Divide and Conquer
Merge Sort

Dr. 何明昕，He Mingxin, Max

program06 @ yeah.net

Email Subject：(L1-|L2-|L3-) + *last 4 digits of ID* + *Name*: *TOPIC*

Your Lab Class

Sakai: CS203B Fall 2022

数据结构与算法分析B

Data Structures and Algorithm Analysis

Lecture 6

- Divide-and-Conquer (Ch4 of Text B)
- Merge Sort (2.2 of Text A)

To be discussed in Lecture 7:

- Quick Sort (2.3 of Text A)

DIVIDE AND CONQUER

Algorithm Design Technique

Divide and Conquer

- Mathematical Induction Analogy

- Solve the problem by
  - Divide it into smaller parts
  - Solve the smaller parts **recursively**
  - Merge the result of the smaller parts

Induction

- Prove the smallest case **(the basis step)**
- Relate how the result from the smaller case constitutes the proof of the larger case **(the induction step)**
  - What is the relation ?
    - (n-1) → n? (basic induction)
    - (n-m)→n? (complete induction)

Induction Example

- Show that
  - $1+2+3+4+\ldots+n = n(n+1)/2$
- The basis step
  - $1 = 1(1+1)/2$ **true**

The Inductive Step

- The inductive step
  - Assume that it is true for (n-1)
  - Check the case (n)
    - $1 + 2 + 3 + \ldots + n = (1 + 2 + 3 + \ldots + (n-1)) + n$
    - $= (n-1)(n) / 2 + n$
    - $= n((n-1)/2 + 1)$
    - $= n(n/2-1/2 + 1)$
    - $= n(n/2 + 1/2)$
    - $= n(n+1)/2$

The Inductive Step

- The inductive step
  - Assume that it is true for (n-1)
  - Check the case (n)
    - $1 + 2 + 3 + \ldots + n = (1 + 2 + 3 + \ldots + (n-1)) + n$
    - $= \boxed{(n-1)(n)} / 2 + n$
    - $= n((n-1)/2 + 1)$
    - $= n(n/2 - 1/2 + 1)$
    - $= n(n/2 + 1/2)$
    - $= n(n+1)/2$

By using (n-1)

The Induction

- By using the result of the smaller case
- We can proof the larger case

```java
// the d2 day should be equal or after the d1 day
private static int daysPassed (int[] d1, int[] d2) {
    if (d1[Y] == d2[Y] && d1[M] == d2[M])  // same year & month
        return d2[D] - d1[D];

    int days = 0;
    if (d1[Y] == d2[Y]) {  // same year
        days = daysPerMonth( d1[Y], d1[M] ) - d1[D];
        for (int m = d1[M]+1; m < d2[M]; m++)
            days += daysPerMonth( d1[Y], m );
        days += d2[D];
    } else {                // over years
        days = daysPassed( d1, date(d1[Y], DECEMBER, 31) );
        for (int y = d1[Y]+1; y < d2[Y]; y++)
            days += isLeapYear(y) ? 366 : 365;
        days += 1 + daysPassed( date(d2[Y], JANUARY, 1), d2 );
    }
    return days;
}
```

Key of Divide and Conquer

- Divide into smaller parts (subproblems)
- Solve the smaller parts **recursively**
- Merge the result of the smaller parts

Steps in D&C

- Questions
  - **What if we has the solution of the subproblem (smaller problem)?**
    - **What is the subproblem? (how to divide?)**
    - **What can we do with the result? (how to conquer?)**
- If we know the answer, the rest comes automatically from the recursion

Code Example

```
ResultType DandC(Problem p) {
    if (p is trivial) {
       solve p directly
       return the result
    } else {
       divide p into p1,p2,...,pn

       for (i = 1 to n)
         ri = DandC(pi)

       combine r1,r2,...,rn into r
       return r
    }
}
```

