Session 02

Introduction to Algorithms

Overview

- What is an algorithm?
- Different algorithms for a problem: why?
- Algorithm performance: What to measure? How to measure?
- Order notation
- Example Problem: Maximal Subsequence
- Complexity classes

Different Algorithms?

- Usability of an algorithm depends on
  - space requirement
  - time requirement
  - characteristics of likely inputs
  - frequency of use, etc.
- Different algorithms for different situations.
- We need to
  - evaluate algorithms: strengths and weaknesses.
  - match algorithms with application requirements.

Efficiency of My Algorithm

- Run it and measure the time taken?
  - depends on speed of my machine.
  - depends on the load on my machine.
  - depends on data used to test.
  - depends on way of coding.
  - depends on how much tuning has been done.
  - and so on
- Is there a more reliable measure?

Estimating Run-time

- We need a measure based on the logic used.
- A *reasonable* computer with a *reasonable* set of instructions assumed.
- Each reasonable instruction assumed to take one unit of time.
- Analytically estimate runtime and measure variation of runtime against input size.

Example

Consider the code segment:

```
S = 0;
for x = 0 to N do
   S += a[x];
```

What is the runtime estimate?

`S = 0` => 1 unit

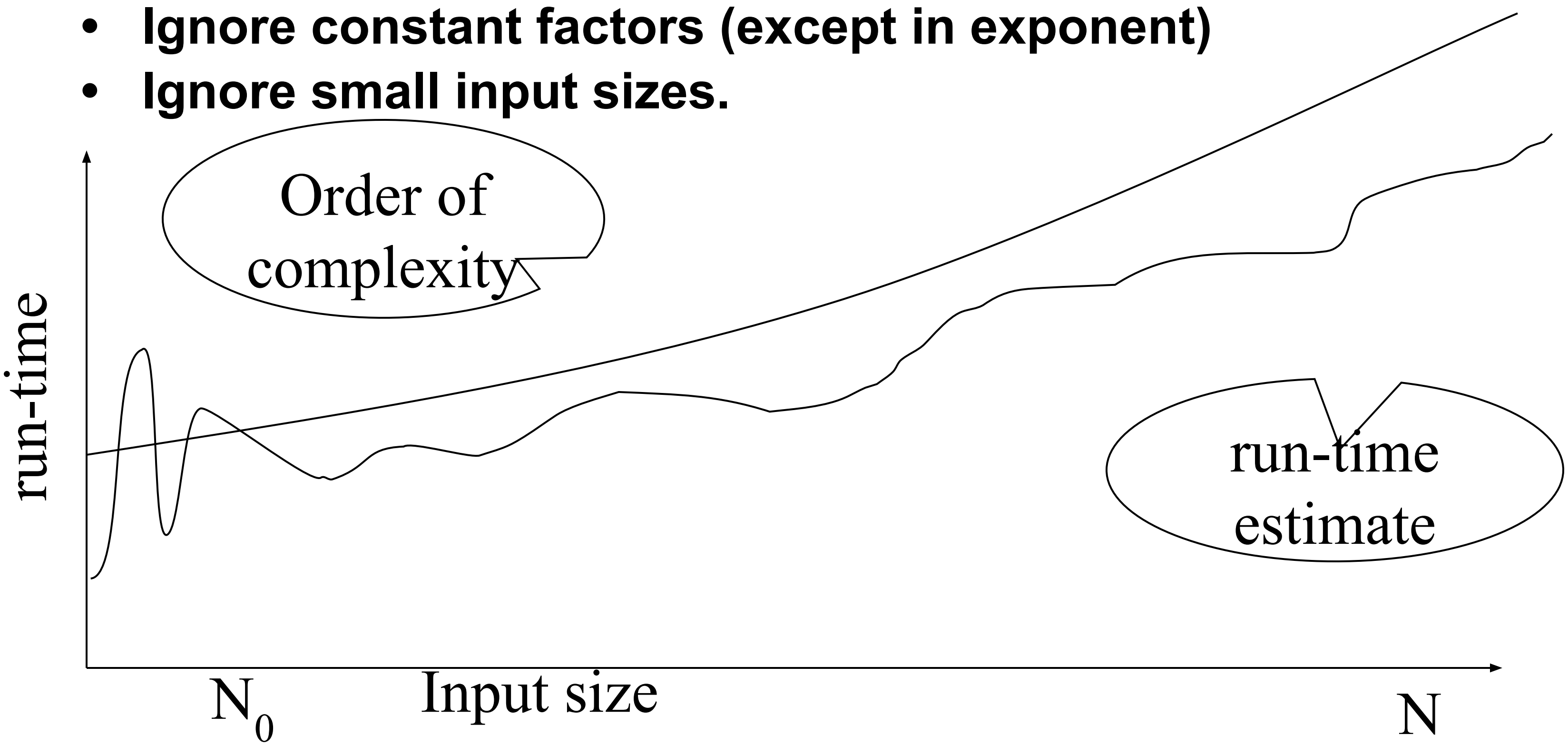
for-loop: `x = 0` => 1 unit, once

