
# Lecture 3: Algorithm analysis

## Algorithm

Jeong-Hun Kim

# Remind

- What is an algorithm?
  - Procedure for finding the optimal solution to a finite problem
- Algorithm analysis
  - Predicting the required resources
  - Experimental vs. theoretical analysis
- Computational complexity
  - Time and space complexities
- Asymptotic notations
  - Big O, Omega, Theta

# Table of Contents

Part 1

# PERFORMANCE EVALUATION

# Performance Evaluation

- What is the performance of an algorithm?
  - Quantifying how well it aligns with the goal of the task
  - Major performances:
    - Accuracy
    - Running time
  - Each performance has different importance depending on the purpose of the task
    - Big data system: data processing time
    - Real-time streaming service: querying time, data throughput
    - Recommendation system: mean average precision (MAP)
    - Forecasting system: precision, recall, etc.

# Performance Evaluation

- Running time
  - Evaluate the elapsed time for each unit task
  - Using a function to get the current time
    - e.g., 'clock()' function in the C language

```
void main(){
        int n = 250000;
        float list[] = {0.000001, ..., 0.25};
        clock_t start, end;
        start = clock();
        float res = sum(list, n);
        end = clock();
        elapsed = (double)(end-start);
```

```
float sum(float list[], int n){
        float tempsum = 0;
        int i;
        for(i=0; i<n; i++)
                tempsum += list[i];
        return tempsum
}
```

# Performance Evaluation

- Number of operations
  - Evaluating the number of unit task executions
    - e.g., number of [comparisons, queries, computations, etc.]
  - Running time may be changed slightly with each run
  - It should not be used simultaneously with the running time
    - Why? counting operation is included in the running time

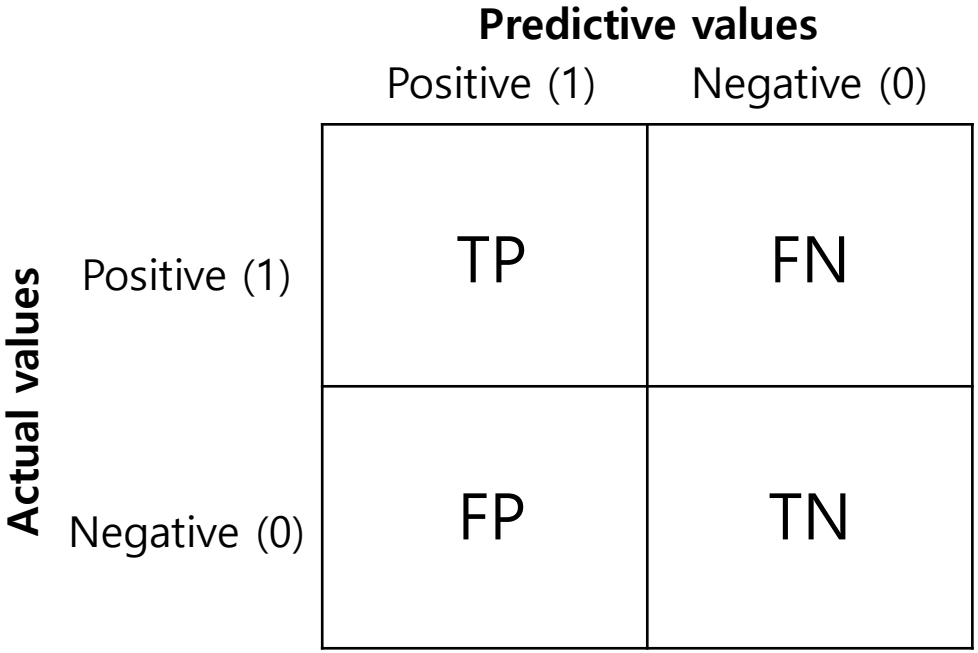
```
float sum(float list[], int n){
        float tempsum = 0;
        int i, nsum = 0;
        for(i=0; i<n; i++)
                tempsum += list[i];
                nsum++;
        return tempsum
}
```

# Performance Evaluation

- Data throughput
  - It corresponds to the data (or operation) processing rate per second
    - e.g., transaction per second (TPS), frame per second (FPS), Floating point operations per second (FLOPS), etc.
  - It is mainly utilized in real-time systems
    - Video streaming, bank transactions, stock market, etc.

# Performance Evaluation

- Confusion matrix
  - Primary metrics for evaluating the accuracy of an algorithm regarding the task's purposes
  - Matrix for comparing ground truth (GT) and predicted results
    - GT: Information known as facts



