=2

为什么不把k变得更大? 即多分成几份

原因: 1. n分2份 → $\log_2 n$

2. n分k份 → $\log_k n$

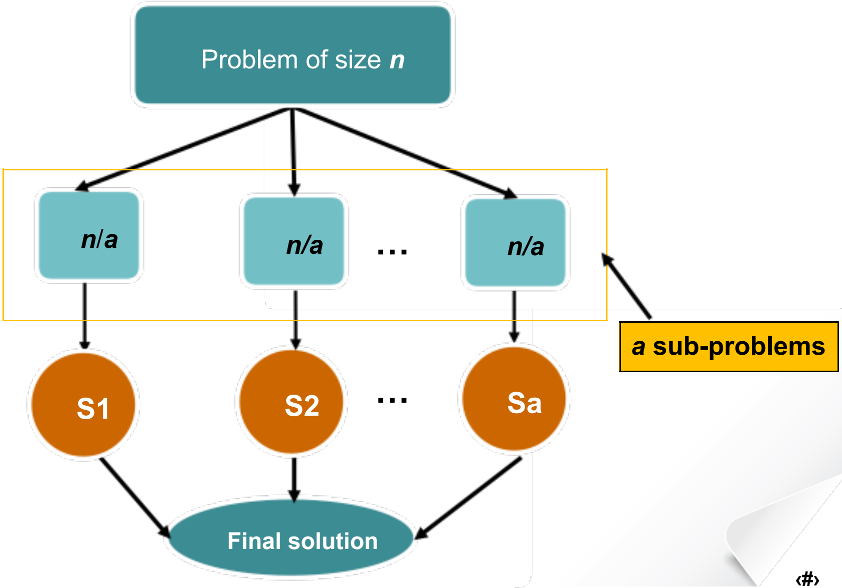
$$\log_k n = \frac{\log_2 n}{\boxed{\log_k 2}} \rightarrow \text{constant.}$$

3. 效率变化不明显, 而combine的成本明显变大.

The skeleton of this approach:

```
solve (problem of size n)
{   if (n <= minimum size)
        solve the problem directly;
    else {
            divide the problem into p1, p2, … , pk;
            for each sub-problem ps
                solutions = solve (ps);
            combine all solutions;
    }
}
```

Mergesort


Problem of size *n*
n/*a*
n/*a*
…
n/*a*
S1
S2
…
Sa
Final solution
a sub-problems

Mergesort (Algorithm)

NANYANG TECHNOLOGICAL UNIVERSITY

Mergesort (Algorithm)

排序的基本base case都是这俩

Base case: $n=0$ or $n=1$

return

```
mergeSort(list) {
    if (length of list > 1) {
        Partition list into two (approx.) equal sized
            lists, L1 & L2;
        mergeSort (L1);
        mergeSort (L2);
        merge the sorted L1 & L2;
    }
}
```

all dirty work done in the function

Mergesort

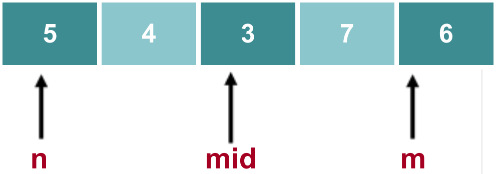
index

```
void mergesort(int n, int m)
{   int mid = (n+m)/2;
   if (m-n <= 0)
         return;
   else if (m-n > 1) {
      mergesort(n, mid);
      mergesort(mid+1, m);
   }
   merge(n, m);
}
```

不需要进入 recursive call

m-n =1 会直接跳到这一步.