`x++, x < N, a[x], +=` => 4 units, N times

Hence: runtime = 4N+2

Order Notation

- **Upper Bound on runtime**
- **Ignore constant factors (except in exponent)**
- **Ignore small input sizes.**



Order Notation

T(n) is O(f(n)) if T(n) <= f(n)*c for all n >= n_0

for some value of c and n_0

- $3n^2+7n - 200$ is order ….

  n^2?, n?, n^3?, log n?

- There are other related notations; but we will restrict to this one, with the assumption that we will always choose the f(n) which is as close to T(n) as possible.

Types of Complexities

- Runtime often depends on characteristics of input. eg: consider sorting a set of numbers.
- Best case: Under the best input, what is the time?
- Worst case: What is the worst runtime possible?
  - Required to bound run-time. eg: "It will be no worse than N^2; and generally N log N"

Types of Complexities

- Average case: Try over a random collection of inputs and find average run-time. Often difficult to estimate analytically. eg: all inputs are not equally likely in practice.
- How often can the worst case happen?
  - may not be very infrequent always!

Case Study

- Let us now take a problem and develop a few algorithms for it
- Objectives:
  - can there be different algorithms for a problem?
  - how much can they differ in performance?

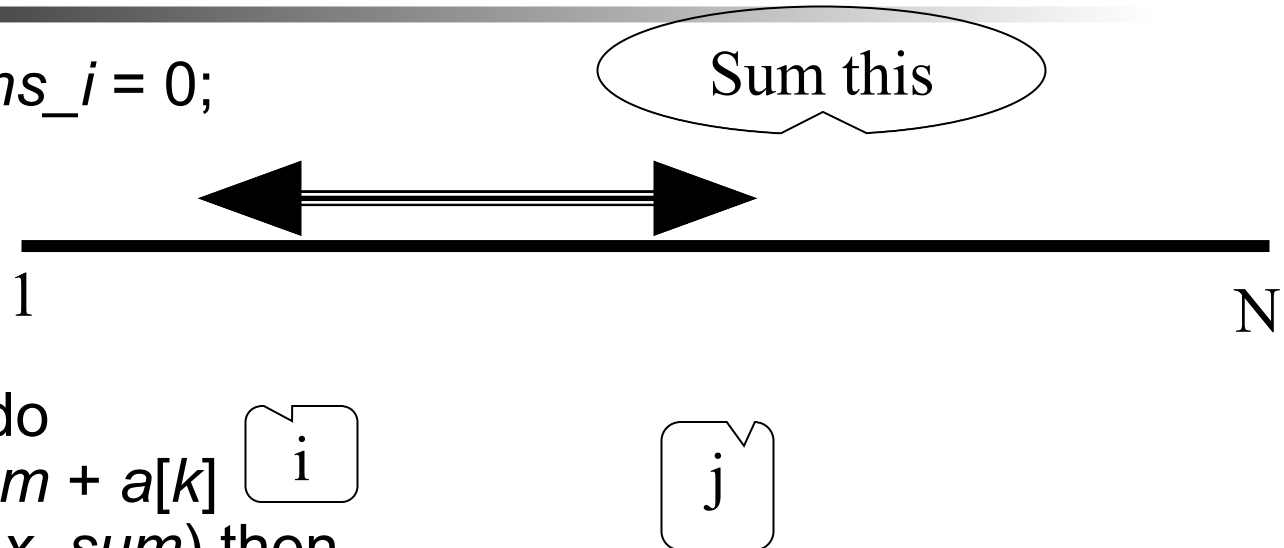
Maximal Subsequence Problem (MSP)