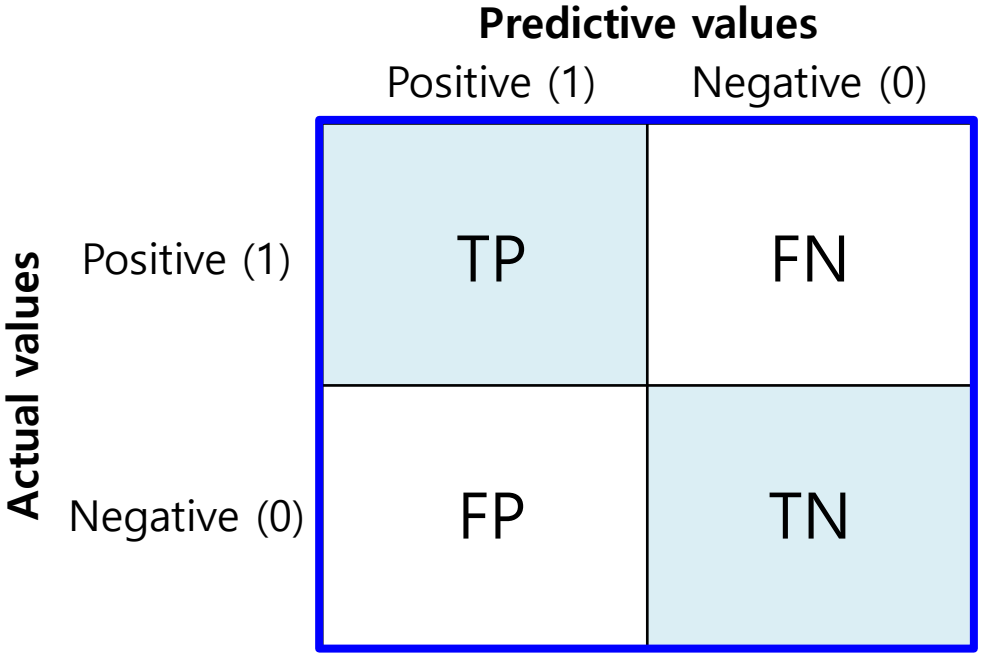

**TP, TN:** correctly predicting the actual values

**FP, FN:** differently predicting from the actual values

# Performance Evaluation

- Confusion matrix (cont'd)
  - Accuracy
    - Proportion of correct predictions out of the total tasks

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$



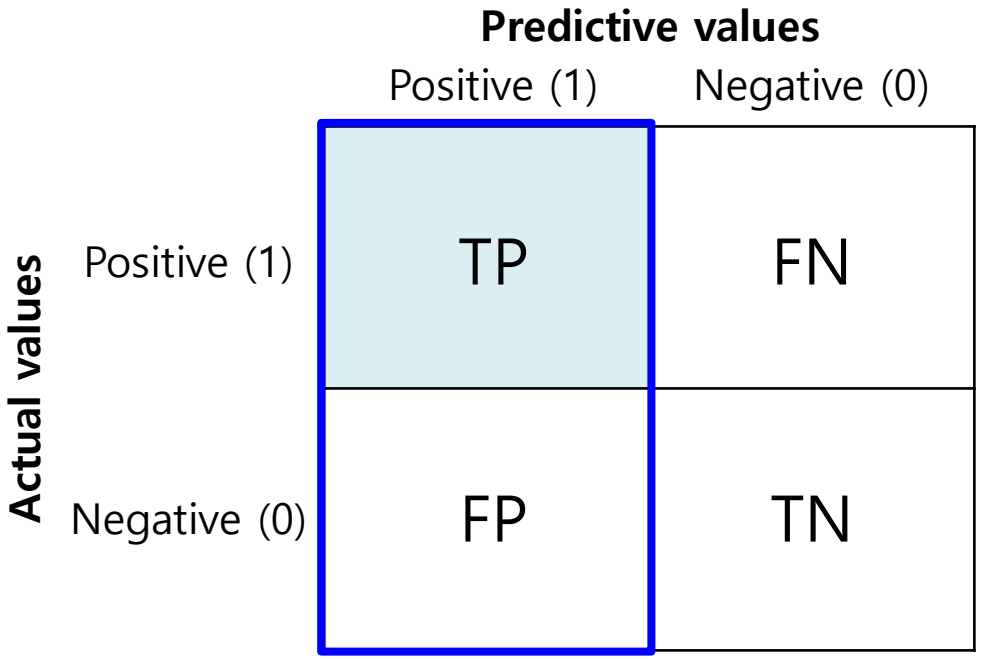

**TP, TN:** correctly predicting the actual values

**FP, FN:** differently predicting from the actual values

# Performance Evaluation

- Confusion matrix (cont'd)
  - Precision
    - Ratio of actual positives out of the predicted positives

$$Precision = \frac{TP}{TP + FP}$$



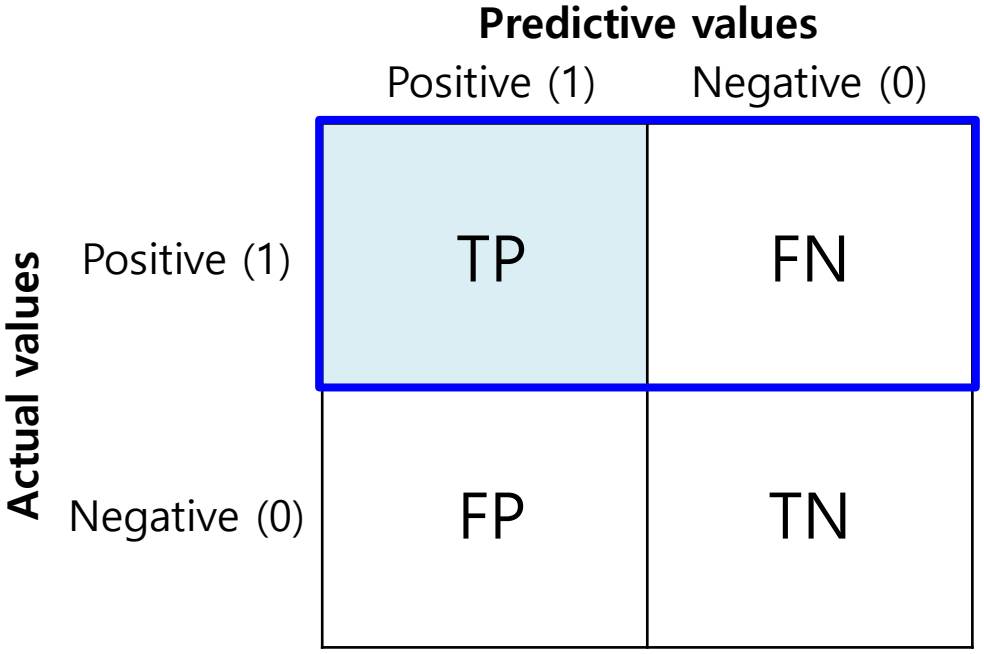

**TP, TN:** correctly predicting the actual values

**FP, FN:** differently predicting from the actual values

# Performance Evaluation

- Confusion matrix (cont'd)
  - Recall
    - Ratio of predicted positives out of the actual positives