5 | 4



if m-n = 0, 5

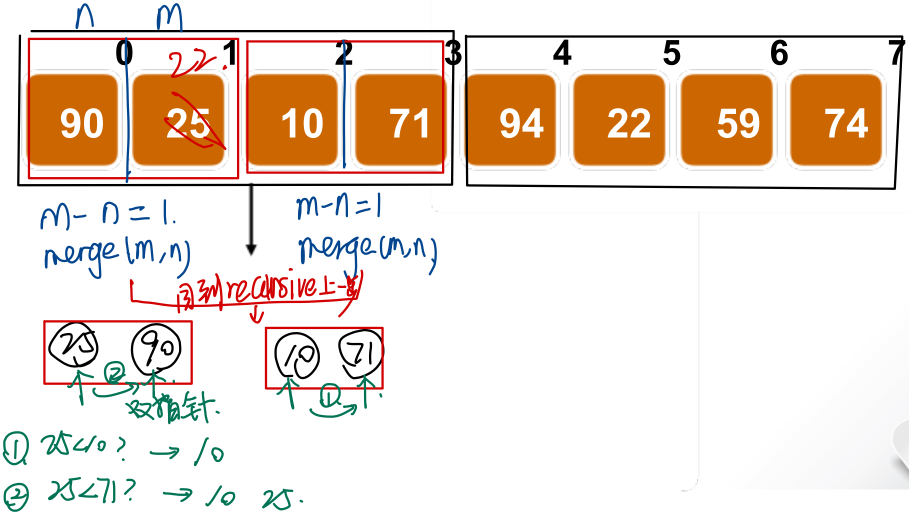
m = n

if m-n < 0, **Empty array**

问：如果列表有重复？
指针同时后移.

Mergesort

Sort in ascending order



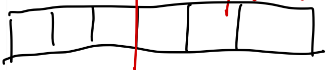
Merge (Pseudo Code)

NANYANG TECHNOLOGICAL UNIVERSITY

Merge (Pseudo Code)

为什么两边都排好了,还要分?

conceptual → 分隔两边列表



```
void merge(int n, int m) {
  if (m-n <= 0) return;
  divide the list into 2 halves;  // both halves are sorted
    while (both halves are not empty) {
          compare the 1st elements of the 2 halves;  // 1 comparison
    if (1st element of 1st half is smaller)
          1st element of 1st half joins the end of the merged list;
    else if (1st element of 2nd half is smaller)
          move the 1st element of 2nd half to the end of the
      merged list;
```

?

?

NANYANG TECHNOLOGICAL UNIVERSITY

Merge (Pseudo Code)

(tie).

```
else {   // the 1st elements of the 2 halves are equal
            if (they are the last elements)   break;
          1st element of 1st half joins end of the merged list;
          move the 1st element of 2nd half to the end of the
   merged list;
   }
  } // end of while loop;
} // end of merge
```

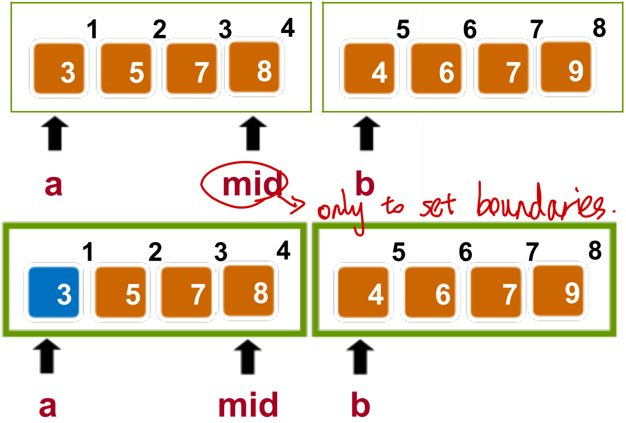
Challenge:
How to do it without auxiliary
storage for the merged list?

Merge (Case Scenarios)

(without additional space)

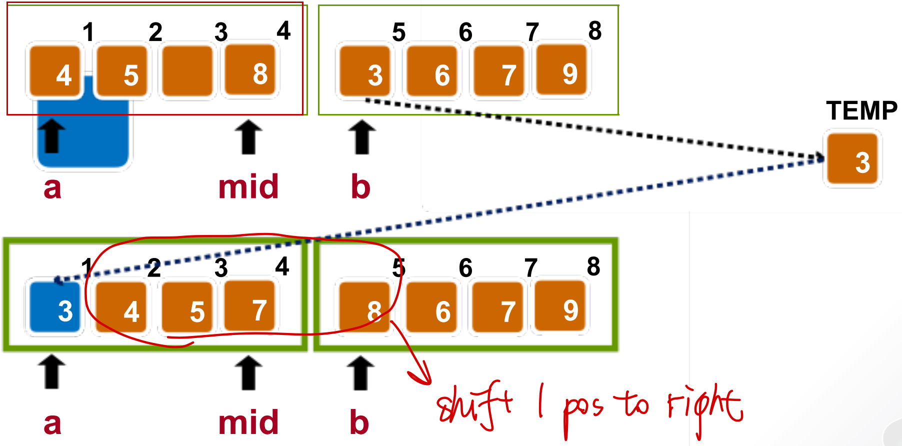
Merge (Case Scenarios)

