# Course organization

- **Introduction ( Week 1-2)**
  – Course introduction
  – A brief introduction to molecular biology
  – A brief introduction to sequence comparison
- **Part I: Algorithms for Sequence Analysis (Week 3 - 8)**
  – Chapter 1-3, Models and theories
    » Probability theory and Statistics (Week 3)
    » Algorithm complexity analysis (Week 4)
    » Classic algorithms (Week 5)
  – Chapter 4. Sequence alignment (week 6)
  – Chapter 5. Hidden Markov Models ( week 7）
  – Chapter 6. Multiple sequence alignment (week 8)
- **Part II: Algorithms for Network Biology (Week 9 - 16)**
  – Chapter 7. Omics landscape (week 9)
  – Chapter 8. Microarrays, Clustering and Classification (week 10)
  – Chapter 9. Computational Interpretation of Proteomics (week 11)
  – Chapter 10. Network and Pathways (week 12,13)
  – Chapter 11. Introduction to Bayesian Analysis (week 14,15)
  – Chapter 12. Bayesian networks (week 16)

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY


# Chapter 2: Algorithm Complexity Analysis

Chaochun Wei

Spring 2018

# Contents

- Reading materials
- Why do we need to analyze the complexity of an algorithm?
  - Examples
- Introduction
  – Algorithm complexity
  – "Big O" notation: O()

# Reading

Cormen book:

Thomas, H. ,Cormen, Charles, E., Leiserson, and Ronald, L., Rivest . Introduction to Algorithms, The MIT Press.

(read Chapter 1 and 2, page 1-44).

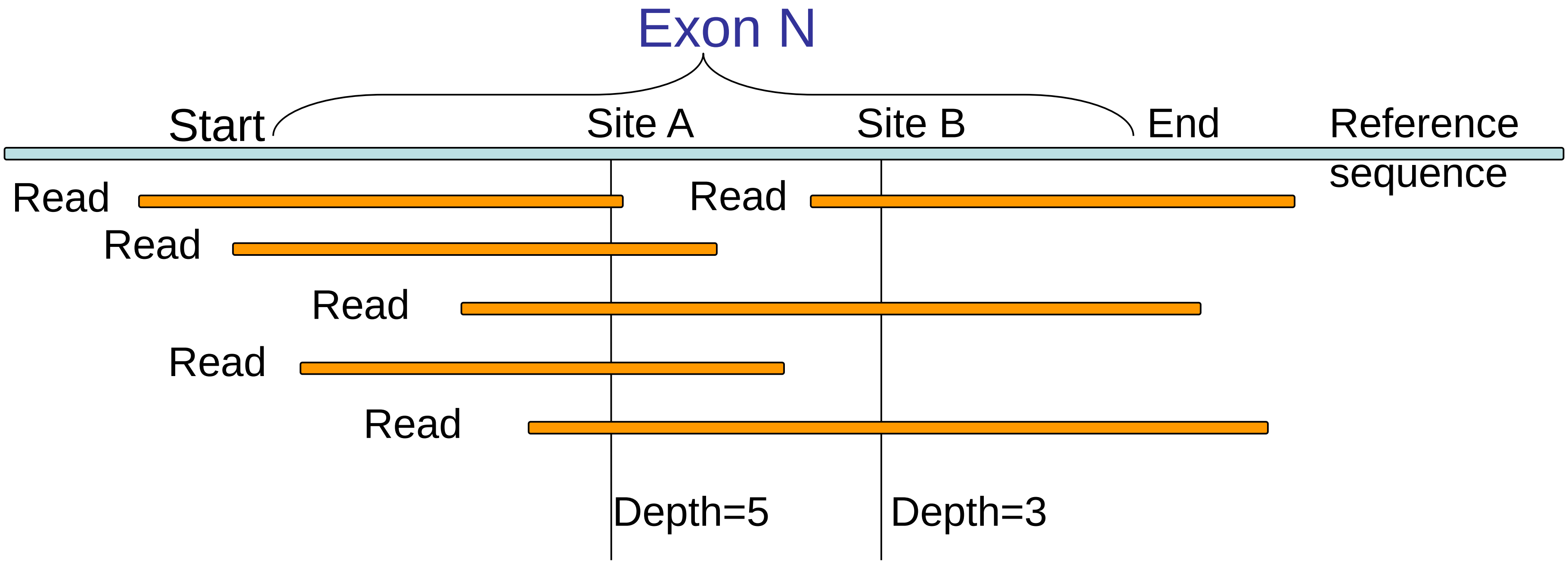
# A real example: Exon-capture data analysis

There are ~60 millions of short reads sequenced from exon regions of a human genome. We need to figure out the how many exons were covered with at least 10 reads.