$$Recall = \frac{TP}{TP + FN}$$



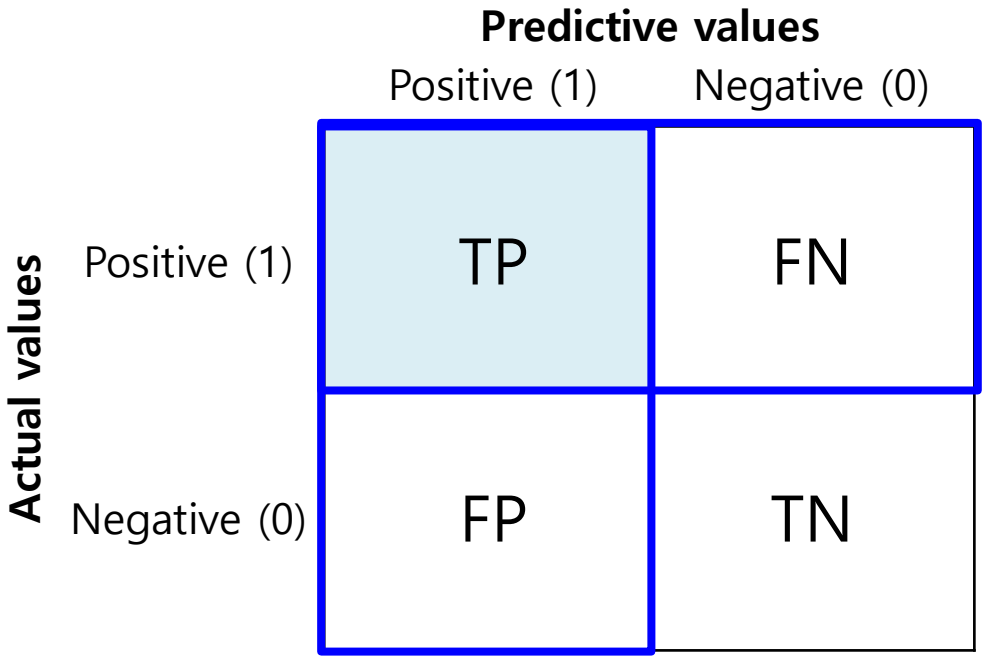

**TP, TN:** correctly predicting the actual values

**FP, FN:** differently predicting from the actual values

# Performance Evaluation

- Confusion matrix (cont'd)
  - F1-score
    - Harmonic mean of precision and recall

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$




**TP, TN:** correctly predicting the actual values

**FP, FN:** differently predicting from the actual values

# Performance Evaluation

- Confusion matrix (cont'd)
  - Comparison between precision and recall
    - Precision: how many of the selected objects were correct
      - Algorithm mistakenly believes it is correct
    - Recall: how many of the objects that should have been selected were actually selected
      - Algorithm mistakenly believes it is incorrect

**Example.** Forgotten girlfriend's birthday

Gf: Baby, When is my birthday?

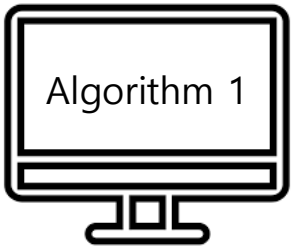
Bf: Ooops!! 

Bf: (Make 5 predictions to save the relationship)

Out of which 1 date is correct. Thank god !!
- Recall is 100%
- Precision is 20%

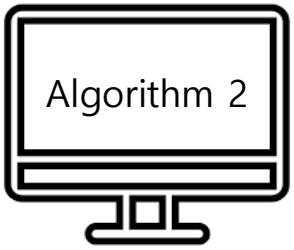
# Performance Evaluation

- Confusion matrix (cont'd)
  - Why is the F1-score used?
    - F1-score is used when data is imbalanced




Predicted

| Actual | A | B | C | D |
|---|---|---|---|---|
| A | 100 | 80 | 10 | 10 |
| B | 0 | 9 | 0 | 1 |
| C | 0 | 1 | 8 | 1 |
| D | 0 | 1 | 0 | 9 |

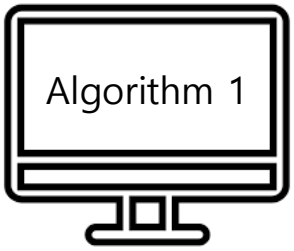



Predicted

| Actual | A | B | C | D |
|---|---|---|---|---|
| A | 198 | 2 | 0 | 0 |
| B | 7 | 1 | 0 | 2 |
| C | 0 | 8 | 1 | 1 |
| D | 2 | 3 | 4 | 1 |

# Performance Evaluation

- Confusion matrix (cont'd)
  - Accuracy of algorithm 1:
    - $(100 + 9 + 8 + 9)/230 = 0.547$
  - F1-score of algorithm 1:
    - $(2 * 0.492 * 0.775)/(0.492 + 0.775) = 0.601$



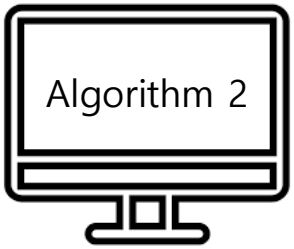

Predicted

Actual

| | A | B | C | D |
|---|---|---|---|---|
| A | 100 | 80 | 10 | 10 |
| B | 0 | 9 | 0 | 1 |
| C | 0 | 1 | 8 | 1 |
| D | 0 | 1 | 0 | 9 |

# Performance Evaluation

- Confusion matrix (cont'd)
  - Accuracy of algorithm 2:
    - $(198 + 1 + 1 + 1)/230 = 0.87$
  - F1-score of algorithm 2:
    - $(2 * 0.369 * 0.323)/(0.369 + 0.323) = 0.344$



