FAKULTAS
ILMU
KOMPUTER

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

# Basic Algorithm Analysis (2)

A Survey of Common Running Time,

Algorithm Analysis Case: Insertion Sort

DAA Term 2 2023/2024

# A Survey on Common Running Times



- Constant Time
- Linear Time
- Linearithmic Time ($O(n \lg n)$)
- Quadratic Time
- Cubic Time
- Exponential Time

# A Survey: Constant Time ($O(1)$)

- This running time is bounded by a constant and does not depend on the input size.

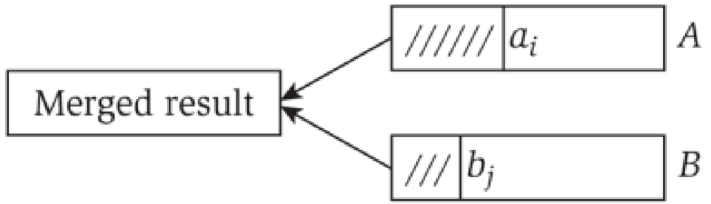
- Examples:
  - Conditional branch
  - Arithmetic/logic operation
  - Declaration/Initialization of a variable
  - Follow a link in a linked list
  - Access element in an array
  - …

# A Survey: Linear Time ($O(n)$)

- This running time is at most constant factor times the input size $n$.
- Example 1: Find **maximum number** of $a_1, a_2, \dots\ a_n$

```
max ← a1
for i = 2 to n {
   if (ai > max)
      max ← ai
}
```

- Example 2: **Merge two sorted lists** $\mathrm{A} = a_1, a_2, \dots\ a_n$ and $\mathrm{B} =$




```
i = 1, j = 1
while (both lists are nonempty) {
   if (ai ≤ bj) append ai to output list and increment i
   else         append bj to output list and increment j
}
append remainder of nonempty list to output list
```

# A Survey: Logarithmic Time ($O(\lg n)$)



- Example: Binary Search (find the index of an integer $x$ from a sorted array consists of $n$ distinct elements).

```
lo ← 1; hi ← n
while (lo ≤ hi)
        mid ← floor((lo + hi)/2)
        if (x < A[mid]) hi ← mid – 1
        else if (x > A[mid]) lo ← mid + 1
        else return mid
return -1
```

After $k$ iterations of WHILE loop, $(hi - lo + 1) \leq \frac{n}{2^k} \Rightarrow k \leq 1 + \lg n$

# A Survey: Linearithmic Time ($\mathrm{O}(n \lg n)$)



- This running time occurs on divide and conquer paradigm.
- Examples:
  - Merge sort and Heapsort

  - Largest Empty Interval (Given $n$ time-stamps $x_1, \dots, x_n$ on which copies of a file arrive at a server, what is largest interval when no copies of file arrive?)

    **Solution (in $\mathbf{O}(\boldsymbol{n} \lg \boldsymbol{n})$):**

    Sort the time stamps, scan the sorted list on order, identify the maximum gap between successive time-stamps.

# A Survey: Quadratic Time ($O(n^2)$)

- This running time is obtained by enumerate all pairs of elements
- Examples:
  - Bubble sort, Insertion sort in the worst-case scenario
  - Closest Pair of Points (Given a list of points in a plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is closest to each other.

```
min ← ∞
for i = 1 to n
        for j = i+1 to n
                d ← (xi−xj)² + (yi−yj)²
                if (d < min)
                        min ← d
```

# A Survey: Cubic Time ($O(n^3)$)

- Example:
  - 3-Sum (Given an array of $n$ distinct integers, find three that sum to 0)
    - Enumerate all triples $i$, $j$, and $k$ with $i < j < k$
  - Set disjointness (Given $n$ sets $S_1, S_2, .. S_n$ each of which is subset of 1,2,3,…n. is there some pair of these which are disjoint?)

```
foreach set Si {
   foreach other set Sj {
      foreach element p of Si {
         determine whether p also belongs to Sj
      }
      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
   }
}
```