- Consider a series of numbers: e.g.
  -2, 11, -4, 13, -5, -2
- What is its maximal subsequence, i.e., a contiguous part of this series whose sum is maximum of all such subsequences?
- Total sum here = 11
- Subsequence <-2,11,-4,13> = 18
- Subsequence <11,-4,13> = 20 :**Maximum**

MSP – contd

- How does one solve this problem?
- Simplest: brute force approach.
- Enumerate every possible subsequence, find the sum and keep the largest so far.

Brute-force Solution (V1)



max_sum = 0; *ans_i* = 0;
ans_j = 0;
for *i* = 1 to *n*
    for *j* = *i* to *n*
        sum = 0;
        for *k* = *i* to *j* do
            sum = *sum* + *a*[*k*]
        if (*sum* > *max_sum*) then
            max_sum = *sum*; *ans_i* = *i*; *ans_j* = *j*;

Complexity?

Improved…(v2)

max_sum = 0; *ans_i* = 0; *ans_j* = 0;
for *i* = 1 to *n*
  sum = 0;
  for *j* = *i* to *n*
      sum = *sum* + *a*[*j*];
      if (*sum* > *max_sum*) then
          max_sum = *sum*; *ans_i* = *i*; *ans_j* = *j*;

Complexity?

Still Further…(v3)

```
max_sum = 0; ans_i = 0; ans_j = 0;
i = 1; sum = 0;
for j = 1 to n
    sum = sum + a[j];
    if (sum > max_sum) then
        max_sum = sum; ans_i = i; ans_j = j;
    else if sum < 0 then
        i = j+1; sum = 0;
```

Complexity?

MSP - Analysis

- Complexity of v1 is $O(N^3)$, v2 is $O(N^2)$ & v3 is $O(N)$.
- Does the complexity improvement mean anything in practice?

Timing Study

- 890 elements: v1 took 27 seconds, v2 0.1 second.
- 18000 elements: v3 took 0.1 sec, v2 took 45 seconds, v1 took over 3 hours!
- Thus we have three different algorithms for solving the same problem, with vastly differing performance.

Growth of Functions

N	10	50	100	1000
5N	50	250	500	5000
N log N	33	282	665	9966
N^2	100	2500	10000	1 million (7 digits)
N^3	1000	125000	1 million (7 digits)	1 billion (10 digits)
2^N	1024	16 digits	31 digits	302 digits
N!	7 digits	65 digits	161 digits	too large
N^N	11 digits	85 digits	201 digits	too large

Complexity Classes

- log N, N – algorithms most preferred.
- polynomial complexity – tolerable.
- exponential – expensive except for very small input sizes.
- But, many real-life problems, currently, have only exponential complexity algorithms.

Changing Complexity

- Apparently minor changes in problem can change complexity substantially.
- Given a graph:
  - Euler tour visiting all edges exactly once has a linear time algorithm.
  - Hamiltonian tour visiting all vertices once has only an exponential algorithm
- Given a set of jobs to be scheduled on a set of machines, find optimal schedule:
  - optimise average completion time: easy
  - optimise total completion time: exponential!

Some Problems

- Given a set of boolean expressions, is there a choice of truth values which satisfies all of them?
  - (AvBvC) and (A'vB') and (B'vC') = 1
- Given a set of weights s1,s2,s3,… each with a profit p1, p2, p3, etc, is it possible to obtain a profit M or more if I can carry only a total weight of W?
- These form part of an interesting class of problems – NP Complete Problems

NP-Complete Problems

- Non-deterministic Polynomial(NP): Given a proposed solution, verifying if that is a solution can be done in polynomial time.
- If one of these problems can be solved in polynomial time, it is possible to solve ALL of them in polynomial time.
- As of now, no one has found a polynomial solution; and no one has proved that there cannot be a polynomial solution.
- Considered to be a set of very hard problems.

Summary

- Order notation as a measure to compare algorithms for performance.
- Be careful while interpreting the order of complexity!
- Best/worst/average case complexities.
- Types of complexity functions and characteristics.
- Looking for algorithms with better time complexity is often worthwhile.