Divide and Conquer

Truong Ngoc Tuan

Nội dung

- Ý tưởng chia để trị
- Lược đồ chung
- Merge sort
- Binary Search
- Quick sort

Nội dung

- Ý tưởng chia để trị
- Lược đồ chung
- Merge sort
- Binary Search
- Quick sort

Ý tưởng chia để trị

https://doctailieu.com/soan-bai-ke-chuyen-cau-chuyen-bo-dua

Ý tưởng chia để trị

- Bài học từ cuộc sống: chia nhỏ bó đũa để dễ bẻ hơn
- Ý tưởng cơ bản: chia nhỏ bài toán lớn thành các bài toán con để có thể tìm lời giải dễ dàng hơn

Ý tưởng chia để trị

- Là một phương pháp được áp dụng rộng rãi
- Giải các bài toán con theo cùng 1 cách thức
- "Tổng hợp" lời giải các ài toán con để có được kết quả bài toán ban đầu

➡ Tư tưởng chung: Chia để trị

Nội dung

- Ý tưởng chia để trị
- Lược đồ chung
- Merge sort
- Binary Search
- Quick sort

Lược đồ chung

❑ Lược đồ chung

- Chia:
  - Chia tập hợp các đối tượng của bài toán thành bàu toán con "độc lập"
  - Tiếp tục chia bài toán con cho đến khi có thể giải trực tiếp (không cần, hoặc không thể chia nhỏ được nữa)
- Trị:
  - Trên các bài toán con thực hiện cùng một cách thức: chia nhỏ nếu cần hoặc giải trực tiếp
- Tổng hợp:
  - Khi mỗi bài toán con được giải, tổng hợp để có kết quả bài toán ban đầu

Lược đồ chung

Gọi R là miền dữ liệu

```
void D_and_C(R) {
    if (R đủ nhỏ) {
        giải bài toán
    } else {
        chia R thành R1…Rm
        for (i=1; i<=m; i++)
            D_and_C(Ri)
        Tổng hợp kết quả
    }
}
```

Nội dung

- Ý tưởng chia để trị
- Lược đồ chung
- Merge sort
- Binary Search
- Quick sort

Merge Sort

Bài toán

- Cho mảng gồm n phần tử A[1..n], sắp xếp mảng A theo thứ tự tăng dần

Merge Sort

❑ Ý tưởng

- Nếu có hai dãy a và b đã được sắp xếp, tiến hành trộn hai dãy này thành dãy c đã được sắp xếp
- Nếu chia nhỏ mảng cần sắp xếp thành các đoạn 1 phần tử thì nó là đoạn được sắp xếp
- Tiến hành ghép các đoạn nhỏ thành các đoạn lớn đã được sắp xếp

Merge Sort

- Cài đặt

```
MERGE_SORT (S, C) {
    if (S.size() > 1) {
        (S1, S2) ← partition(S, n/2)

        MERGE_SORT(S1, C)
        MERGE_SORT(S2, C)

        S ← MERGE(S1, S2)
    }
}
```

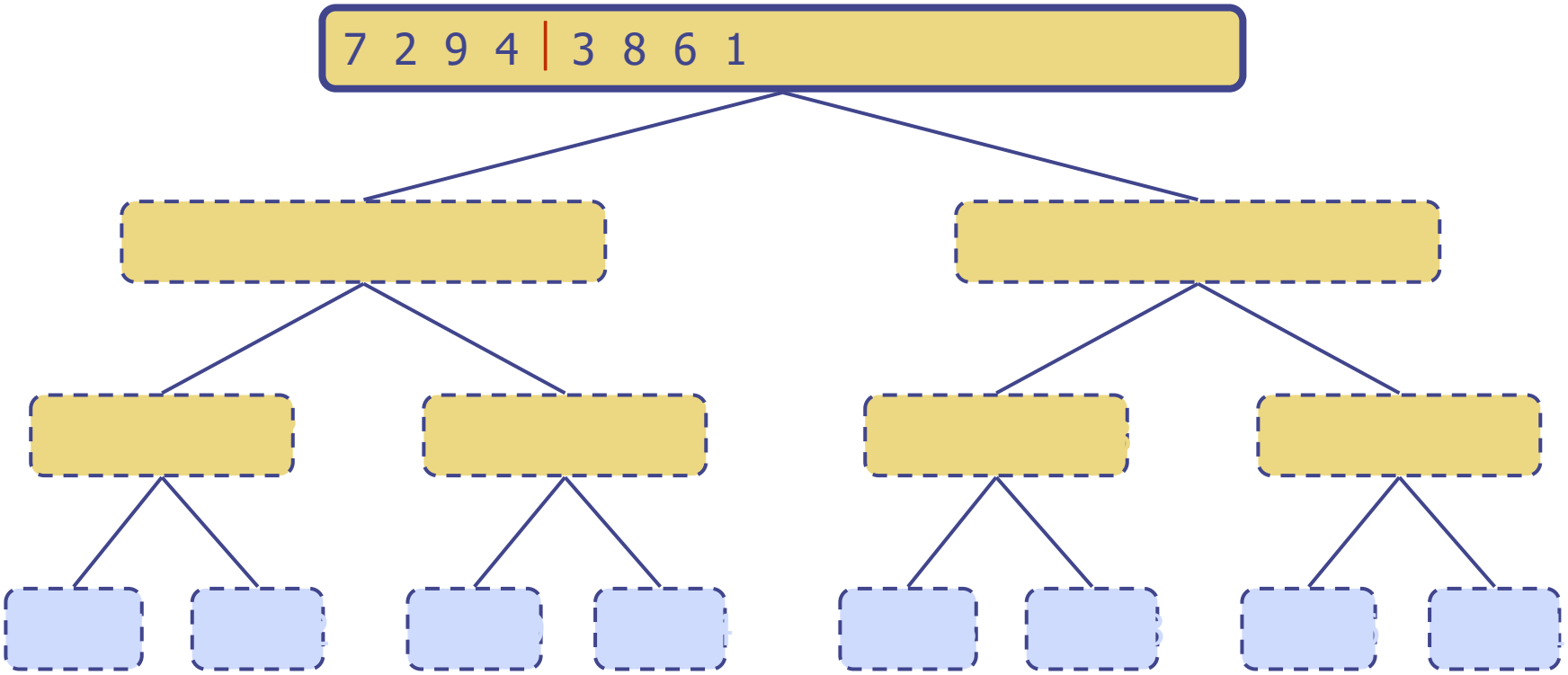
Merge Sort

```
MERGE (A, B) {
    while (!A.empty() || !B.empty())
        if (A.front() < B.front())
            S.addBack(A.front()); A.eraseFront();
        else
            S.addBack(B.front()); B.eraseFront();
    while (!A.empty())
        S.addBack(A.front()); A.eraseFront();
    while (!B.empty())
        S.addBack(B.front()); B.eraseFront();

    return S;
}
```

