Algorithms and Complexity

Algorithm Analysis

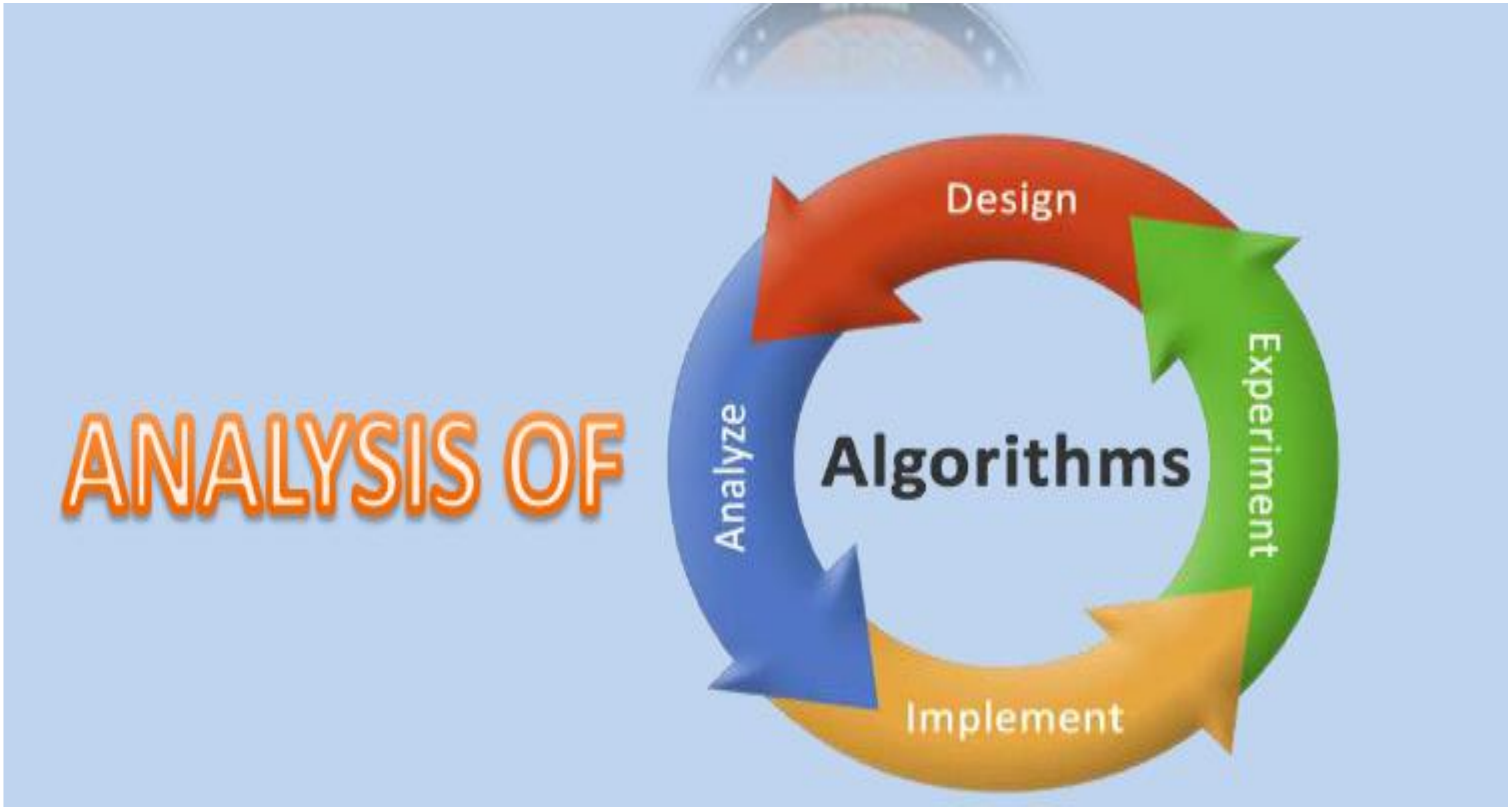
Lecturer: Dr. Alaa Ahmed Abbood
Lecture 3.
Class 2nd .
Time: 8:30-10:30
Department: Businesses Information Technology (BIT)

Outline

Lecture 3

- Analysis of Algorithms.
- Order of Growth
- Efficiencies of the Algorithm
- How to Determine Complexities?
- Exercises.

Analysis of Algorithms



Analysis of Algorithms

- Algorithm is step by step procedure to solve any problem.
- Analyzing / Judgment of the Algorithm

- An algorithm can be written in different ways for solving a single problem.
- So by analyzing the algorithms we can find the best solution (algorithm) to the problem.