Code Example

```
ResultType DandC(Problem p) {
    if (p is trivial) {
        solve p directly                      ts
        return the result
    } else {
        divide p into p1,p2,...,pn            td

        for (i = 1 to n)                      tr
          ri = DandC(pi)

        combine r1,r2,...,rn into r           tc
        return r
    }
}
```

Trivial Case

Divide

Recursive

Combine

Examples

Let’s see two examples

- Merge Sort (details in notes06B)
- Quick Sort (details in Lecture 7)

MERGE SORT

(Brief Introduction, To Be Tuned in Notes06B)

The Sorting Problem

- Given a sequence of numbers
  - $A = [a_1, a_2, a_3, \ldots, a_n]$
- Output
  - The sequence A that is sorted from min to max

The Question

- What if we know the solution of the smaller problem?
  - What is the smaller problem?
    - Try the same problem → sorting

- What if we know the result of the sorting of some elements?

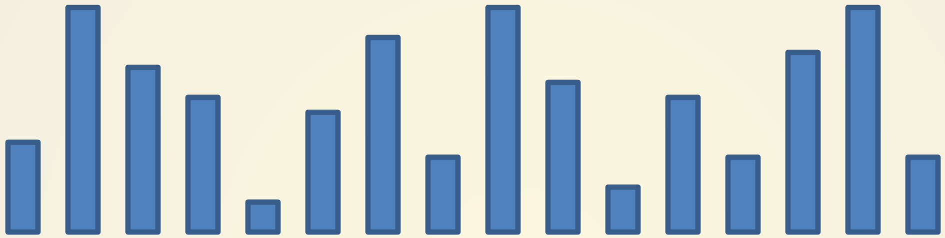
The Question

- How to divide?
  - Let’s try sorting of the smaller array
    - Divide exactly at the half of the array
- How to conquer?
  - Merge the result directly

Idea

- Simplest dividing
  - Divide array into two array of half size
- Laborious conquer
  - Merge the result