Case 1: 1st element of 1st half is smaller



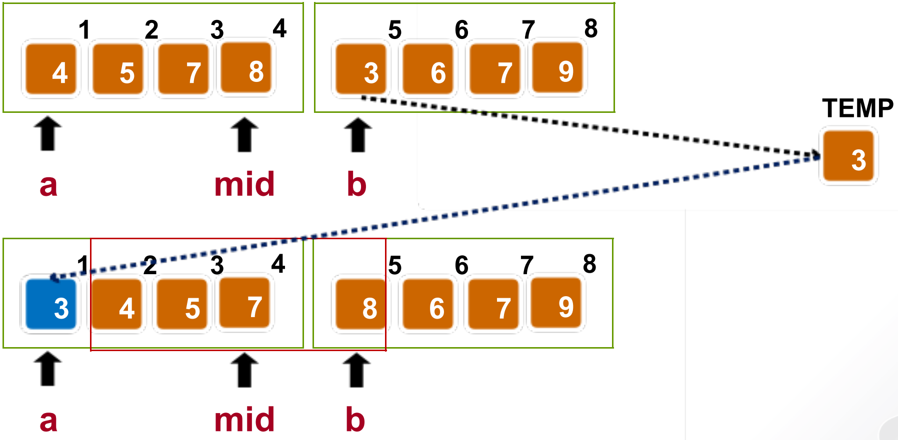
Merge (Case Scenarios)

Case 2: 1st element of 2nd half is smaller



Merge (Case Scenarios)

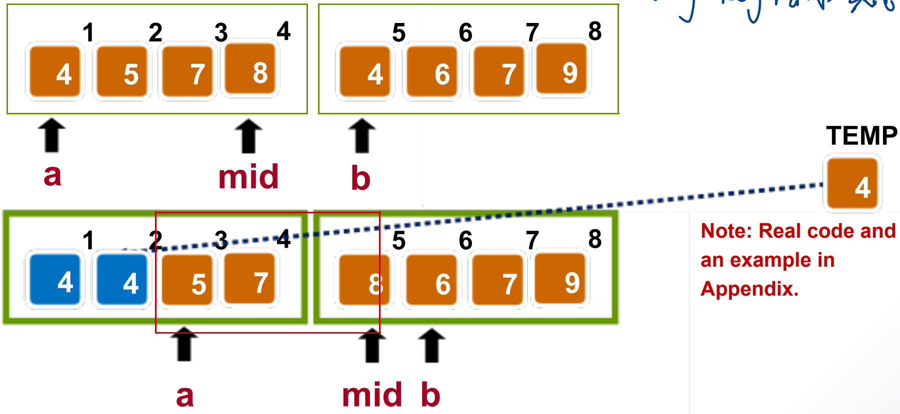
Case 2: 1st element of 2nd half is smaller



Merge (Case Scenarios)

Case 3: 1st element of 2nd half is equal → cannot move and copy

(only key相同, 其他可能会不一样



Note: Real code and an example in Appendix.

Complexity:

Mergesort Algorithm (Recap)

Mergesort Algorithm (Recap)

- Since merging is performed directly on the original array, swapping and shifting are needed
- mergesort() partitions a contiguous array of elements between index n and m into two subarrays

```
void mergesort(int n, int m)
{   int mid = (n+m)/2;
   if (m-n <= 0)
      return;
   else if (m-n > 1) {
      mergesort(n, mid);
      mergesort(mid+1, m);
   }
   merge(n, m);
}
```

NANYANG TECHNOLOGICAL UNIVERSITY

Mergesort Algorithm (Recap)

- Since merging is performed directly on the original array, swapping and shifting are needed
- mergesort() partitions a contiguous array of elements between index n and m into two subarrays
- Recursively partitions until m-n<=0, then merge the resulting two subarrays

```
void mergesort(int n, int m)
{   int mid = (n+m)/2;
    if (m-n <= 0)
        return;
    else if (m-n > 1)
    …..
}
```

Mergesort Algorithm (Recap)