$O(n^3)$ for each pair of sets, to determine if they are disjoint

# A Survey: Exponential Time



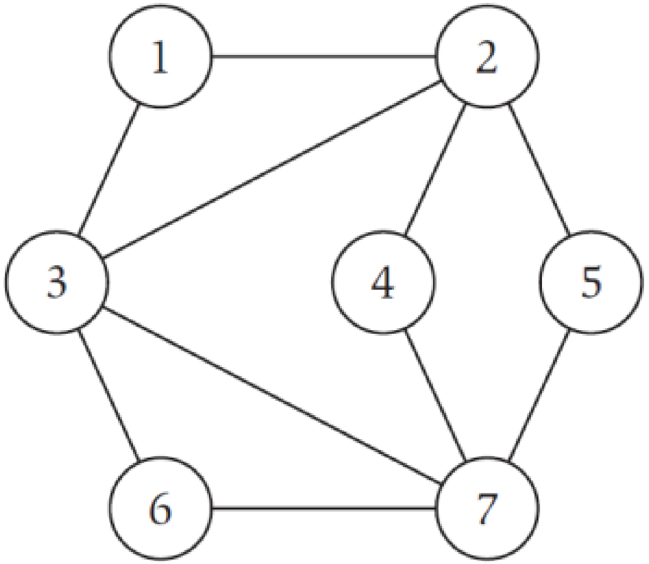
- Given a graph, what is maximum size of independent set?

An ***independent set*** of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in $E$ is incident on at most one vertex in $V'$. The ***independent-set problem*** is to find a maximum-size independent set in $G$.

Or we can say that V' is an independent set if no two nodes in V' are adjacent.

The maximum-size of an independent set from

This graph is 4, achieved by V'= {1,4,5,6}



# A Survey: Exponential Time

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

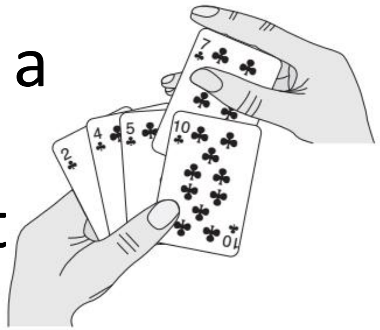
- Independent Set's Solution in $O(n^2 2^n)$ by enumerating all subsets

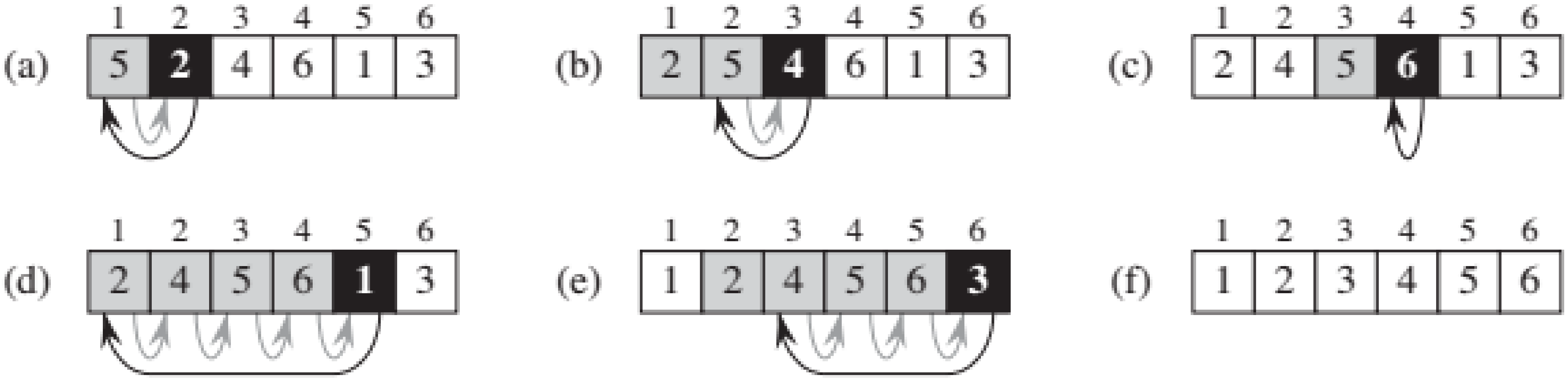
```
S* ← ϕ
foreach subset S of nodes {
   check whether S in an independent set
   if (S is largest independent set seen so far)
      update S* ← S
   }
}
```

# Algorithm Analysis Case: Insertion Sort

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

- Insertion sort is an **in-place sorting** algorithm, ood for sorting a **small number of elements,** it works the way you might sort a hand of playing cards. The running time depends on the input characteristic (e.g. sorted or not).