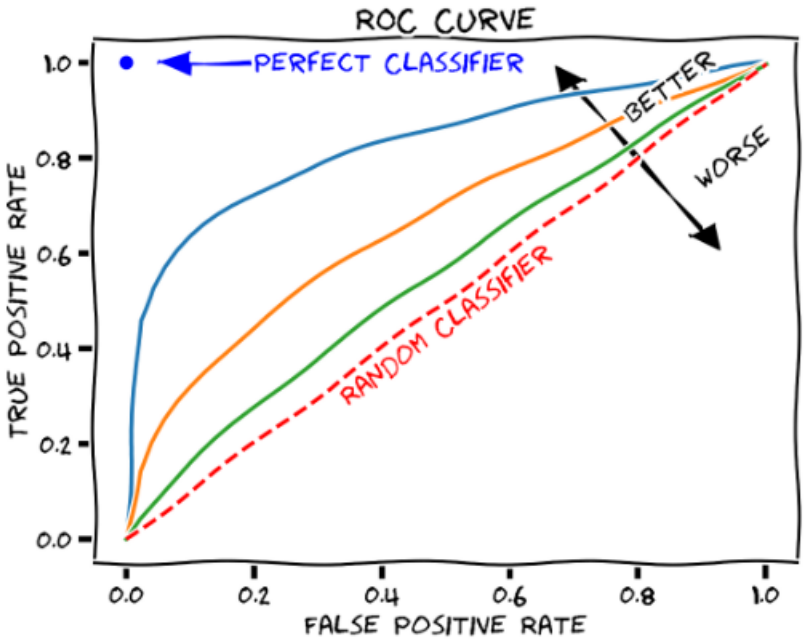

Predicted

Actual

| | A | B | C | D |
|---|---|---|---|---|
| A | 198 | 2 | 0 | 0 |
| B | 7 | 1 | 0 | 2 |
| C | 0 | 8 | 1 | 1 |
| D | 2 | 3 | 4 | 1 |

# Performance Evaluation

- ROC-AUC
  - Receiver operation characteristic curve (ROC)
    - It represents the changes in TPR based on FPR
      - TPR(sensitivity)=$TP/(TP+FN)$ ← the higher, the better
      - FPR(1-specificity)=$FP/(TN+FP)$ ← the lower, the better



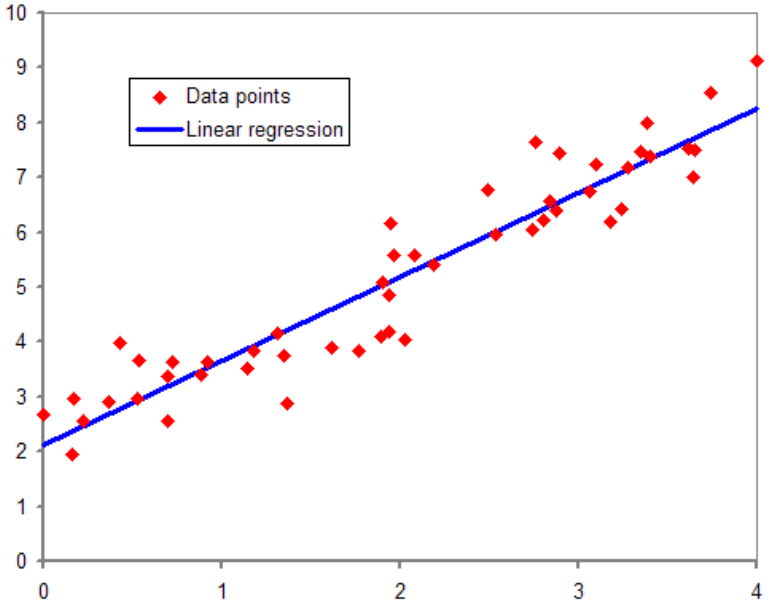

Reference: https://en.Wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roc-draft-xkcd-style.svg

# Performance Evaluation

- Mean absolute error (MAE)
  - Average difference between the actual value and the predicted value

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i|$$

  - Mainly used in regression tasks




Reference: https://commons.wikimedia.org/wiki/File:Normdist_regression.png

# Performance Evaluation

- Mean squared error (MSE)
  - Mean squared difference between the actual and predicted values

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

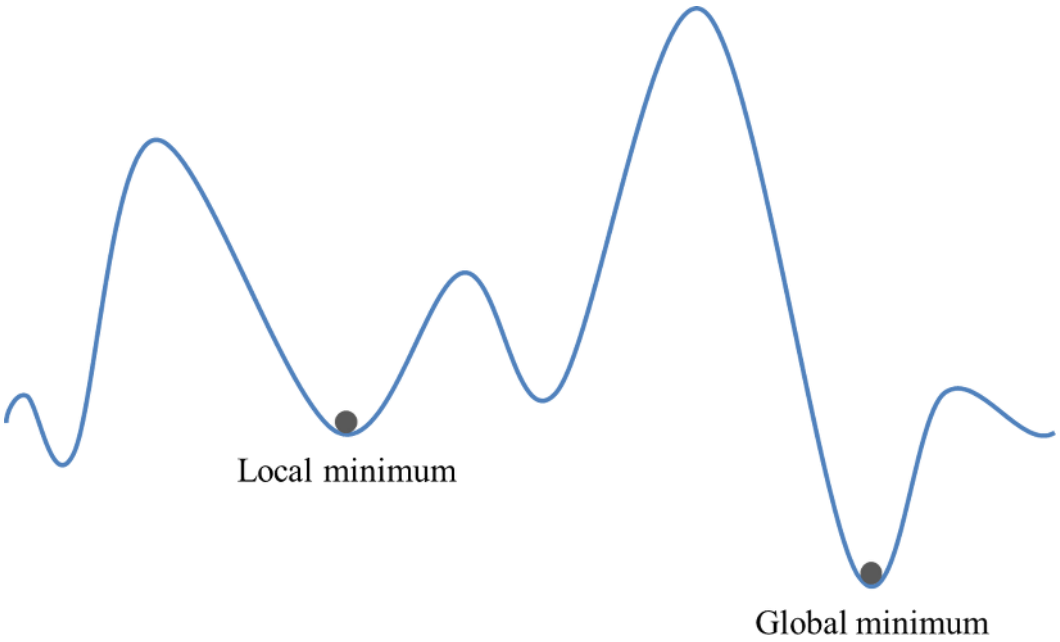
  - It is also used in regression tasks

- MAE vs. MSE
  - MAE is relatively less sensitive to outliers
  - MSE is more likely to converge to the global optimum compared to MAE