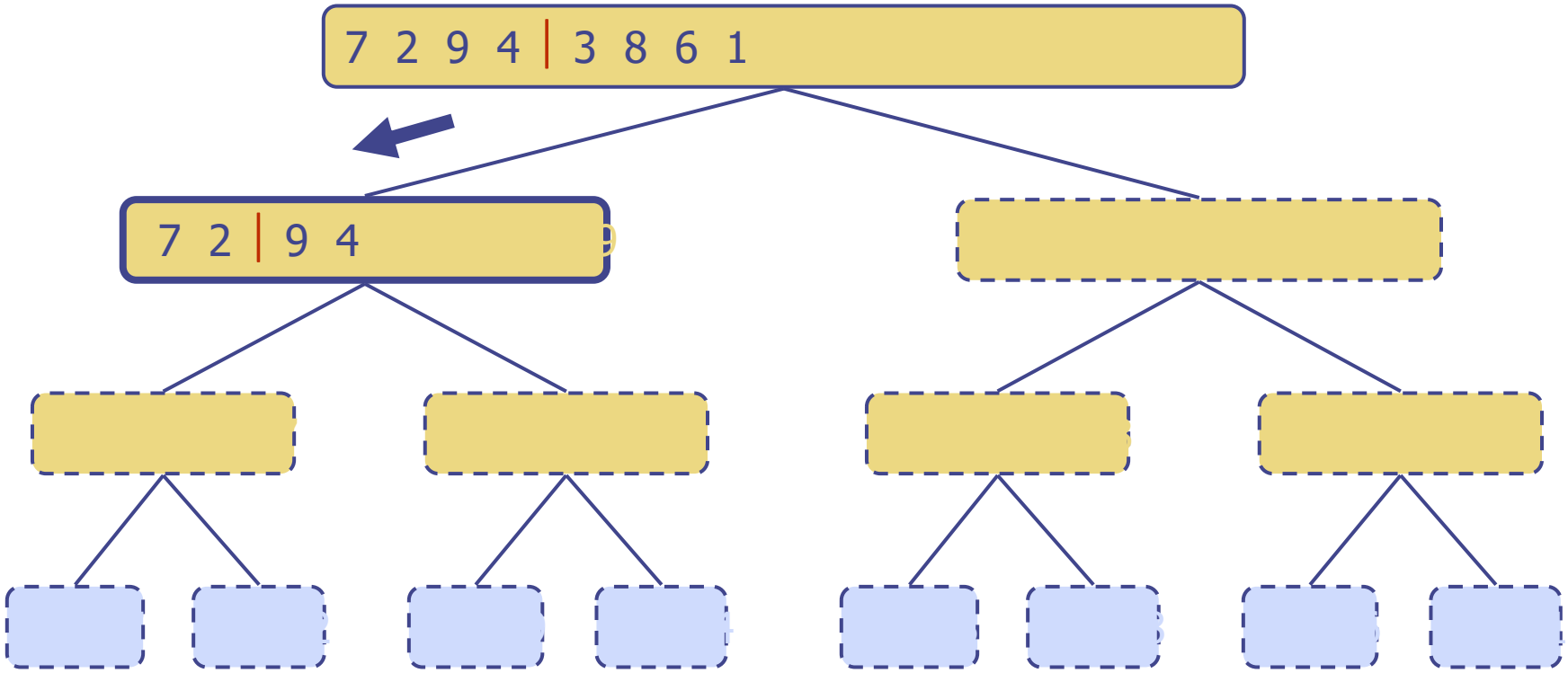
Merge Sort

- Partition



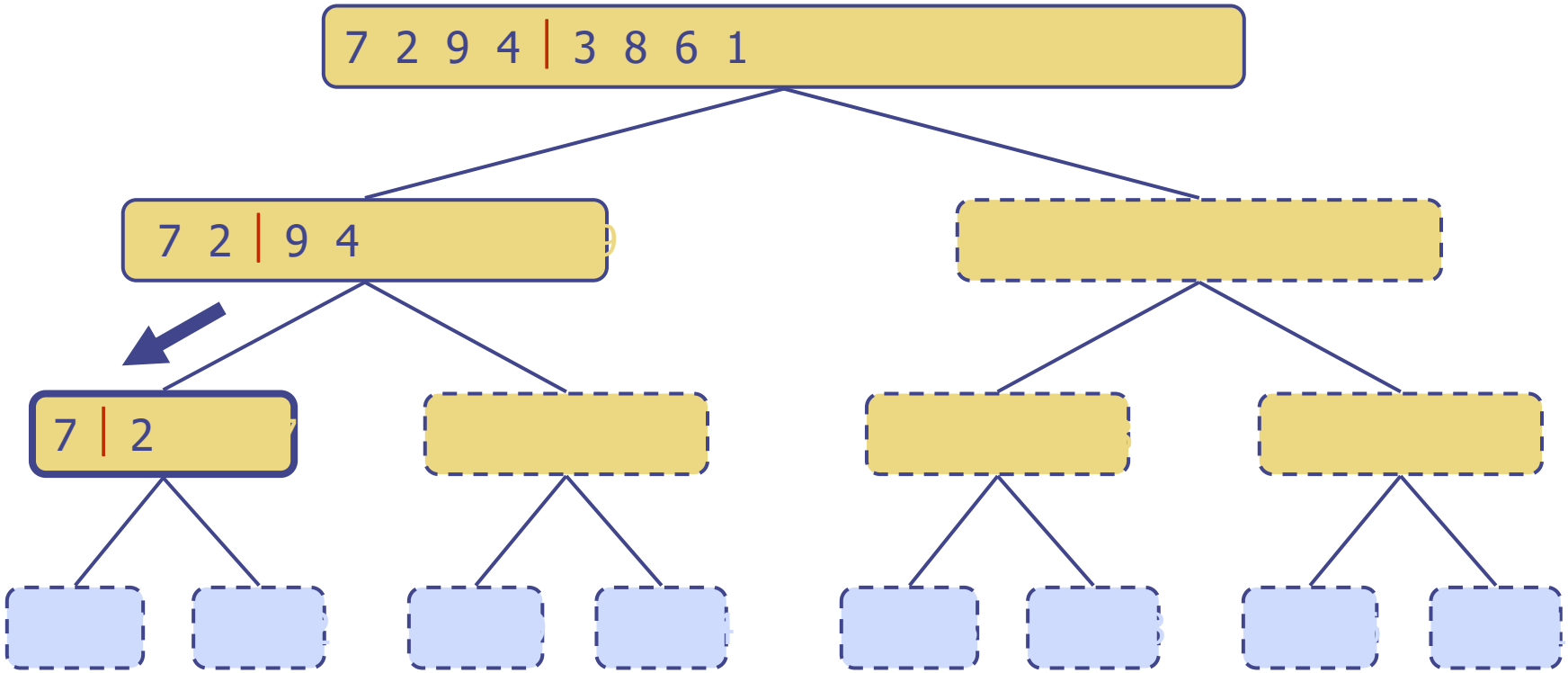
Merge Sort

- Recursive call, partition



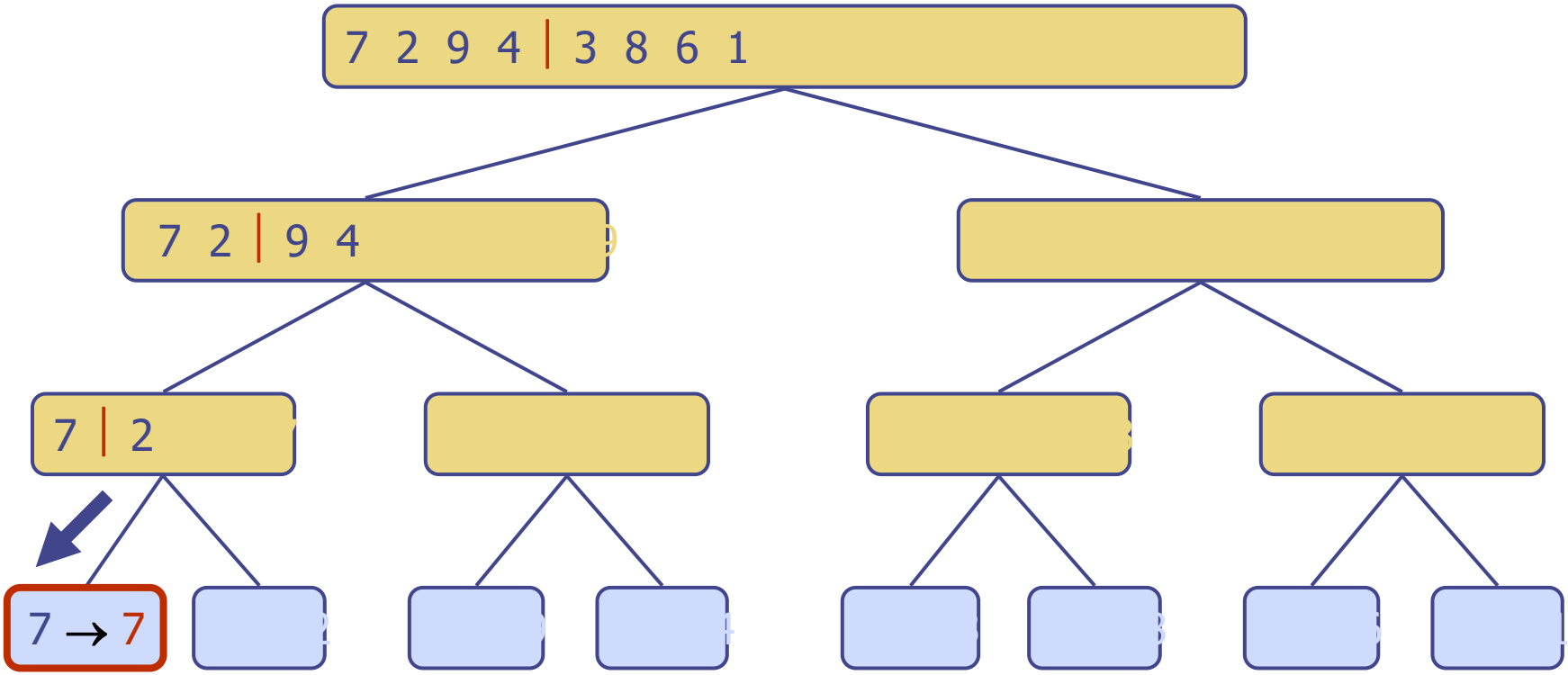
Merge Sort

- Recursive call, partition



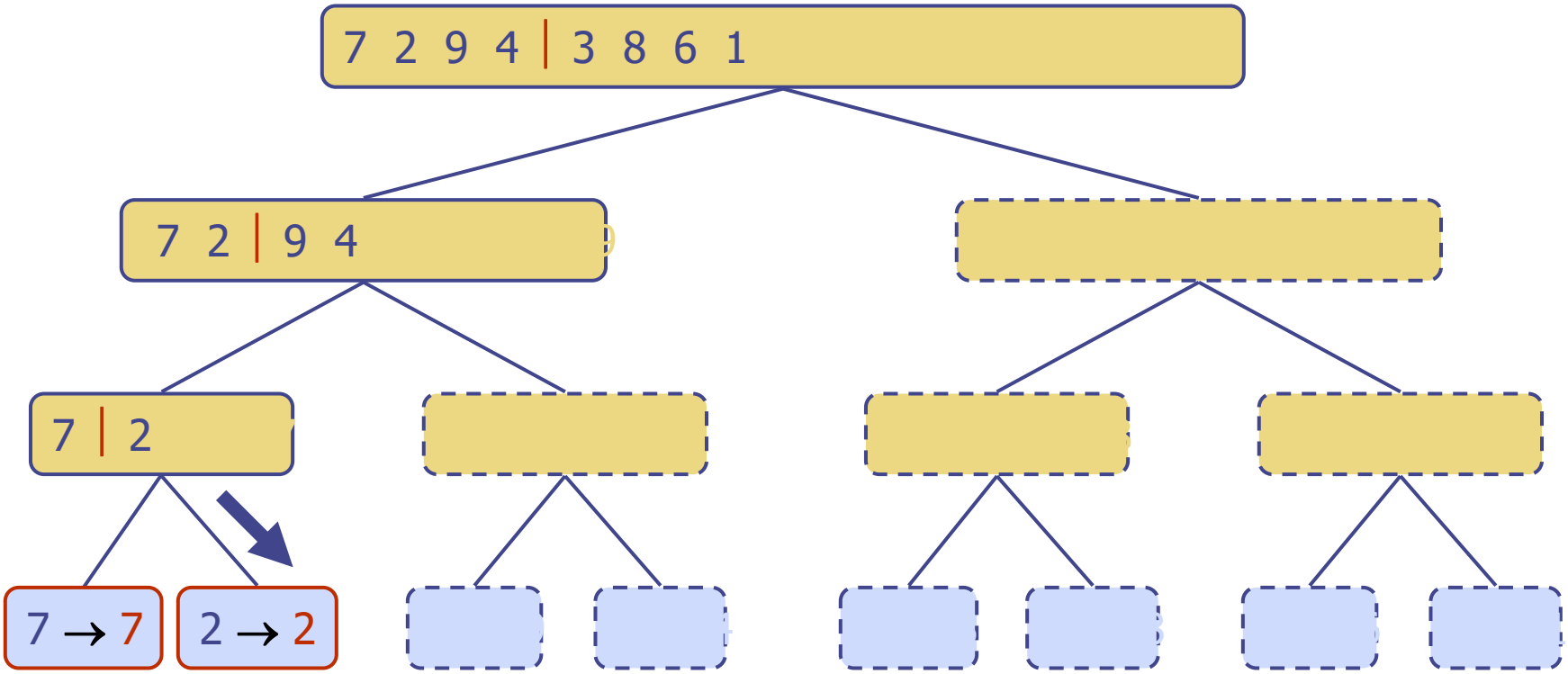
Merge Sort

- Recursive call, base case



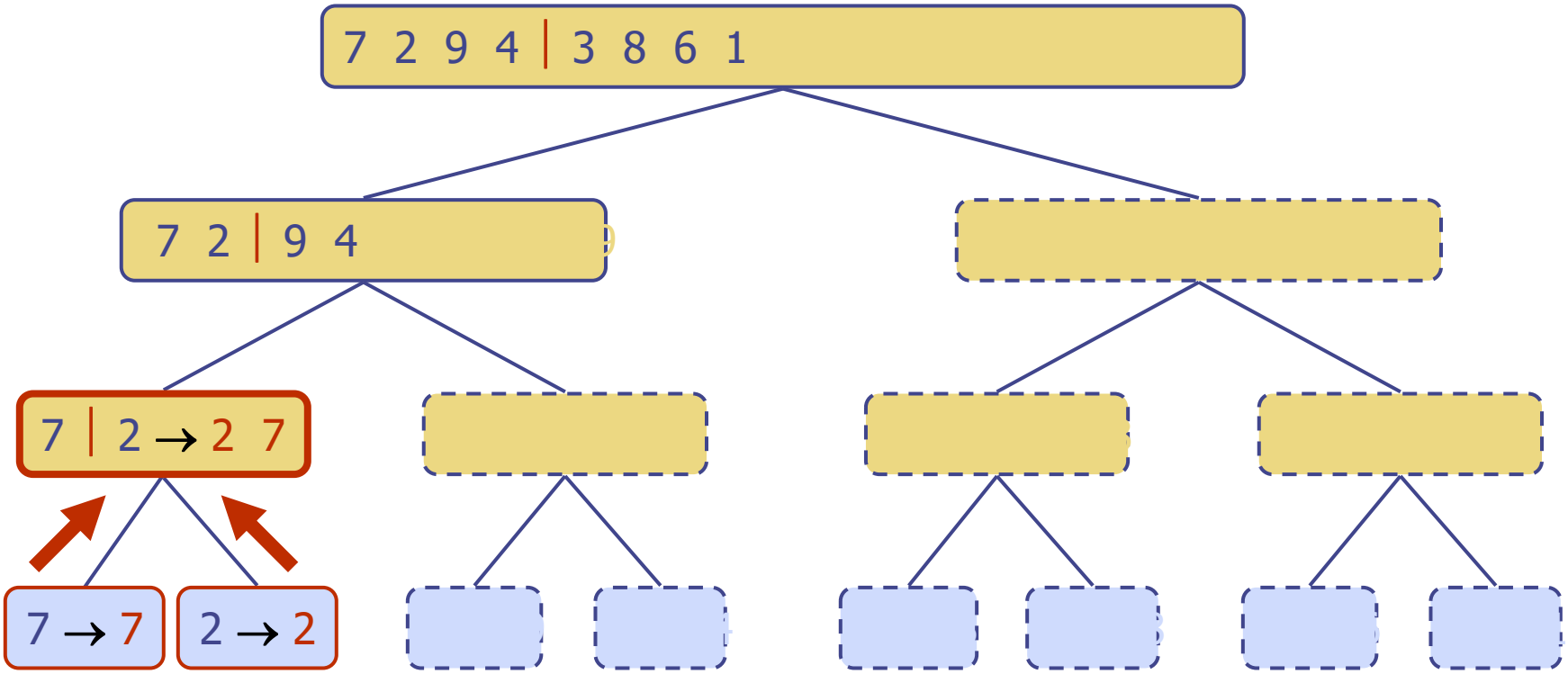
Merge Sort

- Recursive call, base case



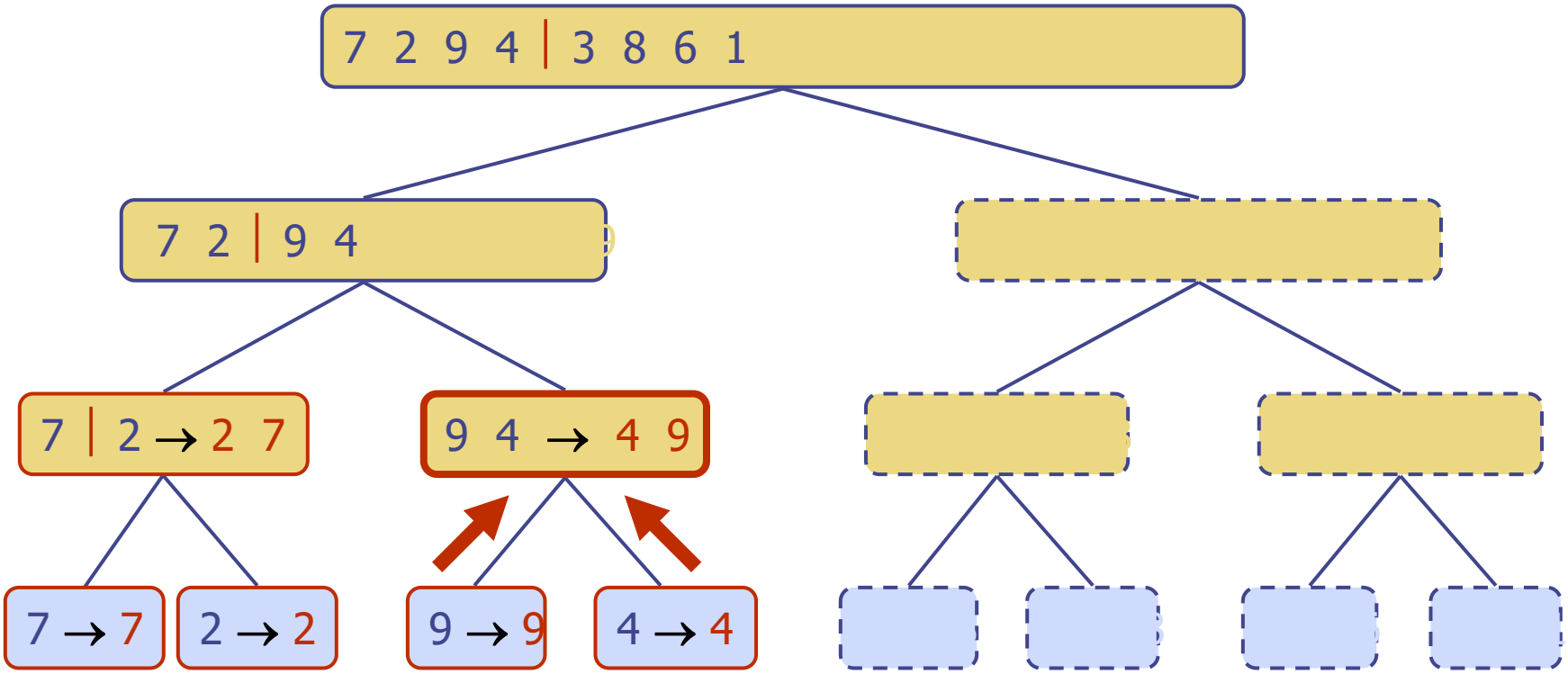
Merge Sort

- Merge