# Running Time of Insertion Sort

- Insertion sort of an array *A* length *n*

| | |
|---|---|
| `1. for j = 2 to length(A):` | $c_1 * n$ |
| `2.     key = A[j]` | $c_2 * (n-1)$ |
| `3.     //insert A[j] into the sorted sequence A[1..j-1]` | $c_3 * (n-1) = 0 \ (c_3=0)$ |
| `4.     i = j - 1` | $c_4 * (n-1)$ |
| `5.     while i > 0 and A[i] > key:` | $c_5 * (\sum_{j=2}^{n} t_j)$ |
| `6.         A[i+1] = A[i]` | $c_6 * (\sum_{j=2}^{n} (t_j - 1))$ |
| `7.         i = i - 1` | $c_7 * (\sum_{j=2}^{n} (t_j - 1))$ |
| `8.     A[i+1] = key` | $c_8 * (n-1)$ |

$t_j$ is the number of times statement in line 5 is executed for the value of *j*

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\sum_{j=2}^{n} t_j\right) + c_6\left(\sum_{j=2}^{n}(t_j - 1)\right) + c_7\left(\sum_{j=2}^{n}(t_j - 1)\right) + c_8(n-1)$$

# Running Time of Insertion Sort: **Best Case**

- It runs in ***linear time***
    - When the input array is already sorted ($t_j = 1$ for $j = 2,3,4,\dots n$)
    - The statements inside the inner loop are not executed

| A[1] | A[2] | A[3] | A[4] | … | A[n] |
|---|---|---|---|---|---|

key=A[2]

When $j$ = 2, i > 0 and A[1] < key → while loop terminates

| A[1] | A[2] | A[3] | A[4] | … | A[n] |
|---|---|---|---|---|---|

key=A[3]

When $j$ = 3, i > 0 and A[2] < key → while loop terminates

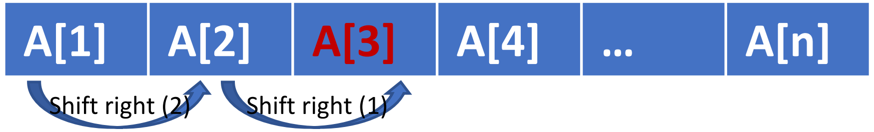
… and so on

- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\sum_{j=2}^{n} 1\right) + c_6\left(\sum_{j=2}^{n} 0\right) + c_7\left(\sum_{j=2}^{n} 0\right) + c_8(n-1)$
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$
- $\boldsymbol{T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)}$

# Running Time of Insertion Sort: **Worst Case**

- It runs in *quadratic time*
  - When the input array is in reverse sorted order ($t_j = j$ for $j$ = 2,3,4,…$n$)
  - The statements inside inner loop are executed $j$-1 times


When $j$ = 2, i > 0 and A[1] > key → shift right 1 time




When $j$ = 3, i > 0 and A[2] > key → shift right 2 times

… and so on

- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\sum_{j=2}^{n} j\right) + c_6\left(\sum_{j=2}^{n}(j-1)\right) + c_7\left(\sum_{j=2}^{n}(j-1)\right) + c_8(n-1)$
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$
- $\boldsymbol{T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)}$

# Discussion

- What is the Average case running time of Insertion sort?
- The running time of Insertion Sort belongs to both $\Omega(n)$ and $O(n^2)$. True or False?
- Running time of Insertion Sort is $\Omega(n^2)$. True or False?
- Worst case running time of Insertion Sort is $\Omega(n^2)$ True or False?

# References

- Lecturer Slides by Bapak L. Yohanes Stefanus
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
- https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/02AlgorithmAnalysis.pdf