# Performance Evaluation

- Global optimal solution
  - Best solution within the entire solution set

- Local optimal solution
  - Best solution within a specific range or region



Part 2

# COMPARISON OF ALGORITHMS

# Comparison of Algorithms

❖ Theoretical comparison

1. Expressing the number of basic operations for each algorithm as a function of the input size $n$
2. Representing the estimated functions in Big-Oh notation
3. Comparing the two functions expressed in Big-Oh notation

Algorithm that can be expressed as a function that grows more slowly is theoretically a better algorithm

# Comparison of Algorithms

❖ Example

Algorithm 1: $f_1(n) = 2n + 20$

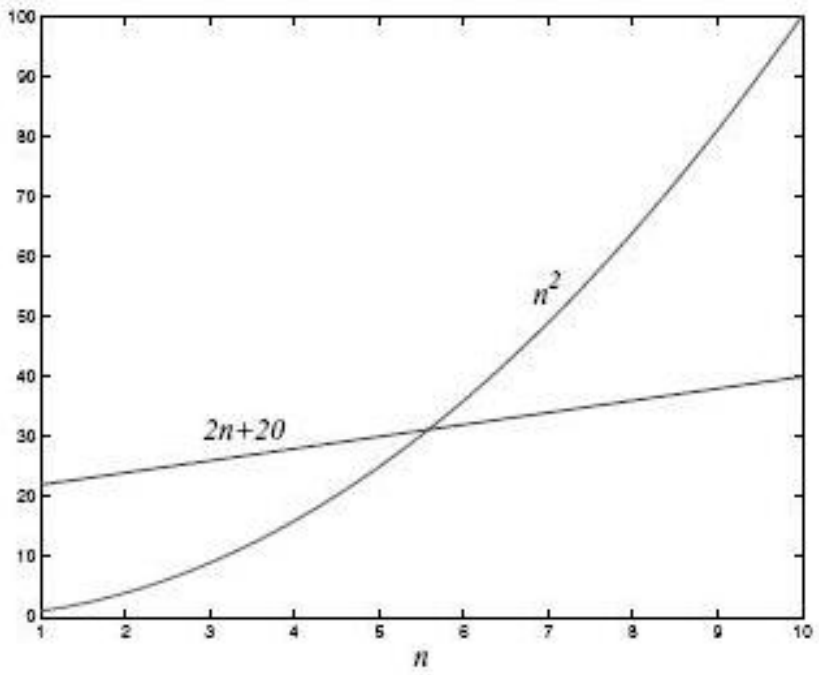
Algorithm 2: $f_2(n) = n^2$

$f_1(n) = 2n + 20 \rightarrow O(n)$

$f_2(n) = n^2 \rightarrow O(n^2)$

If $n < 6$, $f_2$ is a better algorithm than $f_1$

If $n \geq 6$, $f_1$ is a better algorithm than $f_2$



# Comparison of Algorithms

- Experimental comparison
  - Fair comparison
    - It measures and compares the algorithm performance
    - It ensures experimental reproducibility and replicability
    - Results are often changed depending on the experimental settings
  - Experimental reproducibility and replicability
    - Ability of an experiment to produce the consistent result when repeated multiple times
    - Issues
      - Data preprocessing
      - Hyperparameter selection
      - Experimental environment
      - Etc.

Part 3

# ALGORITHM ANALYSIS USING BIG-OH NOTATION

# Algorithm Analysis using Big-Oh Notation

- Defining a function with respect to input size $n$
  - Writing pseudocode for the algorithm
    - Representing the total amount of the following operations as a function of the input size $n$
      - Assigning a value to a variable
      - Number of iterations
      - Number of function calls
      - Number of returns

# Algorithm Analysis using Big-Oh Notation

- $O$(Big-Oh) vs. $\Omega$ (Omega) vs. $\Theta$ (Theta)
  - Big-Oh notation is the most commonly used in algorithm analysis
    - Why?
      - Most concerns focus on the worst-case rather than the best-case

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Add two n-bit binary numbers
    - Assume that basic bit operations on a single bit take $O(1)$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Carry | 1 | | | 1 | 1 | 1 | |
| | | 1 | 1 | 0 | 1 | 0 | 1 |
| | | 1 | 0 | 0 | 0 | 1 | 1 |
| | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

The number of operations for $n$ bits: $n + 1$

$f(n) = n + 1 \rightarrow f(n) = O(n)$

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Multiply two n-bit binary numbers

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | 1 | 1 | 0 | 1 | $x$ |
| $\times$ | | | | | 1 | 0 | 1 | 1 | $y$ |
| | | | | | 1 | 1 | 0 | 1 | |
| | | | | 1 | 1 | 0 | 1 | | One left-shift |
| | | | 0 | 0 | 0 | 0 | | | Two left-shift |
| | | 1 | 1 | 0 | 1 | | | | Three left-shift |
| | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |

Number of bit shift operations: $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