- Since merging is performed directly on the original array, swapping and shifting are needed
- mergesort() partitions a contiguous array of elements between index n and m into two subarrays
- Recursively partitions until m-n<=0, then merge the resulting two subarrays
- merge() function merges two sub-arrays of elements between index n and 'mid', and between 'mid+1' and m

```
void mergesort(int n, int m)
{   int mid = (n+m)/2;
   if (m-n <= 0)
  ........
  merge(n, m);
}
```

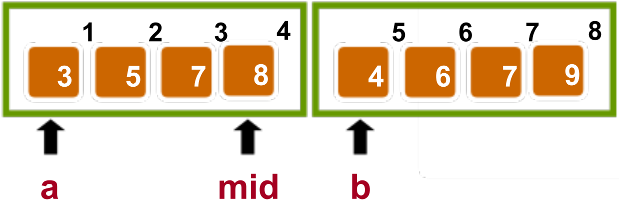
Mergesort Algorithm (Recap)

- Since merging is performed directly on the original array, swapping and shifting are needed
- mergesort() partitions a contiguous array of elements between index n and m into two subarrays
- Recursively partitions until m-n<=0, then merge the resulting two subarrays
- merge() function merges two sub-arrays of elements between index n and 'mid', and between 'mid+1' and m
- During merging, one element from each subarray is compared and the smaller one is inserted into new list

```
void mergesort(int n, int m)
{   .....
    merge(n, m);
}
```

Mergesort Algorithm (Recap)

- Left subarray runs from n to 'mid' with a as running index; right subarray runs from mid+1 to m with b as running index

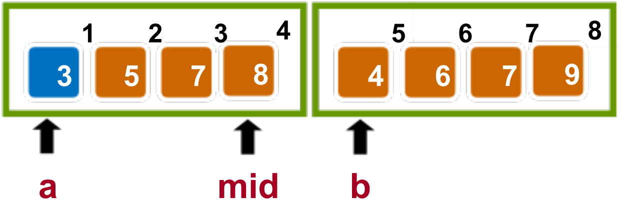


Mergesort Algorithm (Recap)

- Left subarray runs from n to ‘mid’ with a as running index; right subarray runs from mid+1 to m with b as running index
- slot[a] is the head element of left subarray, slot[b] is the head element of right subarray

Mergesort Algorithm (Recap)

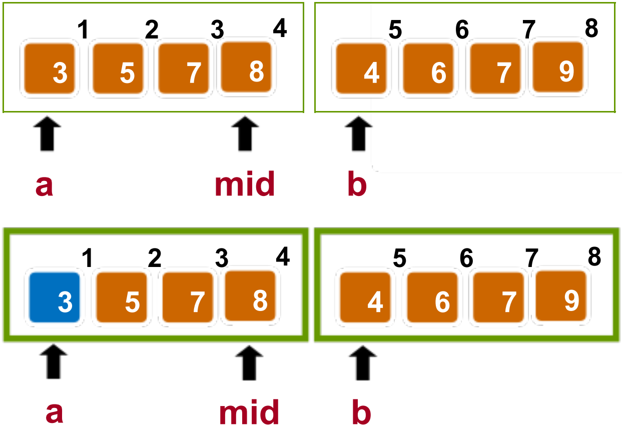
- Left subarray runs from n to 'mid' with a as running index; right subarray runs from mid+1 to m with b as running index
- slot[a] is the head element of left subarray, slot[b] is the head element of right subarray
- During merging, both left and right subarrays shrink towards the right to make space for the newly merged array



NANYANG TECHNOLOGICAL UNIVERSITY

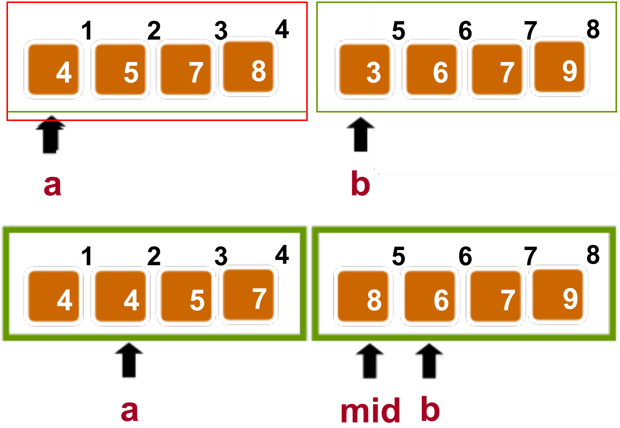
Mergesort Algorithm (Recap)