Order of Growth

- Order of Growth

- Any algorithm is expected to work fast for any input size.
- For smaller input size our algorithm will work fine but for higher input size the execution time is much higher.
- By increasing the size of n (input size) we can analyze how well our algorithm works.

Order of Growth Cont.

- Let input size, n=5 and we have to sort the list of elements for e.g. 25,29,10,15,2
- So for n=5 our algorithm will work fine but what if n=5000?
- So our algorithm will take much longer time to sort the elements or cause small delays to give the result.
- So how the behaviour of algorithm changes with the no. of inputs will give the analysis of the algorithm and is called the ***Order of Growth.***

Order of Growth Cont.

- For calculating the order of growth we have to go for higher value of n, because:-

1. As the input size grow higher algorithm makes delays.
2. For all real time applications we have higher values for n.

Efficiencies of the Algorithm

❖ Efficiencies of the Algorithm

- There are three cases

1. Best case
2. Worst case
3. Average case

- Let we have 5 nos.(n=5) 25,31,42,71,105 and we have to find any element in the list.

Efficiencies of the Algorithm Cont.

❖ Best case efficiency

- let we have to find 25 in List =>25,31,42,71,105
- k=25
- 25 is present at the first position
- Since only single comparison is made to search
- the element so we say that it is best case efficiency
- CBest (n)=1

Efficiencies of the Algorithm Cont.

- Worst case efficiency
- If we want to search the element which is present at the last of the list or not present at all in the list then such cases are called the worst case efficiency.
- let we have to find 105 in List =>25,31,42,71,105
- k=105
- Therefore we have to make 5 (=n) comparisons to search the element
- CWorst (n)=n
- And if we have to find 110
- k=110
- Since the element is not in the list even then we have to make 5 (=n) comparisons to search the element.
- CWorst (n)=n

Efficiencies of the Algorithm Cont.

- Average case efficiency

- • Let the element is not present at the first or the last position
- Let it is present somewhere in middle of the list
  - We know that probability of a successful search = p where

$$\mathbf{0} \leq \boldsymbol{p} \leq \mathbf{1}.$$

- And probability of unsuccessful search = 1-p

Efficiencies of the Algorithm Cont.

❖ Average case efficiency

- Let the element we are searching for is present at position 'i' in the list
- Therefore probability of the element to be found is given by $\boldsymbol{p/n}$.
- Therefore CAvg (n)
- =[1*p/n + 2*p/n + ... + i*p/n + ... + n*p/n] + n(1-p)
- =p/n[1+2+...+i+...+n] + n(1-p)
- =p/n[n*(n+1)/2] + n(1-p)
- =p[(n+1)/2] + n(1-p)

Efficiencies of the Algorithm Cont.

❖ Average case efficiency

- **case 1.** If element is available therefore p=1 (for successful search)
- Now substituting p=1 .

$$\boldsymbol{CAvg\ (n)\ =\ 1[(n+1)/2]\ +\ n(1-1)\ =\ (n+1)/2}$$

- **case 2**. If element is unavailable therefore p=0 (for unsuccessful search)
- Now substituting p=0 in above eqn

$$\boldsymbol{CAvg\ (n)\ =\ 0[(n+1)/2]\ +\ n(1-0)\ =\ n}$$

- Therefore on average half of list will be searched to find the element in the list

How to Determine Complexities

- ❖ In general, how can you determine the running time of a piece of code? The answer is that it depends on what kinds of statements are used.

1. Sequence of statements

- statement 1;
- statement 2;
- statement k;

How to Determine Complexities cont.

- The total time is found by adding the times for all statements:
- Total time = time (statement 1) + time (statement 2) + ... + time (statement k)
- If each statement is "simple**" (only involves basic operations)** then the time for each statement is constant and the total time is also constant: O(1). In the following examples, assume the statements are simple unless noted otherwise.

How to Determine Complexities cont.

- **if-then-else statements**

- if (condition) {
- Sequence of statements 1

        }

- else {
- sequence of statements 2

  }

How to Determine Complexities cont.

❖ **if-then-else statements**

- Here, either sequence 1 will execute, or sequence 2 will execute. Therefore, the worst-case time is the slowest of the two possibilities: max (time (sequence 1), time (sequence 2)).
- For example, if sequence 1 is O(N) and sequence 2 is O(1) the worst-case time for the whole if-then-else statement would be O(N).