Steps:

1. Reads are aligned to the genome;
2. Each alignment is checked to see the exon it covers;
3. For each exon, check the number of reads cover the exon;
4. For all exons, filter out those with read number < 10.

# A real example: Exon-capture data analysis



# A real example: Exon-capture data analysis

1 days later

**Student:** I have created a program to do the analysis. It's running.

**Teacher:** Cool. Let me know when your analysis finishes.

# A real example: Exon-capture data analysis

6 days later…

**Student:** My program has been running for 5 days, and it keeps on running. I have no idea about what is happening and what to do with it.

**Teacher:** Its core is a sorting algorithm with a complexity of at most O(N*lgN). It should be done within a few minutes!

**Student:** What?.....

# Algorithm and its complexity

An **algorithm** is any well-defined computational procedure that takes in some **inputs** and produces some **outputs**.

Example: Sort an array of numbers

3, 2, 4, 5, 7, 1, 6 → 1, 2, 3,4, 5,6,7

# Algorithm and its complexity

An algorithm is any well-defined computational procedure that takes in some inputs and produces some outputs.

**Complexity**: a function of **input size**

- Time complexity: the running time
- Space complexity: the memory size required

# Algorithm and its complexity

## Input size

- Number of items in the input
  - Sorting problem
  - FFT
- Total number of bits needed to represent the input
  - Arithmetic operation (+,-,x,/)
- The value of input
  - Factorial (N!)

## Multiple input sizes

- Need to specify which input size is used
  - Graph operation (number of Vertices, and edges)

# Algorithm and its complexity

## Before we start

- we use a generic one-processor, random-access machine.
  - No parallel

# Algorithm and its complexity

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

# Algorithm and its complexity

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

```
Insertion sort (A)
 for j = 2 to length(A)
       do key = A[j]
              /*insert A[j] into the sorted sequence A[1…j-1]
              i=j-1
               while i>0 and A[i]>key
                     do A[i+1]=A[i];
                           i=i-1;
                      A[i+1]=key;
```

# Algorithm and its complexity

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

```
Insertion sort (A)
for j = 2 to length(A)
      do key = A[j]
            /*insert A[j] into the sorted sequence A[1…j-1]
            i=j-1
             while i>0 and A[i]>key
                  do A[i+1]=A[i];
                       i=i-1;
                  A[i+1]=key;
```



Algorithm time complexity: $O(N^2)$

# Worst-case and average-case analysis

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

```
Insertion sort (A)
for j = 2 to length(A)
      do key = A[j]
            /*insert A[j] into the sorted sequence A[1…j-1]
            i=j-1
             while i>0 and A[i]>key
                  do A[i+1]=A[i];
                        i=i-1;
                  A[i+1]=key;
```

**Can repeat from 0 to j times**



Algorithm time complexity: $O(N^2)$

# Order of growth （增长的阶数）

Example: Sort an array of numbers
5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

Insertion sort:

Algorithm run time complexity: $O(N^2)$

Order of growth: 2

# O-notation (big-O notation): Asymptotic upper bound

O(g(n)) = {f(n): there exist positive constants c and $n_0$ such that $0 \le f(n) \le c\, g(n)$ for all $n \ge n_0$}

Note about O-notation operations:
$O(k_1 * N^2 + k_2 * N^3) = O(N^3)$ for constants $k_1$, $k_2$

# O-notation (big-O notation):

## Asymptotic upper bound

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

## Insertion sort:

algorithm time complexity: $O(N^2)$

# Sorting with time complexity of O(N*logN)

Example: Sort an array of numbers
5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

```
Sort (A)
for j = 2 to length(A)
      do key = A[j]
            /*Use binary search to insert A[j]
            /*into the sorted sequence A[1...j-1]
            i=j-1
            Binary_search(A[j], A[1…j-1],)
```

# Sorting

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

There are a lot of sorting algorithms:

- Heap sort (O(N*logN))
- Merge sort (O(N*logN))
- *Quick sort (worst-case $O(N^2)$, average O(N*logN))

# Merge sort

```
Merge-Sort (A, p, r)
        if p<r
                then q=[(p+r)/2]
                        Merge-Sort(A, p, q)
                        Merge-Sort(A,q+1,r)
                        Merge(A, p, q, r)
```

Time Complexity: $$T(N) = \begin{cases} O(1); if\ N = 1 \\ 2T(N/2) + O(N); if\ N > 1 \end{cases}$$

Solve it: T(N) = O(N*logN)