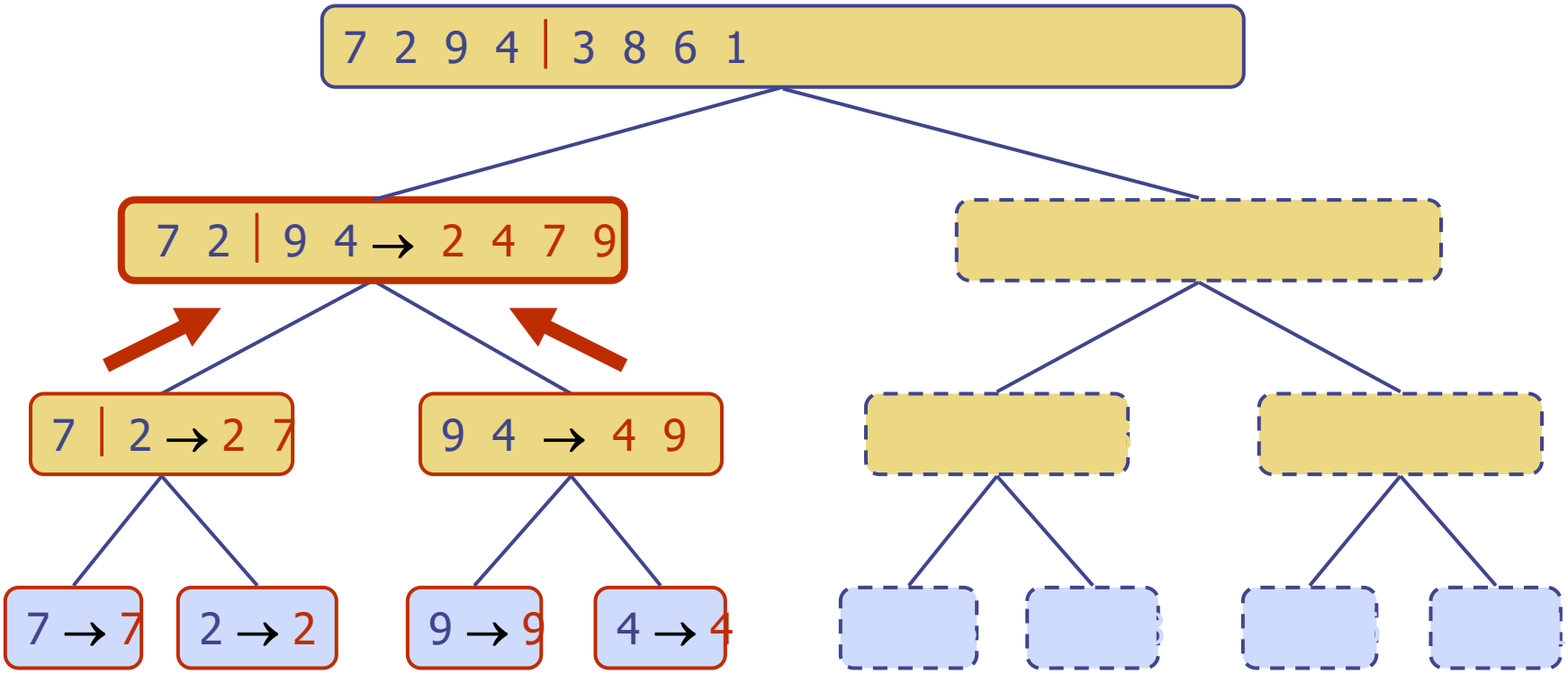
Merge Sort

- Recursive call, …, base case, merge



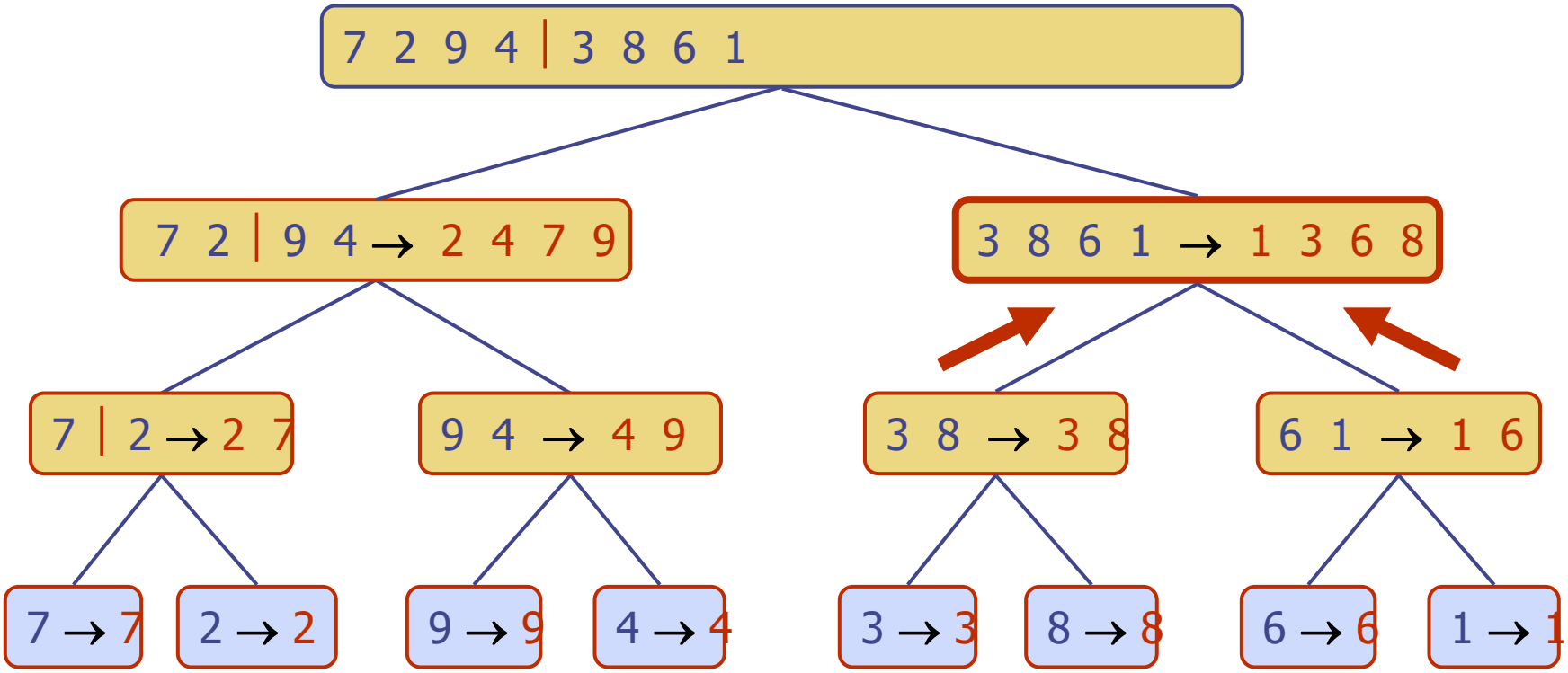
Merge Sort

- Merge



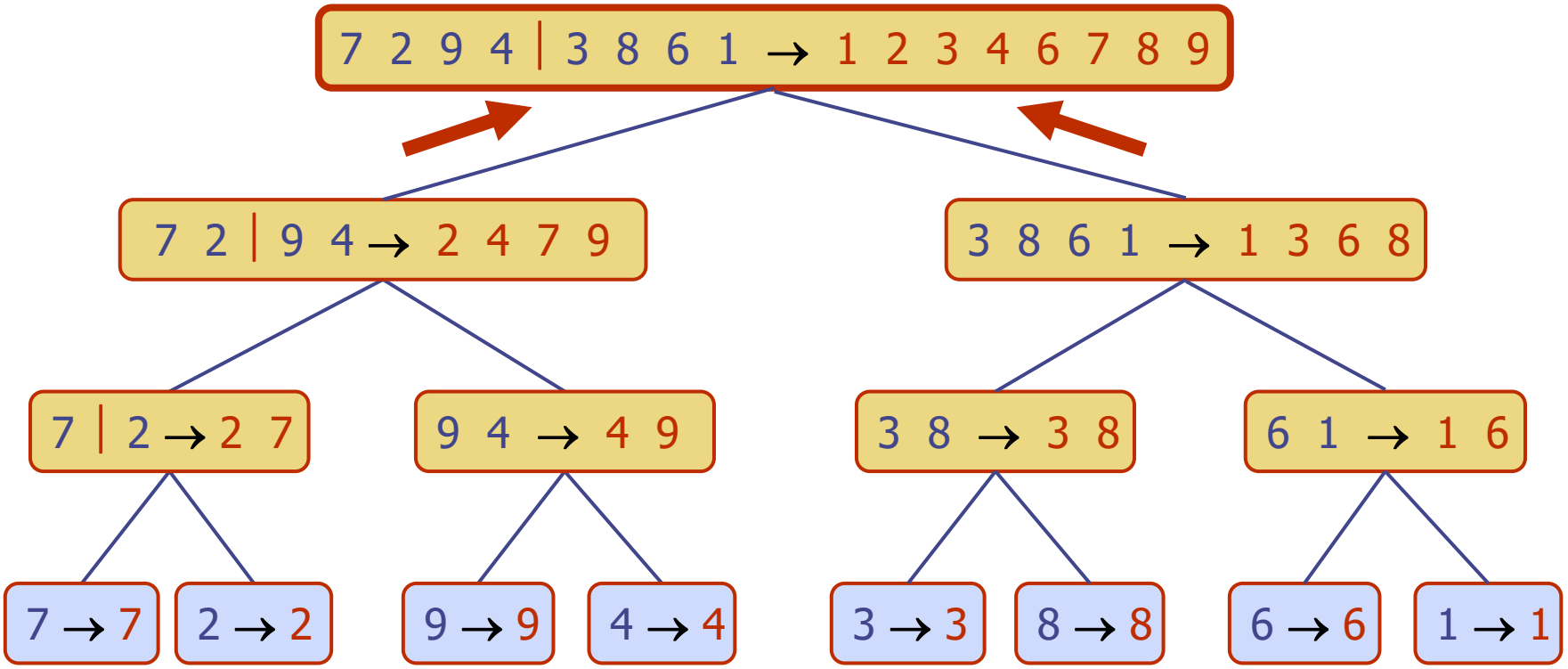
Merge Sort

- Recursive call, …, merge, merge



Merge Sort

- Merge



Nội dung

- Ý tưởng chia để trị
- Lược đồ chung
- Merge sort
- Binary Search
- Quick sort

Binary Search

❑ Bài toán

- Cho mảng gồm n phần tử A[1..n], đã được sắp xếp theo thứ tự tăng dần. Kiểm tra xem dãy A có tồn tại phần tử có giá trị ***x*** không?

