

PROGRAMMING

Lecture 17

Dept. of Computer Engineering
Hanbat National University

OUTLINE

Divide and conquer

Binary search

Merge sort

DIVIDE AND CONQUER

“**Divide and conquer**” is a strategy to solve a problem.

Given a (hard) problem, it **divides** the problem into **sub-problems**, and **solves** the **sub-problems independently**, and finally **combines** their **solutions** to obtain a solution of the original problem.

Julius Caesar(BC 100 - 44): “**divide and impera**”
(or divide and rule) to govern the European continent.

The British practiced this idea to control the Indian sub-continent(1858 - 1947).

Benjamin Franklin(1706 - 1790): “We must all **hang together**, or assuredly we shall all **hang separately**.”

Sequential Search

L = [("John", 20), ("Mary", 19), ("David", 22), ("James", 21),
("Young", 19), ("Susan", 22)]

Given the name of a **person** and an **unsorted list**, report his/her age.

```
def seq_search(L, qname):  
    for i in range(len(L)):  
        name, age = L[i]  
        if qname == name:  
            return age  
    return -1  
seq_search(L, "David")
```

How many comparisons?
At most n comparisons,
where $n = \text{len}(L)$!

What if L is sorted?

Try "Jeff." You may stop after comparing it with "John".
Why?

$L = [(David, 22), (James, 21), (John, 20), (Mary, 19), (Susan, 22)], (Young, 19)]$

However, we need n comparisons in the worst case, anyway, where $n = \text{len}(L)$. Why?

BINARY SEARCH

Basic idea

Assumption: the list L is sorted.

$L = [1, 2, 6, 8, 11, 17, 23]$

$q = 19$

low = 0

3

high = 6

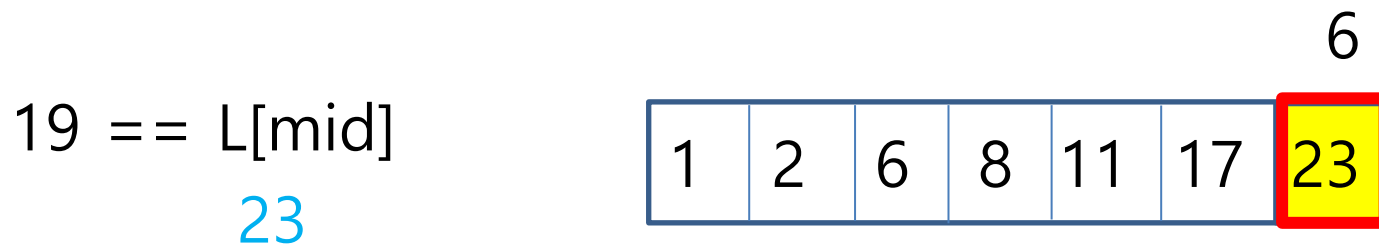
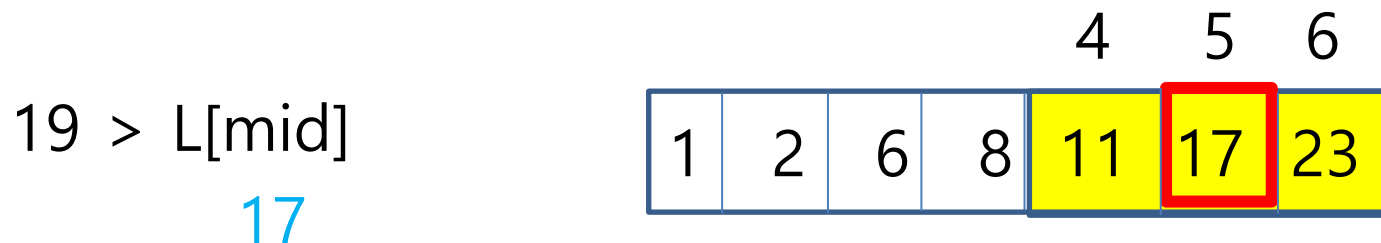
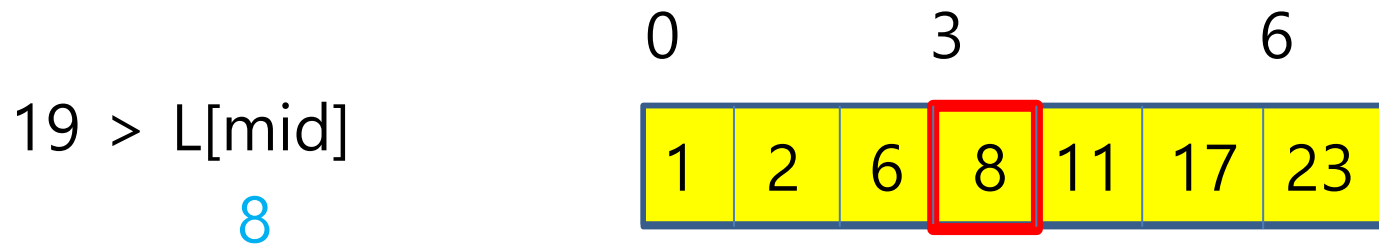
1	2	6	8	11	17	23
---	---	---	---	----	----	----