# Space complexity

Example: Sort an array of numbers

5, 2, 4, 6, 1, 3 → 1, 2, 3,4, 5,6

Need an array of size N: A[1…N], and 3 temporary variables O(N)

Example: Sequence alignment

# Algorithm complexity

## Examples: Sequence alignment

- **Needleman/Wunsch global alignment**
- **Smith/Waterman local alignment**

# Smith/Waterman local alignment (1981)

- Two sequences $X = x_1 \ldots x_n$ and $Y = y_1 \ldots y_m$
- Let $F(i, j)$ be the optimal alignment score *of* $X_{1\ldots i}$ of X up to $x_i$ and $Y_{1\ldots j}$ of Y up to $Y_j$ $(0 \le i \le n, 0 \le j \le m)$, *then we have*

$$F(0,0) = 0$$

$$F(i, j) = \max \begin{cases} 0 \\ F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

# Sequence alignment

| | Δ | C | A | G | C | C | U | C | G | C | U | U | A | G |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Δ | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 |
| A | 0·0 | 0·0 | 1·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 1·0 | 0·0 |
| A | 0·0 | 0·0 | 1·0 | 0·7 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 0·0 | 1·0 | 0·7 |
| U | 0·0 | 0·0 | 0·0 | 0·7 | 0·3 | 0·0 | 1·0 | 0·0 | 0·0 | 0·0 | 1·0 | 1·0 | 0·0 | 0·7 |
| G | 0·0 | 0·0 | 0·0 | 1·0 | 0·3 | 0·0 | 0·0 | 0·7 | 1·0 | 0·0 | 0·0 | 0·7 | 0·7 | 1·0 |
| C | 0·0 | 1·0 | 0·0 | 0·0 | 2·0 | 1·3 | 0·3 | 1·0 | 0·3 | 2·0 | 0·7 | 0·3 | 0·3 | 0·3 |
| C | 0·0 | 1·0 | 0·7 | 0·0 | 1·0 | 3·0 | 1·7 | 1·3 | 1·0 | 1·3 | 1·7 | 0·3 | 0·0 | 0·0 |
| A | 0·0 | 0·0 | 2·0 | 0·7 | 0·3 | 1·7 | 2·7 | 1·3 | 1·0 | 0·7 | 1·0 | 1·3 | 1·3 | 0·0 |
| U | 0·0 | 0·0 | 0·7 | 1·7 | 0·3 | 1·3 | 2·7 | 2·3 | 1·0 | 0·7 | 1·7 | 2·0 | 1·0 | 1·0 |
| U | 0·0 | 0·0 | 0·3 | 0·3 | 1·3 | 1·0 | 2·3 | 2·3 | 2·0 | 0·7 | 1·7 | 2·7 | 1·7 | 1·0 |
| G | 0·0 | 0·0 | 0·0 | 1·3 | 0·0 | 1·0 | 1·0 | 2·0 | 3·3 | 2·0 | 1·7 | 1·3 | 2·3 | 2·7 |
| A | 0·0 | 0·0 | 1·0 | 0·0 | 1·0 | 0·3 | 0·7 | 0·7 | 2·0 | 3·0 | 1·7 | 1·3 | 2·3 | 2·0 |
| C | 0·0 | 1·0 | 0·0 | 0·7 | 1·0 | 2·0 | 0·7 | 1·7 | 1·7 | 3·0 | 2·7 | 1·3 | 1·0 | 2·0 |
| G | 0·0 | 0·0 | 0·7 | 1·0 | 0·3 | 0·7 | 1·7 | 0·3 | 2·7 | 1·7 | 2·7 | 2·3 | 1·0 | 2·0 |
| G | 0·0 | 0·0 | 0·0 | 1·7 | 0·7 | 0·3 | 0·3 | 1·3 | 1·3 | 2·3 | 1·3 | 2·3 | 2·0 | 2·0 |

FIG. 1. $H_{ij}$ matrix generated from the application of eqn (1) to the sequences A-A-U-G-C-C-A-U-U-G-A-C-G-G and C-A-G-C-C-U-C-G-C-U-U-A-G. The underlined elements indicate the trackback path from the maximal element 3·30.

Smith and Waterman, JMB, 1981, 147, 195-197

Time complexity: O(N*M)
Space complexity: O(N*M) or O(max(N, M))
Need a two-dimension array of size N*M, and a constant number of temporary variables

# Other issues impact the speed of a program

- Output size
  - Blast: output can be a problem
- Input/Output method/place/mode
  - Speed
    - screen << hard disk << memory
- Programming language
  - Speed
    - Perl < java < C++ <C