MSMC-204: Design and Analysis of Algorithms

M.Sc. in Mathematics and Computing II-Semester Session 2024-25

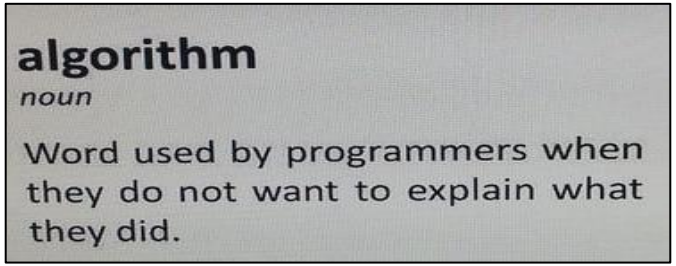
LECTURE 1: by Dr. Supriya Chanda

- Introduction

Logistics

- **Course Name and code**: Design and Analysis of Algorithms, MSMC-204
- **Time**: Saturday (1:00 PM-3:00 PM), Extraday (1:00 PM-2:00 PM)
- **Venue**: DST-CIMS Class Room
- **Course website**: I will update soon
- **Prerequisites**: Knowledge of Programming, Data Structures.
- **Books and References**:
  1. Thomas H Cormen, Charles E Lieserson, Ronald L Rivest and Clifford Stein, Introduction to Algorithms. (4^{th} Edition)
  2. More references specific to the topic will be added in the course website when needed.

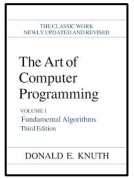
What are algorithms?



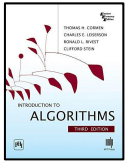
Source: http://redd.it/b5mtc7

Dictionary definition: A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.

By Donald E Knuth: An algorithm is a finite, definite, effective procedure, with some input and some output.



By Cormen *et al.*: An algorithm is a sequence of computational steps that transform the input into the output.



Why is the study of algorithms worthwhile?

- Especially, modern computers are pretty fast and memory cost is low
- (If for no other reason) – show that your algorithm produces the right answer always
- Different algorithms for the same problem often differ dramatically in terms of efficiency
- Before showing an example, lets step into the practical world!

Why is the study of algorithms worthwhile?

- Typical job interview process, e.g., Google's software developer, Amazon, Microsoft…

- Round 1: online assessment (~90 Minutes)
  - Two data structures and algorithms questions you have to complete
  - Write your own test cases and must pass all test cases that you cannot see
  - You can use any IDE of your choice
  - The recruiter will not review your resume if you do not pass the OA

- Round 2: Technical phone interview
  - Solve data structure and algorithm questions
  - Write code (usually in txt/cloud doc) and explain the correctness/time complexity
  - Additional behavioral questions that are usually not decisive

Why is the study of algorithms worthwhile?

- Typical job interview process, e.g., Google's software developer, Amazon, Microsoft…

  - Round 3: Onsite interviews (May not be applicable in India)
    - Spend a full day at a Google office and do four to six interviews.
    - Topics asked are usually data structure and algorithm and system design
    - You are expected to do extremely well in coding interviews

  - Of course algorithms are also important for doing research in all CS areas

Why is the study of algorithms worthwhile?

- Lets see an example of variation of efficiency

- Suppose you want to sort n numbers
  - Algorithm 1 takes $c_1 n^2$ units of time
  - Algorithm 2 takes $c_2 n \log n$ units of time
  - c_1 and c_2 are constants and not dependent on n

- Let computer A execute 10 billion operations/second and computer B execute 10 million operations/second

- Let Algorithm 1 is run on computer A while Algorithm 2 is run on computer B

- Let Algorithm 1 is translated into code really beautifully and Algorithm 2 is not.
  - This results in $c_1 = 2$ and $c_2 = 50$

Why is the study of algorithms worthwhile?