Binary Search

❑ Ý tưởng

- Nếu mảng không có phần tử nào thì *return false*
- ***So sánh x với phần tử ở giữa mảng***
- Nếu bằng nhau thì kết luận mảng A tồn tại x, *return true*
- Nếu bé hơn, thì x nằm ở nửa bên trái của mảng
  -> ***Tìm kiếm nhị phân nửa bên trái***
- Nếu lớn hơn, thì x nằm ở nửa bên phải của mảng
  -> ***Tìm kiếm nhị phân ở nửa bên phải***

Binary Search

❑ Cài đặt

```
1  bool binary_search(const vector<int> &arr,int lo,int hi,int x){
2      if (lo > hi) {
3          return false;
4      }
5
6      int m = (lo + hi) / 2;
7      if (arr[m] == x) {
8          return true;
9      } else if (x < arr[m]) {
10         return binary_search(arr, lo, m - 1, x);
11     } else if (x > arr[m]) {
12         return binary_search(arr, m + 1, hi, x);
13     }
14 }
15
16 binary_search(arr, 0, arr.size() - 1, x);
```

- $T(n) = T(n/2) + 1$
- $O(\log n)$

Binary Search

❑ Cài đặt – khử đệ quy

```
1  bool binary_search(const vector<int> &arr, int x) {
2      int lo = 0,
3          hi = arr.size() - 1;
4
5      while (lo <= hi) {
6          int m = (lo + hi) / 2;
7          if (arr[m] == x) {
8              return true;
9          } else if (x < arr[m]) {
10             hi = m - 1;
11         } else if (x > arr[m]) {
12             lo = m + 1;
13         }
14     }
15
16     return false;
17 }
```

Nội dung

- Ý tưởng chia để trị
- Lược đồ chung
- Merge sort
- Binary Search