$mid = (low + high) / 2$

$L[mid] = 8$

With a single comparison, we can reduce the search space by half. **Divide and conquer!**

How many comparisons? $\boxed{2} + \boxed{1} = 3$



(low = high = mid)

Observations

1. When **low == high**, one **additional comparison** is required to make the final decision. At this time, the **length of the sub-list** is reduced to **one**.

base case

2. As long as **high > low**, the **length of the sub-list** reduced **by half** with a single comparison.

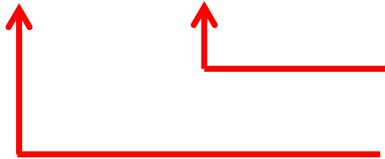
recursive case

With a **single comparison**, the **search space** is **reduced by half**. In order to make the **final decision**, the **search space** should be reduced to **one**.

How many comparisons are required to make the final decision ? At most $\log_2 n$. **Why?**

$$n/2^k = 1 \implies k = \log_2 n, \text{ where } n = \text{len}(L).$$

$$\log_2 n + 1$$

 to make the final decision
to reduce the search space

```
def b_search(L, q, low, high):
```

```
    if low == high:  
        return L[low] == q
```

← Base case

```
    mid = (low + high) / 2  
    if L[mid] == q:  
        return True  
    elif L[mid] < q:  
        return b_search(L, q, mid+1, high)
```

← Recursive case

```
    else:  
        if low != mid:  
            return b_search(L, q, low, mid-1)  
        else:  
            return False
```

← Why this ?
Try L =[5, 7] and q = 3!

MERGE SORT

Basic Idea

1. If the list is of length 0 or 1, it is already sorted.

the base case

2. Otherwise, **divide** the **list** into **two sub-lists of** (almost) **equal size** and **sort each of them** recursively by using **merge sort** .

the recursive case

3. Merge the results.

Divide and conquer

[23, 2, 8, 6, 17, 11, 20, 7]

[23, 2, 8, 6] [17, 11, 20, 7]

[23, 2] [8, 6] [17, 11] [20, 7]

[23] [2] [8] [6] [17] [11] [20] [7]

[2, 23] [6, 8] [11, 17] [7, 20]

[2, 6, 8, 23] [7, 11, 17, 20]

[2, 6, 7, 8, 11, 17, 20, 23]