- To sort $n = 10$ million numbers, Computer A takes
  $\frac{2 \times (10^7)^2}{10^{10}} s = 2 \times 10^4 \approx 5.55$ hrs

- For the same, Computer B takes
  $\frac{50 \times 10^7 \log 10^7}{10^7} s \approx 1163\ s < 20$ mins

Analysis of algorithms

The theoretical study of computer-program performance and resource usage.

What is more important than performance?

- modularity
- correctness
- maintainability
- functionality
- robustness
- user-friendliness
- programmer time
- simplicity
- extensibility
- reliability

Why study algorithms and performance?

- Algorithms help us to understand ***scalability***.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
- Performance is the ***currency*** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Insertion Sort

"pseudocode"

```
INSERTION-SORT (A, n) ▷ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while  i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```

Pseudocode

- But before that – what is a pseudocode and how is it different from a real code?
  - Pseudocode does not bother much about language specific synatxes
  - Whatever helps to express the idea is used
  - Is not concerned with issues of software engineering

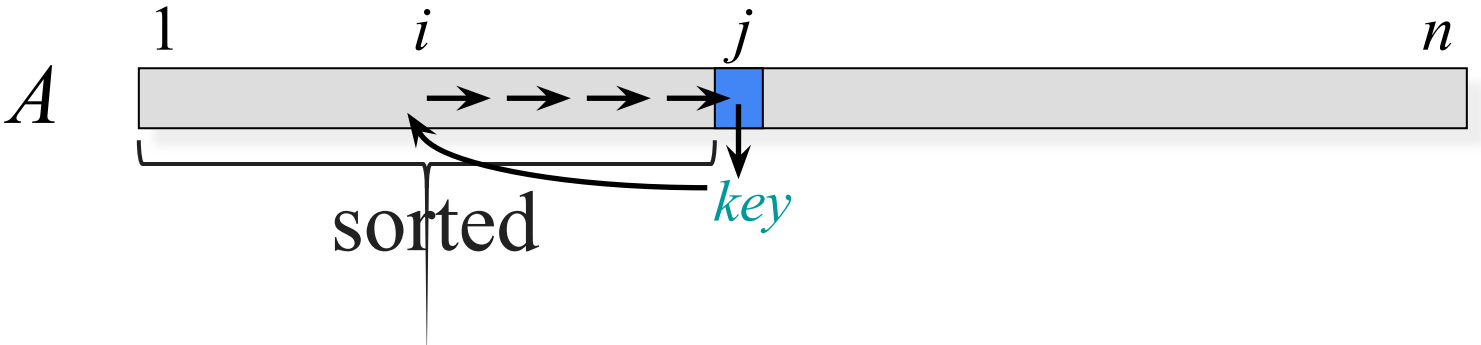
```
#include <stdio.h>

int main()
{
    int a, b, sum;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);
    sum = a + b;
    printf("Sum: %d\n", sum);
}
```

```
START
    Declare variables x, y, and max
    Display "Enter two numbers"
    Input x and y
    IF x is greater than y THEN
        max = x
    ELSE
        max = y
    END IF
    Display "Maximum is" and the value of max
END
```

Insertion Sort

"pseudocode"