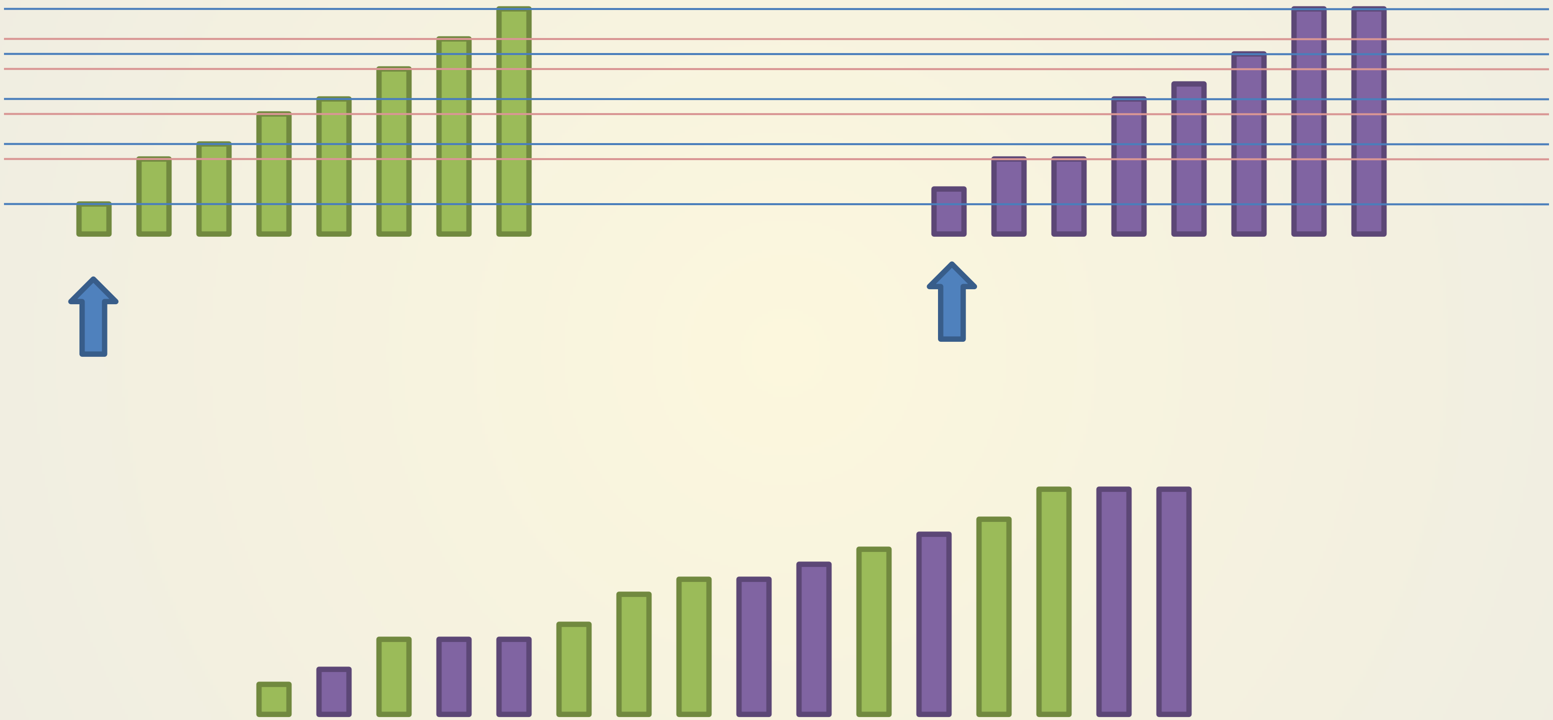
Divide



Divide $\Theta(1)$

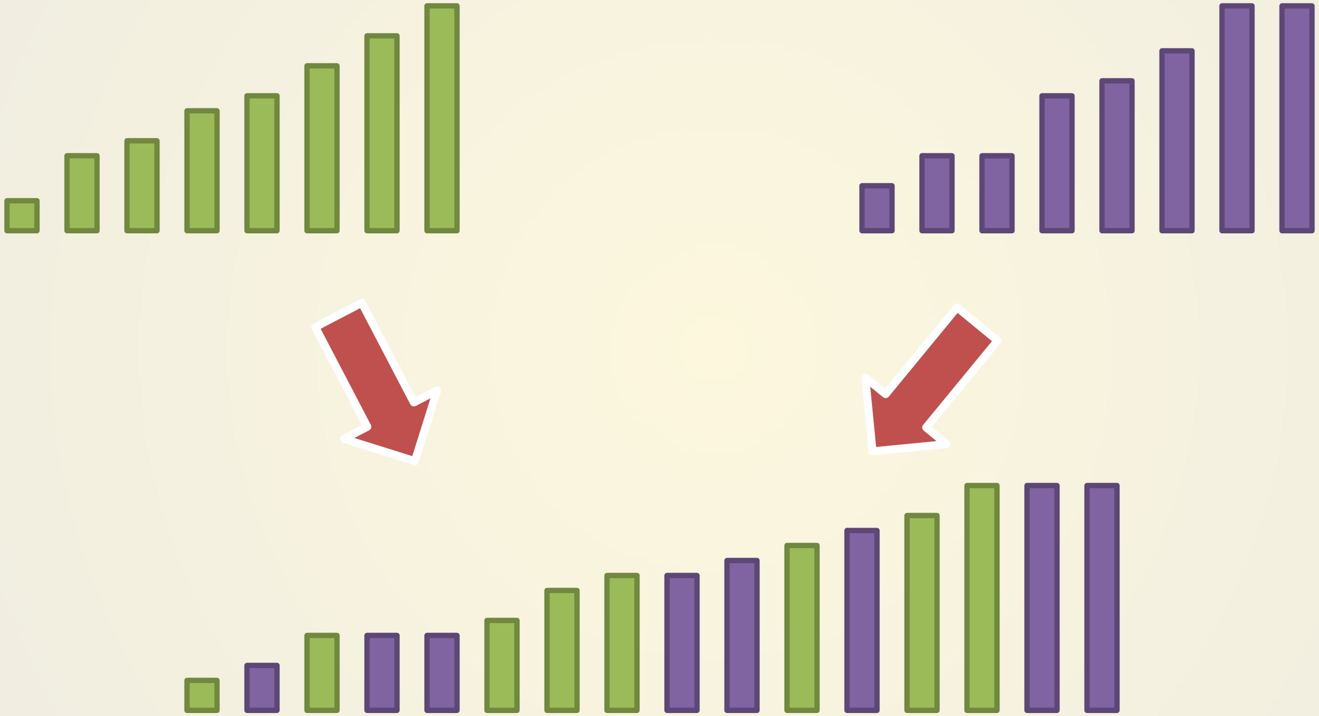
Solve by Recursion T(N/2)