Case 1: if slot[a] < slot[b], there is nothing much to do since smaller element already in correct position (with regard to the merged array)



Mergesort Algorithm (Recap)

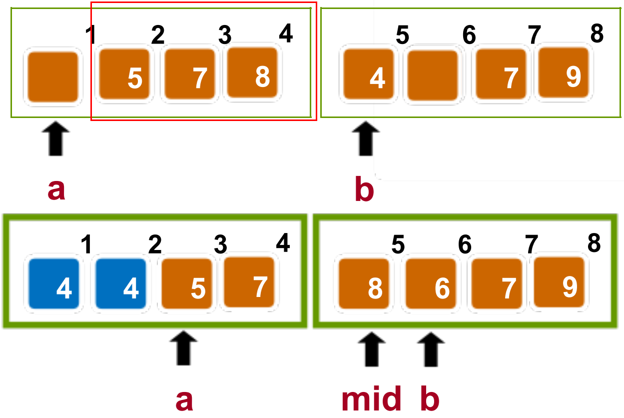
Case 2: if slot[a] > slot[b], then Right-shift (by one) elements of left subarray from index a to 'mid' and insert element at slot[b] into slot[a]



NANYANG TECHNOLOGICAL UNIVERSITY

Mergesort Algorithm (Recap)

Case 3: if slot[a] == slot[b], then slot[a] is in the correct position. So, move slot[b] next to beside slot[a], by Right-shifting and swapping



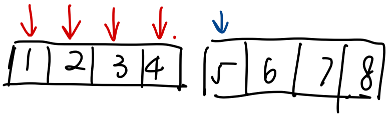
Complexity of Mergesort

Complexity of Mergesort

- Complexity of Merge Function : worst $(n-1)$, best $(\frac{n}{2})$. $\}\ O(n)$

- After **each** comparison of keys from the two sub-lists, **at least one** element is moved to the new merged list and never compared again
- After the **last** key comparison, at least **two** elements will be moved into the merged list
- Thus, to merge two sub-lists of n elements in total, the number of key comparisons needed is at most $n - 1$

at least $= \frac{n}{2}$.



Complexity of Mergesort

∴ Complexity of Mergesort.

```
void mergesort(int s, int e) // s=start, e=end
{   int mid = (s+e)/2;
    if (e-s <= 0) return;
    else if (e-s > 1) {
        mergesort(s, mid);
        mergesort(mid+1, e);
    }
    merge(s, e);
}
```

time complexity.

$\longrightarrow$ $\boxed{W}(1) = 0$

$\longrightarrow$ W(n/2)

$\longrightarrow$ W(n/2)

W(n)

$\longrightarrow$ Worst case: n-1

NANYANG TECHNOLOGICAL UNIVERSITY

Complexity of Mergesort

Mergesort performance (assume n = 2k)

k = lg n

Worst case :

first half, second half, merge.

W(1) = 0,

W(n) = $\boxed{W(n/2)}$ + $\boxed{W(n/2)}$ + $\boxed{n-1}$ Or

W(2k) = 2W(2k-1) + 2k -1

= 2(2W(2k-2) + 2k-1 -1) + 2k -1

= 22W(2k-2) + 2k -2 + 2k -1

= 22(2W(2k-3)+ 2k-2 -1) + 2k -2 + 2k -1

= 23W(2k-3)+ 2k -22 + 2k -2 + 2k -1

…

= 2kW(2k-k) + k2k – (1 + 2 + 4 + … + 2k-1)

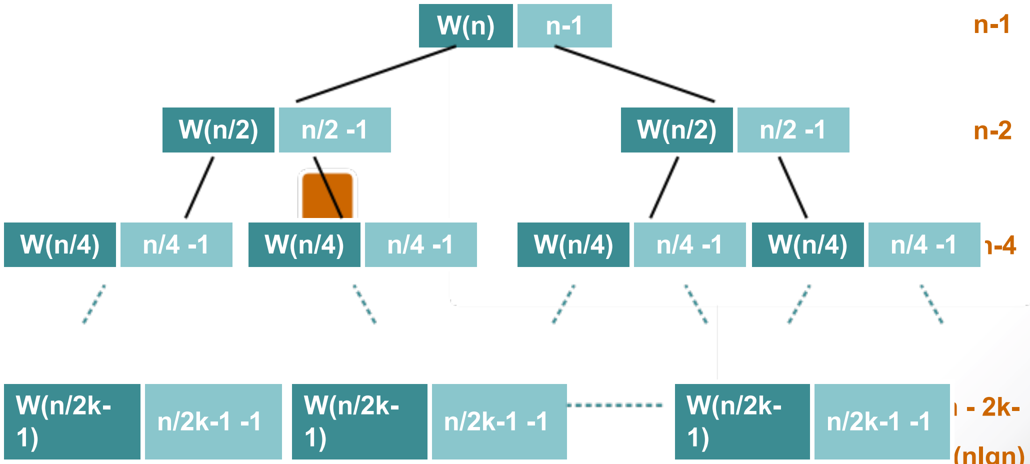
= k2k – (2k – 1)