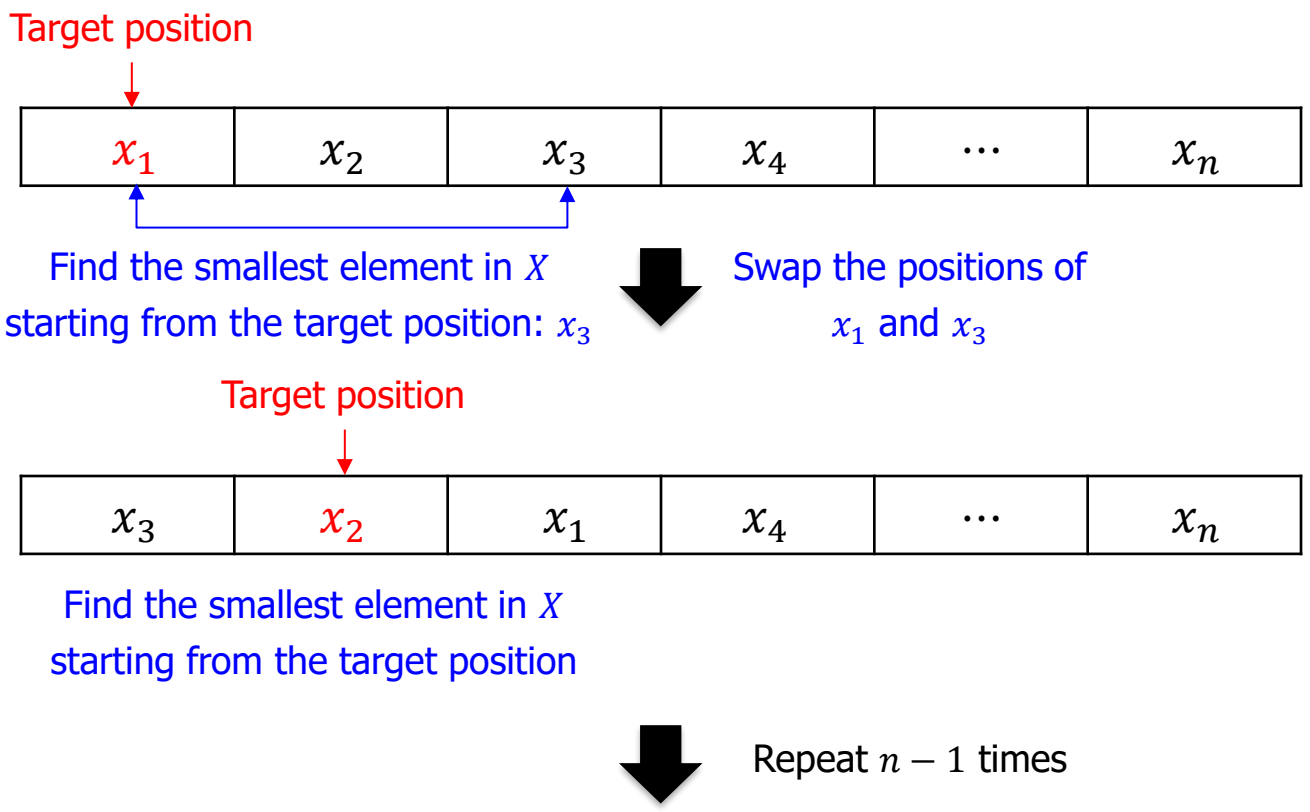
Number of bit addition operations: $\underbrace{n + n + \cdots + n}_{n} = n^2$

$f(n) = \frac{1}{2}(n^2 - n) + n^2 \rightarrow f(n) = O(n^2)$

# Algorithm Analysis using Big-Oh Notation

❖ Example

- Problem. Selection sort (ascending order)
  - Let $X$ be a $n$ integer list $\{x_1, x_2, \ldots, x_n\}$



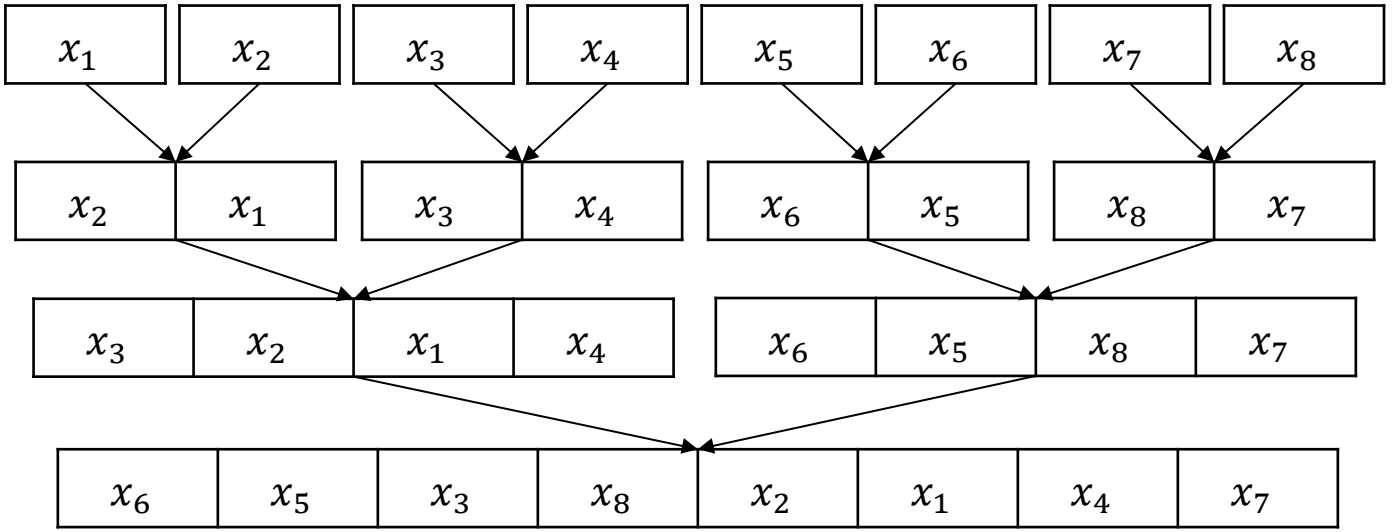
# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Selection sort (ascending order) (cont'd)
    - Total number of comparisons:
      - $n + (n-1) + (n-2) + \cdots + 1 = \frac{1}{2}n(n+1)$
    - Total number of swaps:
      - $n - 1$
    - Time complexity analysis
      - $f(n) = \frac{1}{2}n^2 + \frac{3}{2}n - 1$
      - $f(n) = O(n^2)$