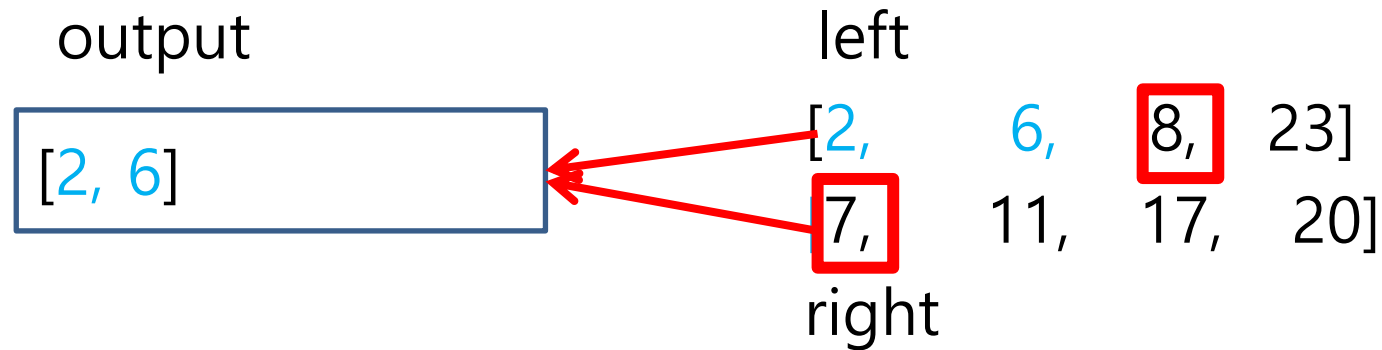
divide

merge

```
def merge_sort(L):  
    if len(L) < 2: } base case  
        return L[:] }  
    mid = len(L) / 2  
    left = merge_sort(L[:mid]) } recursive case  
    right = merge_sort(L[mid:]) }  
    return merge(left, right)
```

```
L =[23, 2, 8, 6, 17, 11, 20, 7]  
L = merge_sort(L)  
print L
```

How to merge to sorted lists



How many comparisons?

$\text{len}(\text{left}) + \text{len}(\text{right}) - 1 = n - 1$, where $n = \text{len}(L)$.

Why?

At most **n comparisons**.

```
def merge(left, right):  
    result = []  
    i, j = 0, 0  
    while i < len(left) and j < len(right):
```

```
        if left[i] < right[j]:  
            result.append(left[i])  
            i += 1  
        else:  
            result.append(right[j])  
            j += 1
```

```
    while i < len(left):  
        result.append(left[i])  
        i += 1
```

No more elements in right

```
    while j < len(right):  
        result.append(right[j])  
        j += 1
```

No more elements in left

```
    return result
```

```
def merge_sort(L):  
    if len(L) < 2:  
        return L[:]  
    mid = len(L) / 2  
    left = merge_sort(L[:mid])  
    right = merge_sort(L[mid:])  
    return merge(left, right)
```

How many comparisons?

```
L = [23, 2, 8, 6, 17, 11, 20, 7]  
L = merge_sort(L)  
print L
```

1 [23, 2, 8, 6, 17, 11, 20, 7]

2 [23, 2, 8, 6] [17, 11, 20, 7]

3 [23, 2] [8, 6] [17, 11] [20, 7]

divide

[23] [2] [8] [6] [17] [11] [20] [7]

1 [2, 23] [6, 8] [11, 17] [7, 20]

2 [2, 6, 8, 23] [7, 11, 17, 20]

3 [2, 6, 7, 8, 11, 17, 20, 23]

merge

How many rounds ?

How many comparisons per round?

How many rounds?

$$n / 2^k = 1$$

$$2^k = n \implies k = \log_2 n$$

How many comparisons at each round?

At most n comparisons

Therefore, at most **$n \log_2 n$** comparisons

$O(n \log_2 n)$

Behavior of the merge sort.

The **dividing** and merging **phases** are **mixed** unlike the figure in the previous slide !

```
def merge_sort(L):
```

```
    global lv
```

```
    lv += 1
```

```
    if len(L) < 2:
```

```
        display(L)
```

```
        lv -= 1
```

```
        return L[:]
```

```
    display(L)
```

```
    mid = len(L) / 2
```

```
    left = merge_sort(L[:mid])
```

```
    right = merge_sort(L[mid:])
```

```
    result = merge(left, right)
```

```
    display(result)
```

```
    lv -= 1
```

```
    return result
```

```
def display(L):
```

```
    if lv == 0:
```

```
        print L
```

```
    else:
```

```
        print " " * 4 * lv, L
```

```
lv = -1
```

```
L = [23,2,6,8,17,11,20,7]
```

```
merge_sort(L)
```

```
[23, 2, 6, 8, 17, 11, 20, 7]
```

```
    [23, 2, 8, 6]
```

```
        [2, 23]
```

```
            [23]
```

```
            [2]
```

```
        [2, 23]
```

```
    [8 6]
```

```
        [8]
```

```
        [6]
```

```
    [6, 8]
```

```
    [2, 6, 8, 23]
```

```
    [17, 11, 20, 7]
```

```
        [17, 11]
```

```
            [17]
```

```
            [11]
```

```
        [11, 17]
```

```
    [20, 7]
```

```
        [20]
```

```
        [7]
```

```
    [7, 20]
```

```
    [7, 11, 17, 20]
```

```
[2, 6, 7, 8, 11, 17, 20, 23]
```