

Data Structures

Algorithms and complexity analysis

Chapter 2_1

Relationship between data structures and algorithms

- The primary objective of programming is to efficiently process the input to generate the desired output
- We can achieve this objective in an efficient and neat style if the input data is organized properly
- Data Structures is nothing but ways and means of organizing data so that it can be processed easily and efficiently
- Data structures dictate the manner in which the data can be processed
- In other words, the choice of an algorithm depends upon the underlying data organization

Data Organization 1

→ Fail in case of large data set

Data are stored in 10 disjoint variables A0, A1, A2, ..., and A9

Algorithm:

```
found = false;
if (key == A0) found = true;
else if (key == A1) found = true;
else if (key == A2) found = true;
else if (key == A3) found = true;
else if (key == A4) found = true;
else if (key == A5) found = true;
else if (key == A6) found = true;
else if (key == A7) found = true;
else if (key == A8) found = true;
else if (key == A9) found = true;
```

Best Case

* Scalability

* Efficiency is not an issue

Worst Case

Data Organization 2

Unordered
Data

Data are stored in an array A of 10 elements

Algorithm:

```
const int N = 10;
found = false;
for (int i = 0; i < N; i++)
{
    if (A[i] == key){
        found = true;
        break;
    }
}
```

loop condition

increment

→ Linear Search

→ Scalability ✓

check to find element

Data Organization 3

Data are stored in an array A in ascending order

Algorithm:

```
const int N = 10;  
found = false;  
low = 0;  
high = N - 1;  
while (( ! found) && ( low <= high)){  
    ✓ mid = (low + high)/2;  
    if (A[mid] == key)  
        ✓ found = true;  
    else if (A[mid] > key)  
        → high = mid - 1;  
    else  
        → low = mid + 1;  
}
```

Two loop condition

Binary search
Way more
efficient than
linear search

Calculation

Two checks
for data

Comparison of Organizations

Which one of the three organizations is better and why?

- We can easily see that the second one is better than the first:
 - In both the cases the number of comparisons will remain the same but the second algorithm is more scalable as it is much easier to modify the second one to handle larger data sizes as compared to the first one
 - For example, if the data size was increased from 10 to 100, the first algorithm would require declaration of 90 more variable, and adding 90 new else-if clauses. Whereas, in the second case, all one has to do is to redefine N to be equal to 100. Nothing else needs to change.

Comparison Between Algorithm 2 and 3

- Second algorithm (linear search) is much simpler as compared to the third one (binary search)
- Furthermore, the linear search algorithm imposes less restriction on the input data as compared to the binary search algorithm as it requires data sorted in ascending or descending order
- From an execution point of view, the body of the while loop in the linear search seems to be more efficient as compared to the body of the loop in the binary search algorithm
- So, from the above discussion it appears that linear search algorithm is better than the binary search algorithm
- So, why should we at all bother about the binary search algorithm?
- Does it offer any real benefit over linear search?

NADRA Database Example

- National Database and Registration Authority (NADRA), Government of Pakistan, maintains a database of the citizens of Pakistan which currently has over 80 million records and is growing incessantly
- Suppose we need to search for a record in this database
- Let us also assume that we have a computer that can execute 100,000 iteration of the body of the while loop in the linear search algorithm in one second
- It is also assumed that, on the average, each iteration of the body of the while loop in the binary search algorithm takes 3 times more than one iteration of the while loop in the linear search algorithm

33,000 iterations

NADRA Database Example (In Numbers)

	Linear Search	Binary Search
Number of iterations of the body of the loop in	100,000	~33000
Searching a record in 80 Million records:		
Worst case – record is not found	800 seconds	~0.0008 seconds
Average case	400 seconds	~0.0004 seconds
Number of searches in one hour (average case)	9	~ 9 million
Number of searches per day	216	~ 213 Million

Complexity of Algorithms

- in general, binary search is much faster than the linear search and with increase in data size (e.g. number of records in the database) the difference will become wider and even more staggering
- How do we get to that conclusion?
- The general question is: given two algorithms for the same task, how do we know which one will perform better?
- This question leads to the subject of complexity of algorithms