How to Determine Complexities cont.

❖ **For Loops**

- for (i = 0; i < N; i++) {

  sequence of statements }

- The loop executes N times, so the sequence of statements also executes N times. Since we assume the statements are O(1), the total time for the for loop is N * O(1), which is O(N) overall.

How to Determine Complexities cont.

❖ **Nested loops**

- First we'll consider loops where the number of iterations of the inner loop is independent of the value of the outer loop's index. For example:
- for (i = 0; i < N; i++) {
- for (j = 0; j < M; j++) {

  sequence of statements

  }

}

How to Determine Complexities cont.

❖ **Nested loops**

- The outer loop executes N times. Every time the outer loop executes, the inner loop executes M times. As a result, the statements in the inner loop execute a total of N * M times.
- Thus, the complexity is O(N * M). In a common special case where the stopping condition of the inner loop is j < N instead of j < M (i.e., the inner loop also executes N times), the total complexity for the two loops is O(N2).

How to Determine Complexities cont.

❖ **Nested loops**

- Now let's consider nested loops where the number of iterations of the inner loop depends on the value of the outer loop's index. For example:
- for (i = 0; i < N; i++) {
- for (j = i+1; j < N; j++) {

  sequence of statements

  }

}

How to Determine Complexities cont.

❖ **Nested loops**

- Now we can't just multiply the number of iterations of the outer loop times the number of iterations of the inner loop, because the inner loop has a different number of iterations each time.
- So let's think about how many iterations that inner loop has.
- That information is given in the following table (Next slide)

How to Determine Complexities cont.

❖ **Nested loops**

- Value of i Number of iterations of inner loop

i	Number of iterations
0	N
1	N-1
2	N-2
N-2	2
N-1	1

- So we can see that the total number of times the sequence of statements executes is: N + N-1 + N-2 + ... + 3 + 2 + 1. We've seen that formula before: the total is $O(N^2)$.

How to Determine Complexities cont.

❖ **Statements with method calls:**

- When a statement involves a method call, the complexity of the statement includes the complexity of the method call.
- Assume that you know that method ***f*** takes constant time, and that method ***g*** takes time proportional to (linear N) the value of its parameter ***k***. Then the statements below have the time complexities indicated.

How to Determine Complexities cont.

❖ **Statements with method calls:**

- f(k); // O(1)
- g(k);   // O(k)
- When a loop is involved, the same rule applies.

For example:

- for (j = 0; j < N; j++) g(N);
- has complexity (N). The loop executes N times and each method call g(N) is complexity O(N).

Exercises:

1. **What is the worst-case complexity of the each of the following code fragments?**

1. Two loops in a row:

```
▪ for (i = 0; i < N; i++) {
          sequence of statements
}
▪ for (j = 0; j < M; j++) {
          sequence of statements
  }
```

How would the complexity change if the second loop went to N instead of M?

Exercises:

1. **What is the worst-case complexity of the each of the following code fragments?**

❖ **Solution**

- The first loop is O(N) and the second loop is O(M). Since you don't know which is bigger, you say this is O(N+M). This can also be written as O(max(N,M)).
- In the case where the second loop goes to N instead of M the complexity is O(N). You can see this from either expression above.
- When N is used instead of M then O(N+M) becomes O(2N) and when you drop the constant it is O(N). O(max(N,M)) becomes O(max(N,N)) which is O(N).

Exercises:

2. A nested loop followed by a non-nested loop:

- for (i = 0; i < N; i++) {

    for (j = 0; j < N; j++) {

- sequence of statements

    }

}

- for (k = 0; k < N; k++) {
- sequence of statements

}

Exercises:

2. A nested loop followed by a non-nested loop:

Solution

- The first set of nested loops is $O(N^2)$ and the second loop is $O(N)$.
- This is $O(max(N^2,N))$ which is $O(N^2)$.

Exercises:

3. **A nested loop in which the number of times the inner loop executes depends on the value of the outer loop index:**

- for (i = 0; i < N; i++) {
- for (j = N; j > i; j--) {
- sequence of statements

          }

}

Exercises:

3. A nested loop in which the number of times the inner loop executes depends on the value of the outer loop index:

Solution

- This is very similar to our earlier example of a nested loop where the number of iterations of the inner loop depends on the value of the index of the outer loop.
- The only difference is that in this example the inner-loop index is counting down from N to i+1. It is still the case that the inner loop executes N times, then N-1, then N-2, etc, so the total number of times the inner most "sequence of statements" executes is $O(N^2)$.

THANK YOU