= n lg n – (n – 1)

= O(n lg n)

Geometric series

Visually :Recursion Tree



W(2) = 2W(1) + 1 = 1

Height of tree is $k = O(\lg_2 n)$

Evaluation of Mergesort

- **Strengths:**
  - Simple and good runtime behavior
  - Easy to implement when using linked list
- **Weaknesses:**
  - Difficult to implement for contiguous data storage such as array without auxiliary storage (requires data movements during merging)

NANYANG TECHNOLOGICAL UNIVERSITY

Summary

- Mergesort uses the Divide and Conquer approach.
- It recursively divide a list into two halves of approximately equal sizes, until the sub-list is too small (no more than two elements).
- Then, it recursively merges two sorted sub-lists into one sorted list.
- The worst-case running time for merging two sorted lists of total size n is $n - 1$ key comparisons.
- The running time of Mergesort is $O(n\lg n)$.

CE2101/ CZ2101: Algorithm Design and Analysis

Appendix

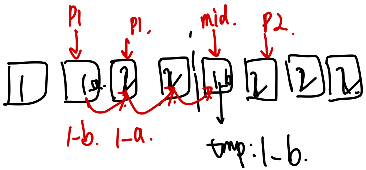
(Merge operation in Mergesort)

Ke Yiping, Kelly

NANYANG TECHNOLOGICAL UNIVERSITY

Merge Function

```
void merge(int n, int m)
  {
    int mid = (n+m)/2;
     int a = n, b = mid+1, i, tmp;
     if (m-n <= 0) return;
    while (a <= mid && b <= m) {
     cmp = compare(slot[a], slot[b]);
     if (cmp > 0) { //slot[a] > slot[b]
                tmp = slot[b++];
              for (i = ++mid; i > a; i--)
                 slot[i] = slot[i-1];
```



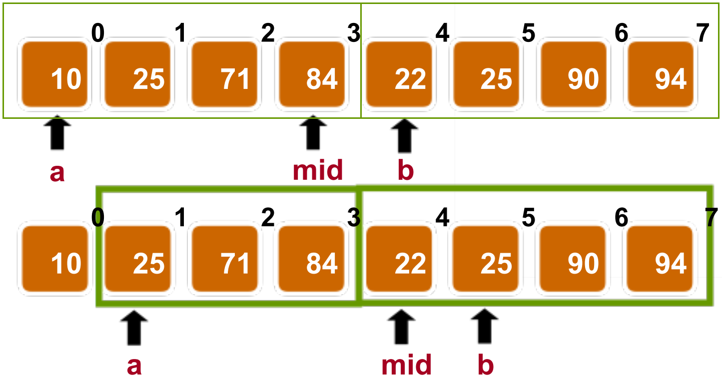
Question:

Why is merge sort stable?

Merge Function

```
            slot[a++] = tmp;
    } else if (cmp < 0) //slot[a] < slot[b]
        a++;
    else {   //slot[a] == slot[b]
            if (a == mid && b == m)
            break;
            tmp = slot[b++];
            a++;
            for (i = ++mid; i > a; i--)
            slot[i] = slot[i-1];
            slot[a++] = tmp;
        }
    } // end of while loop;
} // end of merge
```

Merge Operation



Parameters for merge:

n:0, m: 7

mid = (0+7)/2 = 3;

a = n; **b** = mid+1;

Comparison:

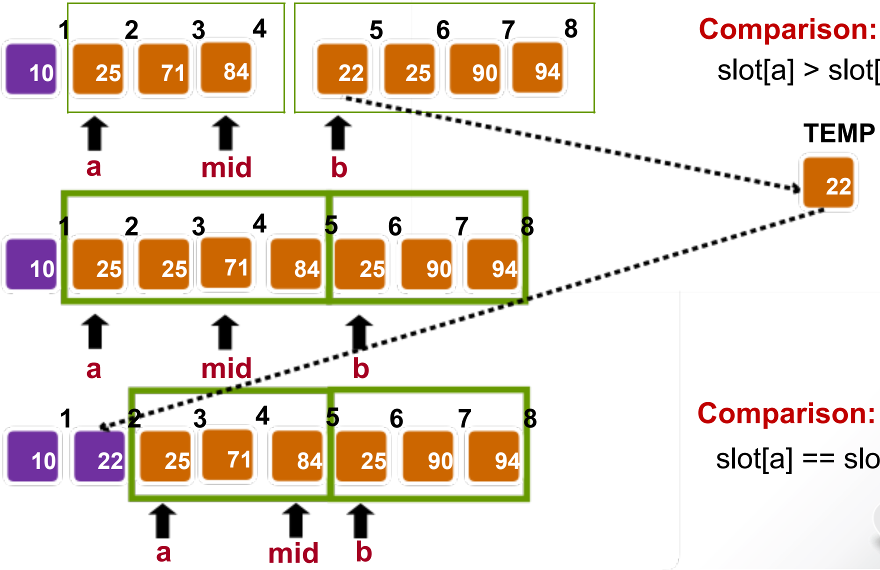
slot[a] < slot[b]

a : the 1st element of the 1st half

mid : the last element of the 1st half

b : the 1st element of the 2nd half

Merge Operation



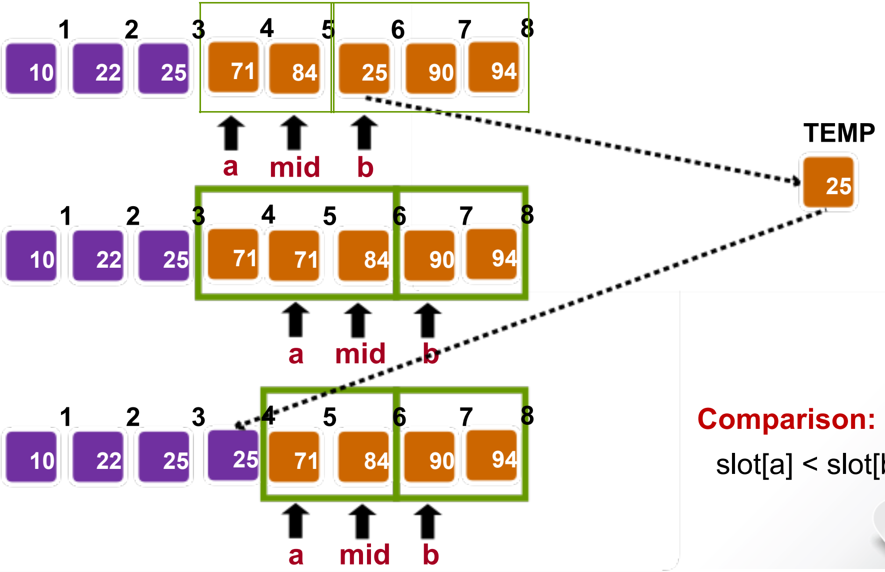
Comparison:

slot[a] > slot[b]

Comparison:

slot[a] == slot[b]

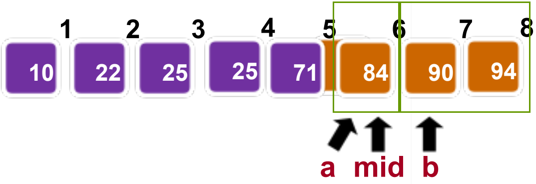
Merge Operation



Comparison:

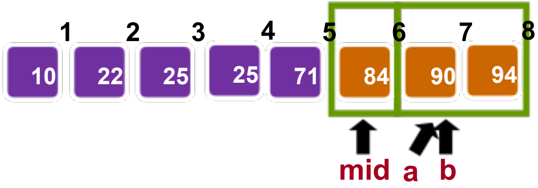
slot[a] < slot[b]

Merge Operation



Comparison:

slot[a] < slot[b]



1st half
empty

Merge operation completed