```
INSERTION-SORT (A, n)  ▷ A[1 . . n]
    for   j ← 2 to n
          do   key ← A[ j]
               i ← j – 1
               while      i > 0 and A[i] > key
                     do   A[i+1] ← A[i]
                          i ← i – 1
               A[i+1] = key
```



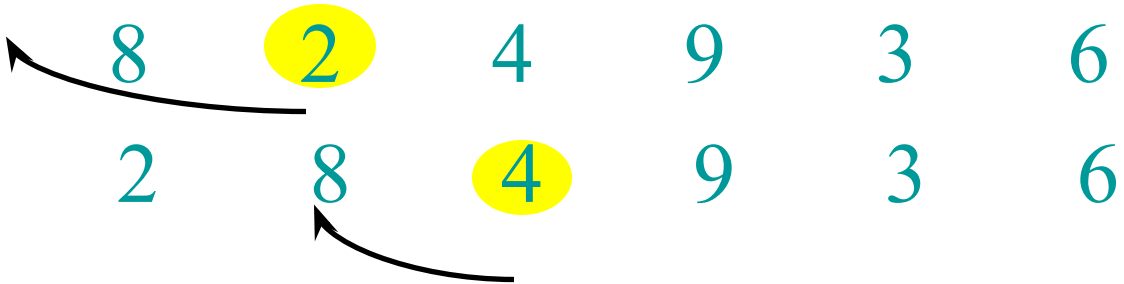
Example of insertion sort

8 2 4 9 3 6

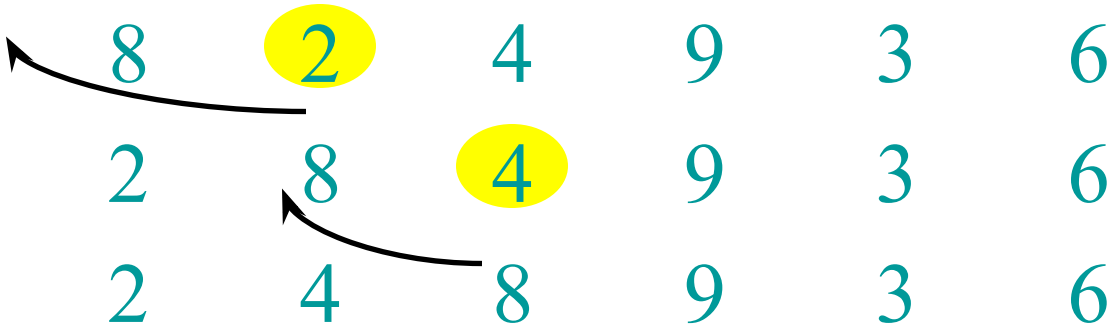
Example of insertion sort

Example of insertion sort

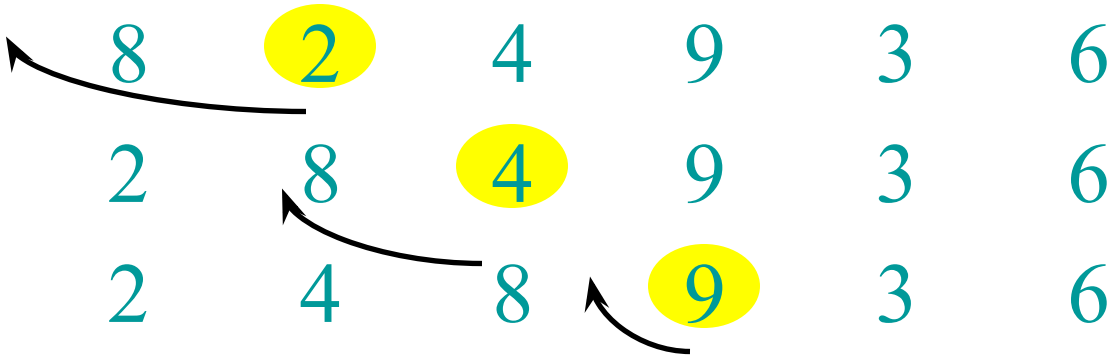
Example of insertion sort



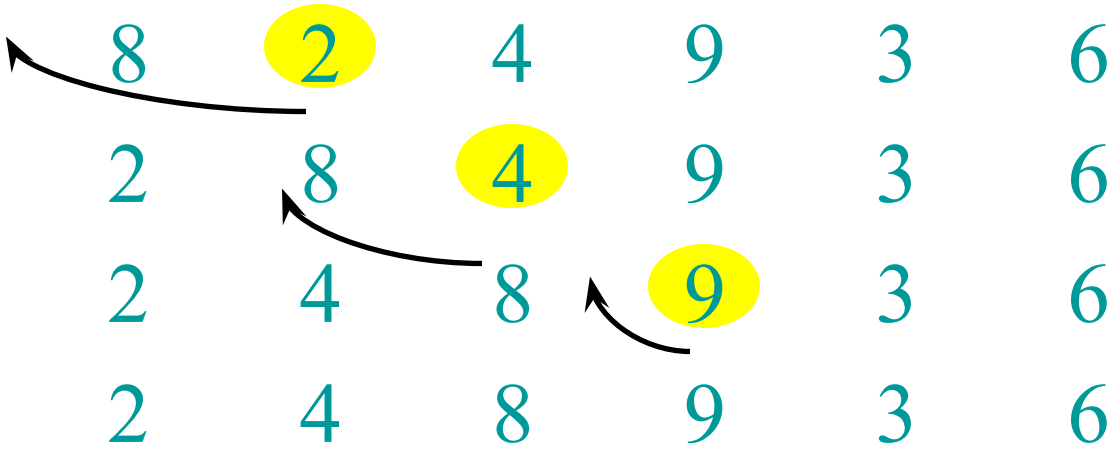
Example of insertion sort



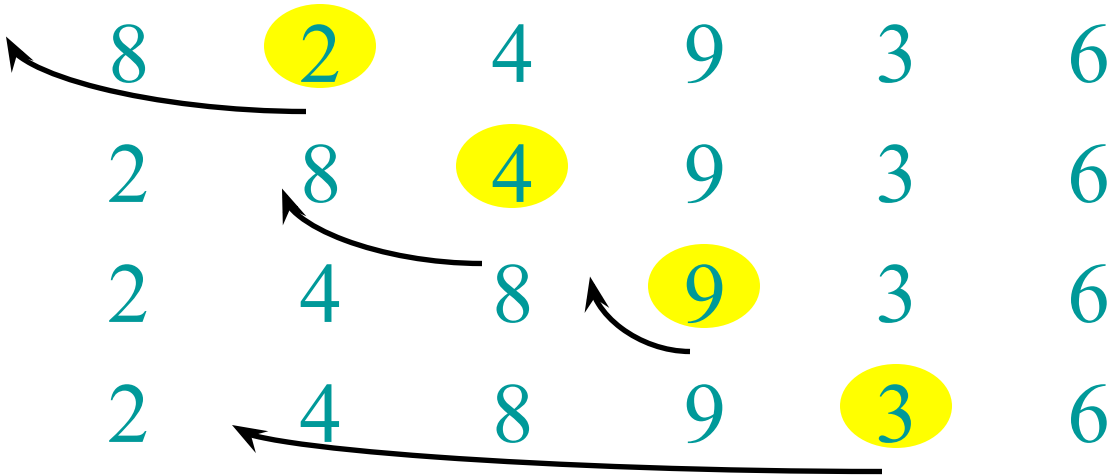
Example of insertion sort



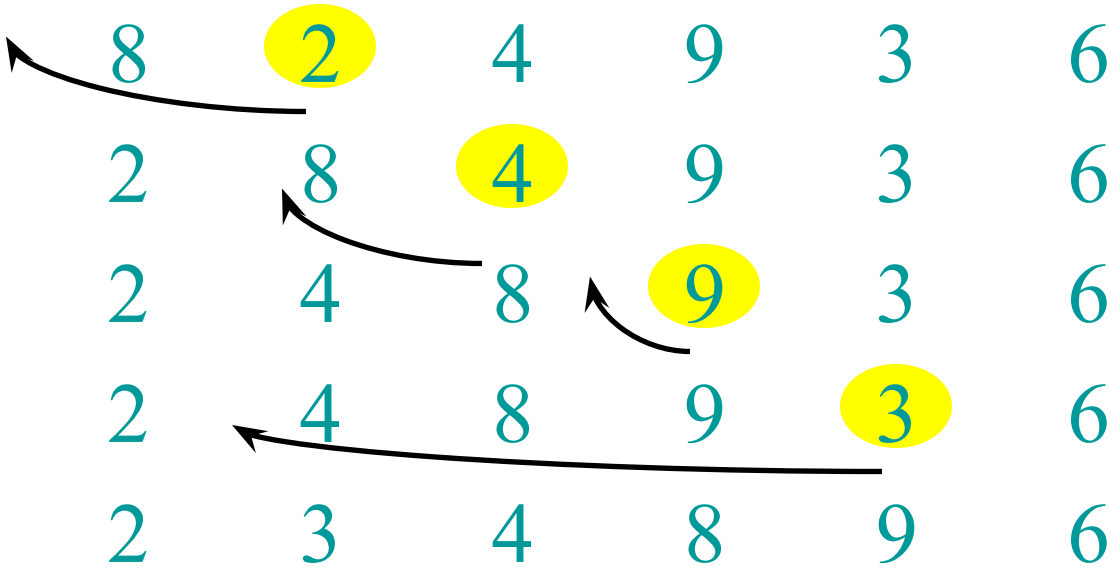
Example of insertion sort



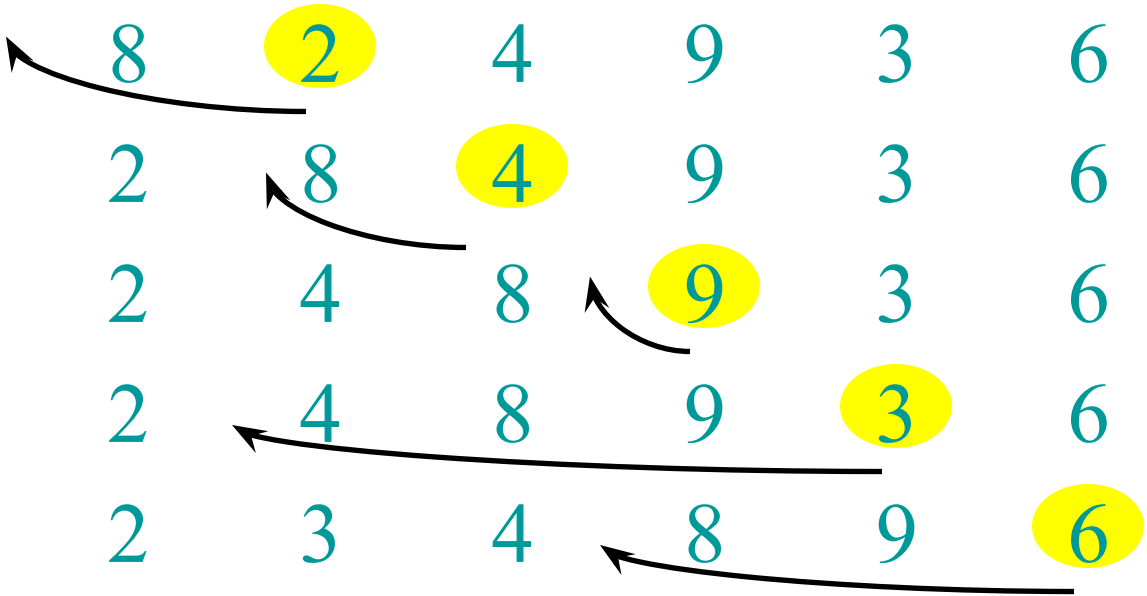
Example of insertion sort



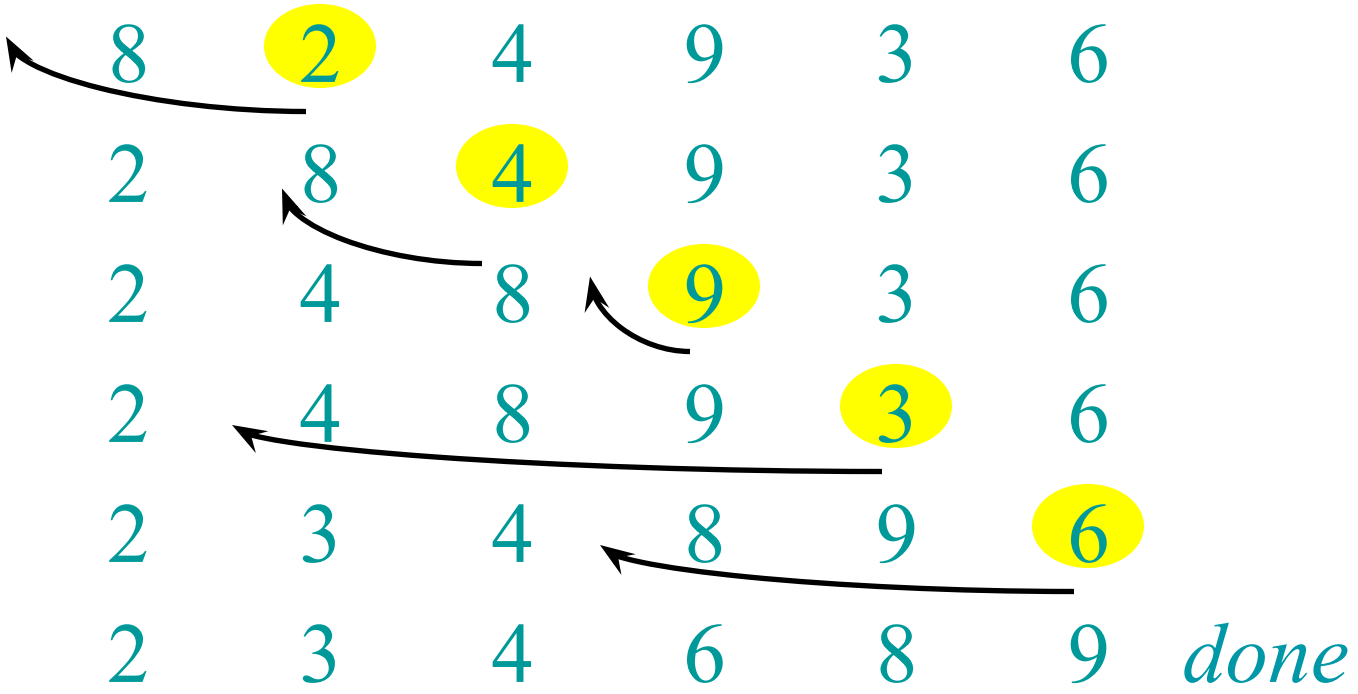