Algorithm Analysis

- Efficiency of an algorithm can be measured in terms of the execution time and the amount of memory required Space Comp
- Time Comp Depending upon the limitation of the technology on which the algorithm is to be implemented, one may decide which of the two measures (time and space) is more important
- For most of the algorithms associated with this course, time requirement comparisons are more interesting than space requirement comparisons and hence we will concentrate more on the that aspect

Algorithm Analysis (continued)

- Comparing two algorithms by actually executing the code is not easy
- To start with, it would require implementing both the algorithms on similar machines, using the same programming language, same compiler, and same operating system
- Then we need to generate a large number of data sets for profiling the algorithms
- We can then try to measure the performance of these algorithms by executing them
- These are no easy tasks and in fact are sometimes infeasible
- Furthermore, a lot depends upon the input sequence

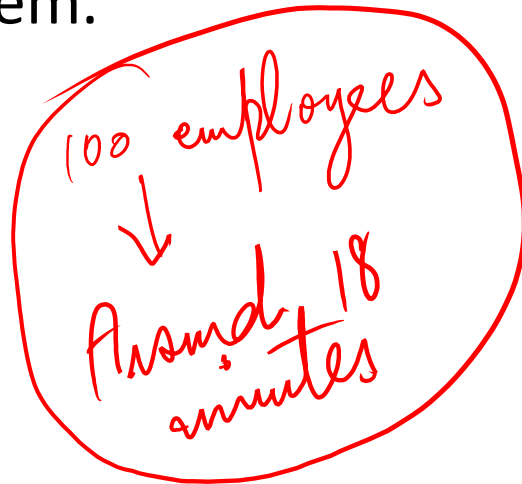
Algorithm Analysis (continued)

- For the same amounts of input data with a different permutation, an algorithm may take drastically different execution times to solve the problem.
- Therefore, with this strategy, whenever we have a 'new' idea to solve the same problem, we cannot answer the question whether the program's running time would be acceptable for the task or not without implementing it and then comparing it with other algorithms by executing all these algorithms for a number of data sets
- This is rather a very inefficient, if not absolutely infeasible, approach. We therefore need a mathematical instrument to estimate the execution time of the algorithm without having to implement the algorithm.

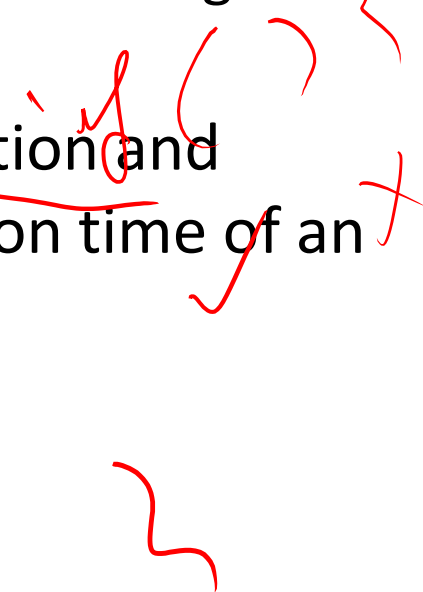
3, 6, 2, 1, 4
3, 4, 1, 2, 6
3, 6, 2, 1, 4
Execution?

Algorithm Analysis (continued)

- Unfortunately, in the real world, coming up with an exact and precise model of the system under study is an exception rather than a rule.
- For many problems that are worth studying it is not easy, if not impossible, to model the problem with an exact formula.
- In such cases we have to work with an approximation of the precise problem.



Algorithm Analysis (continued)

- Model for execution time for algorithms falls in the same category because:
 - each statement in an algorithm requires different execution time, and
 - presence of control flow statements renders it impossible to determine which subset of the statements in the algorithm will actually be executed
 - We therefore need some kind of simplification and approximation for determining the execution time of an algorithm
- 

Time complexity analysis

$$T(n) = \frac{n+5}{100} \checkmark$$

constant

In the time complexity analysis we define the time as a function of the problem size and try to estimate the growth of the execution time with the growth in problem size.

Consider linear search again:

```
const int N = 10;
found = false;
for (int i = 0; i < N; i++)
    if (A[i] == key){
        found = true;
        break;
    }
```

$$T(n) = 2n + 5$$

200 elements
100 elements
250

Problem Size

[Searching Data Strg
Sorting

[Prime Number
nth prime number
[Factorial