Merge



Merge

$\Theta(N)$



Analysis

- $T(n) = 2T(n/2) + O(n)$
- Master method (Ch4, TextB; to be tuned in notes06B)
  - $T(n) = O(n \lg n)$

QUICK SORT

(Brief Introduction, To Be Tuned in Lecture 7)

The Sorting Problem (again)

- Given a sequence of numbers
  - $A = [a_1, a_2, a_3, \ldots, a_n]$
- Output
  - The sequence A that is sorted from min to max

Problem of Merge Sort

- Need the use of external memory for merging

- Are there any other way such that conquering is not that complex?

The Question

- How to divide?
  - Try doing the same thing as the merge sort
  - Add that every element in the first half is less than the second half
    - Can we do that?
- How to conquer?
  - The sorted result should be easier to merge?

Idea

- Laborious dividing
  - Divide array into two arrays
    - Add the requirement of value of the array
- Simplest conquer
  - Simply connect the result

Is dividing scheme possible?

- Can we manage to have two subproblems of equal size
  - That satisfy our need?

- Any trade off?

The Median

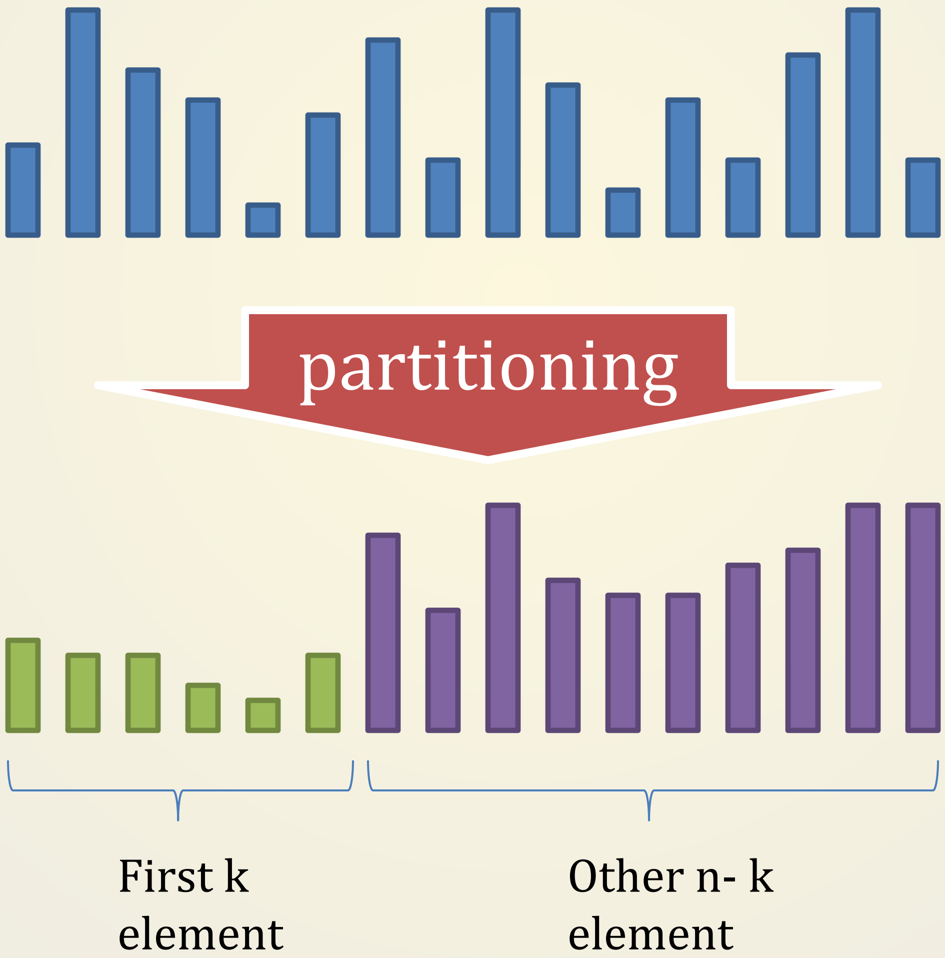
- We need to know the median
  - There are **n/2 elements** which are not more than the median
  - There are another **n/2 elements** which are not less than the median
- Can we have the median?
  - Hardly possible at this step