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Merge sort (ascending order)
    - Let $X$ be a $n$ integer list $\{x_1, x_2, \ldots, x_n\}$

If $n = 8$

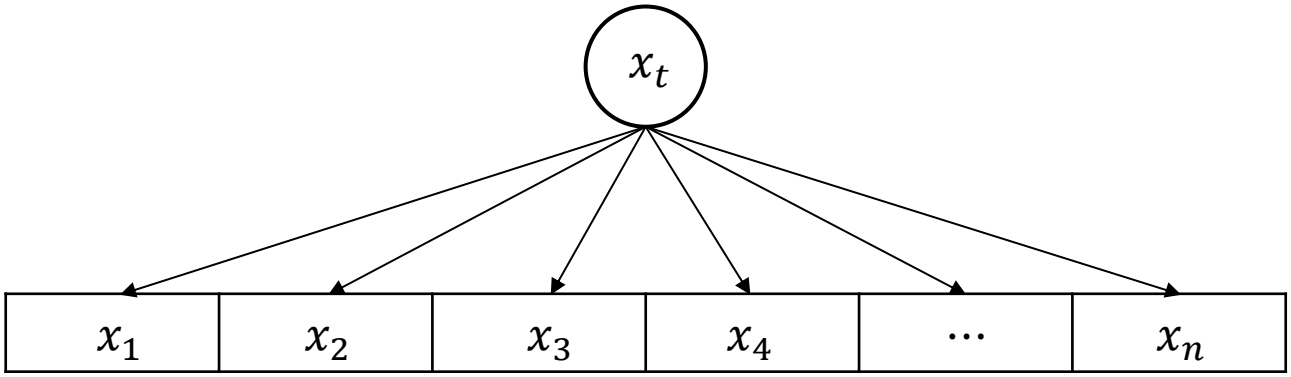


# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Merge sort (ascending order) (cont'd)
    - The number of merges:
      - $\log_2 n$
    - The number of comparisons for each merge:
      - Let $s_i$ be a segment size in $i$-th iteration
      - Let $a_i$ be the number of segments in $i$-th iteration
      - Then, $a_i(s_i - 1)$ comparisons are the worst-case in $i$-th iteration
        - » Iter 1. $4 * 1 = 4\ (< n)$
        - » Iter 2. $2 * 3 = 6\ (< n)$
        - » Iter 3. $1 * 7 = 7\ (< n)$
      - It can be approximated to $n$
    - Time complexity analysis
      - $f(n) = n \log_2 n = \mathrm{O}(n \log n)$

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Sequential searching
    - Let $X$ be a $n$ integer list $\{x_1, x_2, \ldots, x_n\}$ and $x_t$ be a target value