- Quick sort

Quick Sort

❑ Bài toán

- Cho mảng gồm n phần tử A[1..n], sắp xếp mảng A theo thứ tự tăng dần

Quick Sort

❑ Ý tưởng

- Chọn ngẫu nhiên 1 giá trị trong mảng
- So sánh cách phần tử trong mảng với giá trị này.
- Nếu bé hơn thì nằm bên trái
- Nếu lớn hơn thì nằm bên phải
- Lặp lại quá trình cho đến khi nào mảng được sắp xếp

Quick Sort

- Cài đặt

```
int partition(int A[], int lo, int hi) {
   int pivot = A[(lo + hi) / 2]
   int i = lo, j = hi;
   while (true) {
        while(A[i] < pivot)
             i++;
        while(A[j] > pivot)
             j--;
        if (i >= j)
             return j;
        swap(A[i], A[j])
   }
}
```

Quick Sort

- Cài đặt

```
void quick_sort (int A[], int lo, int hi) {
  if (lo >= 0 && hi >= 0 && lo < hi ) {
      int p = partition(A, lo, hi)

      quick_sort(A, lo, p);
      quick_sort(A, p+1, hi);
  }
}
```

https://en.wikipedia.org/wiki/Quicksort#/media/File:Quicksort-example.gif


Q
A


THANK
YOU