Example of insertion sort



Example of insertion sort



Example of insertion sort



Algorithm Analysis

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation.

1. **A Priori Analysis** − This is a theoretical analysis of an algorithm.

2. **A Posterior Analysis** − This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine.

Algorithm Design Techniques

The following is a list of several popular design approaches:

1. **Divide and Conquer Approach:** It is a top-down approach. The algorithms which follow the divide & conquer techniques involve three steps:
   - Divide the original problem into a set of subproblems.
   - Solve every subproblem individually, recursively.
   - Combine the solution of the subproblems (top level) into a solution of the whole original problem.

2. **Greedy Technique:** Greedy method is used to solve the optimization problem. An optimization problem is one in which we are given a set of input values, which are required either to be maximized or minimized (known as objective), i.e. some constraints or conditions.
   - Greedy Algorithm always makes the choice (greedy criteria) looks best at the moment, to optimize a given objective.
   - The greedy algorithm doesn't always guarantee the optimal solution however it generally produces a solution that is very close in value to the optimal.

Algorithm Design Techniques

3. **Dynamic Programming:** Dynamic Programming is a bottom-up approach we solve all possible small problems and then combine them to obtain solutions for bigger problems.

4. **Branch and Bound:** In Branch & Bound algorithm a given subproblem, which cannot be bounded, has to be divided into at least two new restricted subproblems. Branch and Bound algorithm are methods for global optimization in non-convex problems. Branch and Bound algorithms can be slow, however in the worst case they require effort that grows exponentially with problem size, but in some cases we are lucky, and the method coverage with much less effort.

5. **Backtracking Algorithm:** Backtracking Algorithm tries each possibility until they find the right one. It is a depth-first search of the set of possible solution. During the search, if an alternative doesn't work, then backtrack to the choice point, the place which presented different alternatives, and tries the next alternative.

6. **Randomized Algorithm:** A randomized algorithm uses a random number at least once during the computation make a decision.

Example 1: In Quick Sort, using a random number to choose a pivot.