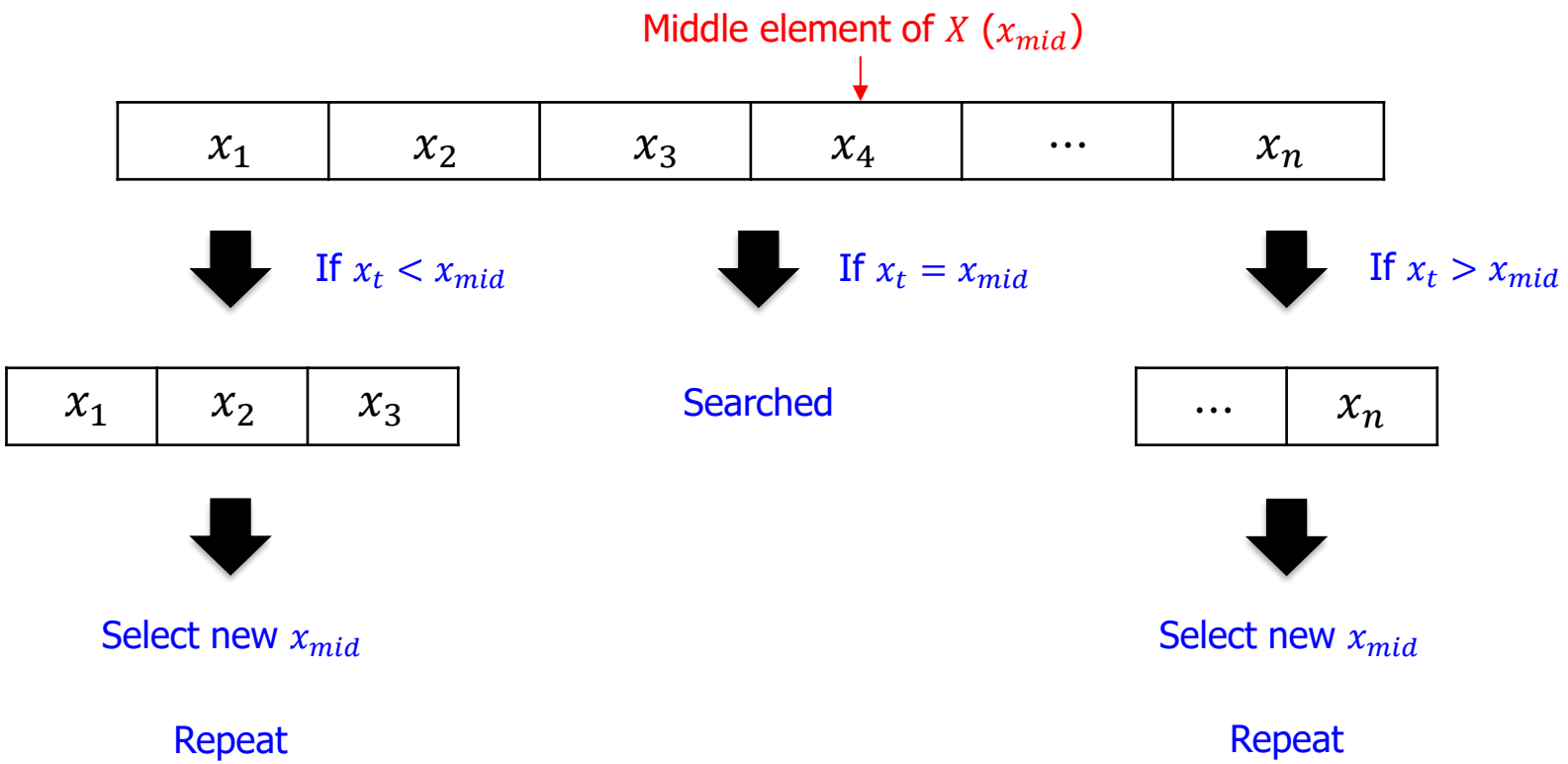

Sequentially search for an element in $X$
that have the same value as $x_t$

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Sequential searching (cont'd)
    - The number of comparisons:
      - $n$
    - Time complexity analysis
      - $f(n) = n \rightarrow O(n)$

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Binary searching
    - Let $X$ be a $n$ integer list $\{x_1, x_2, \ldots, x_n\}$ and $x_t$ be a target value

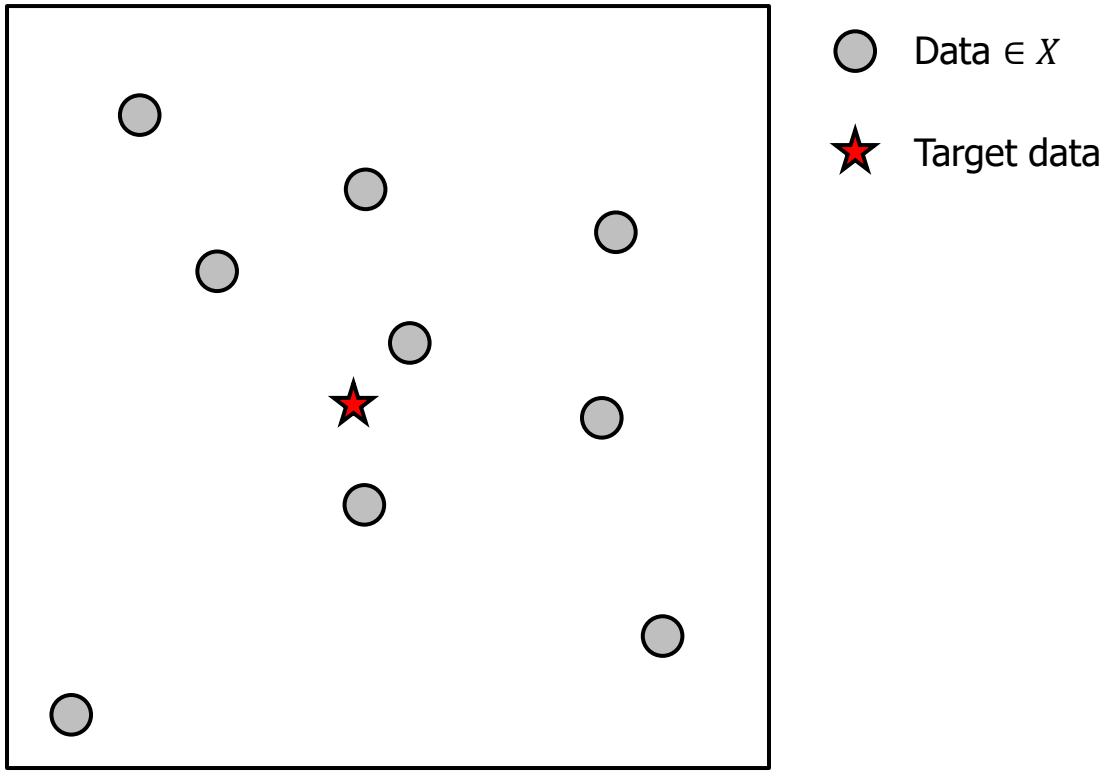


# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. Binary searching (cont'd)
    - The number of comparisons:
      - $\log_2 n$
    - Time complexity analysis
      - $f(n) = \log_2 n \rightarrow O(\log n)$
    - Constraint
      - The list $X$ must be sorted in ascending order

# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. $k$-nearest neighbor ($k$NN) search
    - Let $X$ be a $d$-dimensional dataset and $x_t$ be a target data



# Algorithm Analysis using Big-Oh Notation

- Example
  - Problem. $k$-nearest neighbor ($k$NN) search (cont'd)
    - Number of distance calculations:
      - $n$
    - Distance sorting
      - $n \log_2 n$
    - $k$-nearest neighbor selection
      - $k$
    - Time complexity analysis
      - $f(n) = n + n \log_2 n + k$
      - $\cong n + n \log_2 n$
      - $f(n) = O(n \log n)$

# Summary

- Performance evaluation
  - Running time
  - Confusion matrix
    - Accuracy
    - Precision
    - Recall
    - F1-score
  - MAE, MSE

- Comparison of algorithms
  - Theoretical comparison
  - Experimental comparison
    - Fair comparison

Questions?

# SEE YOU NEXT TIME!