Simplified Division

- Can we simplify?
  - Not using the median?
- Using k^{th} member
  - There are **k elements** which are not more than the median
  - There are another **(n-k) elements** which are not less than the median
- Simply pick any member and use it as a "pivot"

Divide

Θ(N)

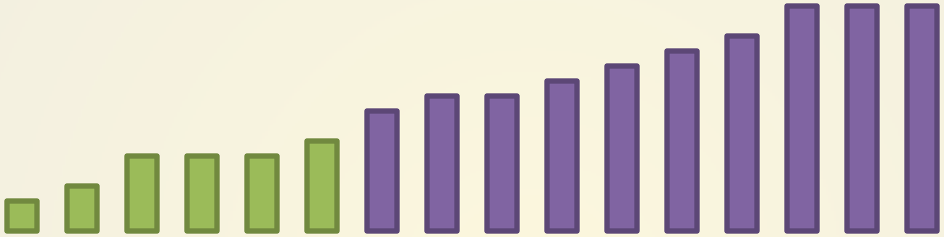


Solve by Recursion

T(K) + T(N – K)

Conquer

O(1)



Do nothing!

Analysis

- $T(n) = T(k) + T(n - k) + \Theta(n)$

- There can be several cases
  - Up to which K that we chosen

Analysis : Worst Case

- K always is 1^{st}

What should be our T(N) ?

What is the time complexity

Analysis : Worst Case

- K always is 1^{st}

$$
\begin{aligned}
T(n) &= T(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Sigma\, \Theta(i) \\
&= \Theta(n^2)
\end{aligned}
$$

$\boxed{\textbf{Not good}}$

Analysis : Best Case

- K always is the median

What should be our T(N) ?

What is the time complexity

Analysis : Best Case

- K always is the median

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&= \Theta(n \log n)
\end{aligned}
$$

The same as the merge sort (without the need of extra / external memory)

Fixing the worst case

- When will the worst case happen?
  - When we always selects the smallest element as a pivot
  - Depends on pivot selection strategy

- If we select the first element as a pivot
  - What if the data is sorted?

Fixing the worst case

- Select wrong pivot leads to worst case.
- There will exist some input such that for any strategy of “deterministic” pivot selection leads to worst case.

Fixing the worst case

- Use “non-deterministic” pivot selection
  - i.e., randomized selection
- Pick a random element as a pivot
- It is unlikely that every selection leads to worst case
- We can hope that, on average,
  - it is O(n log n)

Summary

- Divide-and-Conquer (Ch4 of Text B)
- Merge Sort (2.2 of Text A, to be continued in notes06B)

To be discussed in Lecture 7:

- Quick Sort (2.3 of Text A)