Lancaster University

SCC.121 ALGORITHMS AND COMPLEXITY
Introduction to Algorithms

Emma Wilson e.d.wilson1@lancaster.ac.uk

Algorithms and Complexity Plan

Lancaster University

Algorithms and Complexity

- Week 11: Introduction to algorithms and space/ time complexity
- Week 12: Time complexity using operation counting
- Week 13: Time complexity of searching algorithms
- Week 14/15: Asymptotic notation: Big-O, Big-Ω and Big-Θ
- Week 15: Time complexity of recursive algorithms
- Summer Term (2 weeks): Theoretical time complexity: Introduction to P, NP complete and NP hard problems

Today's Lecture

Lancaster University

Aim: To introduce the concept of algorithms and the space and time complexity of algorithms

Learning objectives:

- To know what an algorithm is, how one can be represented, and the importance of studying them
- To know what is meant by the time and space complexity of algorithms

Today's Lecture

Lancaster University

- What is an algorithm?
- Examples of algorithms
- Why study algorithms?

Today's Lecture

Lancaster University

- **What is an algorithm?**
- Examples of algorithms
- Why study algorithms?

What is an Algorithm?

Lancaster University

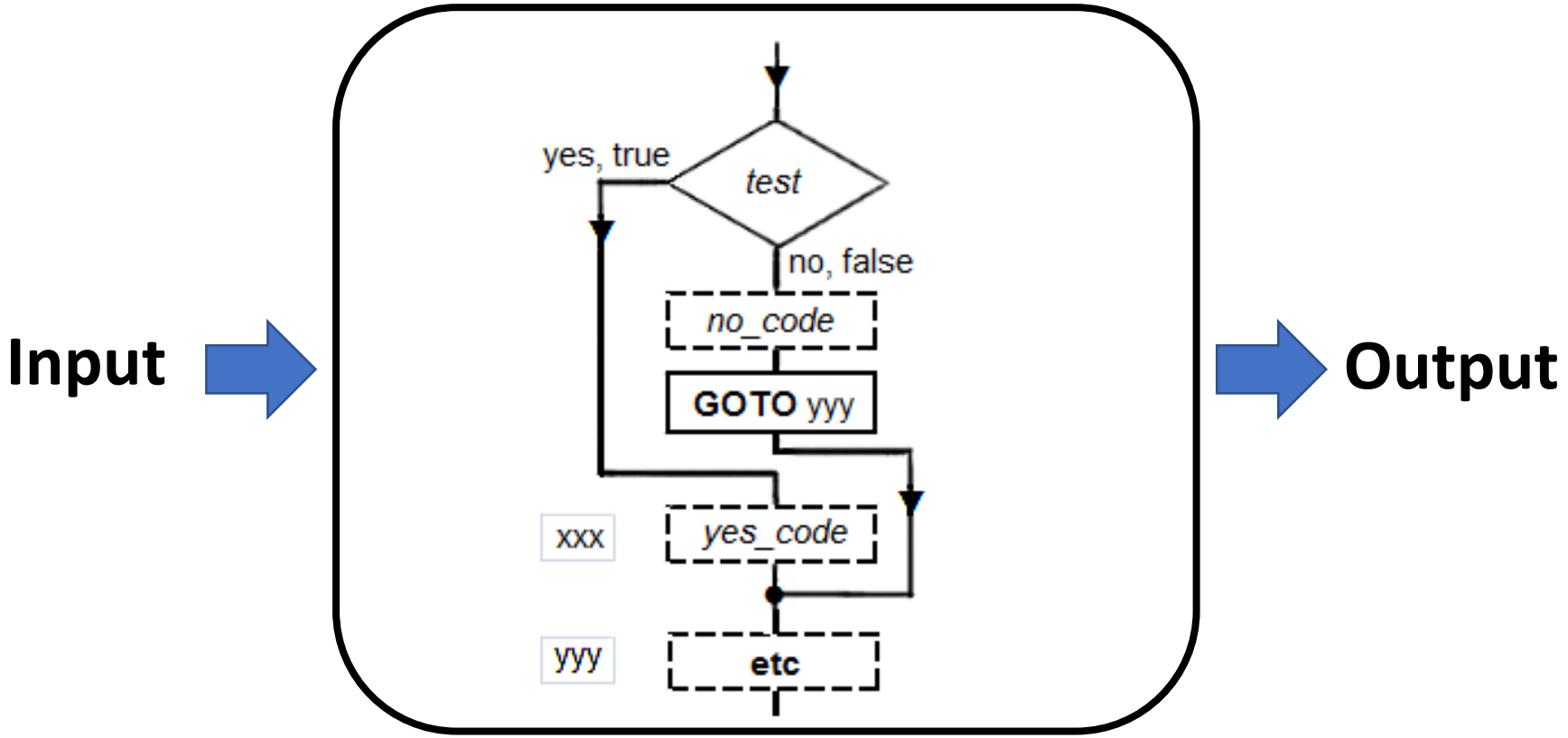
*"An **algorithm** is a set of **computational steps** that transform the **input** into the **output**…A tool for solving a **well-specified computational problem**….The algorithm describes a **specific computational procedure** for achieving that input/ output relationship." From* Cormen, Thomas H., et al. *Introduction to algorithms*. MIT press, 2009.

An algorithm is an effective method for solving a class of problems

- **Input**: Finite, represents instance of a problem
- **Effective Method**
  - Represented by a finite list of instructions
  - When executed on the input
    - Terminates
    - Produces the correct solution (the **output**)
  - Can be followed by a human
    - Doesn't need computer
  - Only thing needed to solve the instance

What is an Algorithm?

Lancaster University



Algorithm Representations

Lancaster University

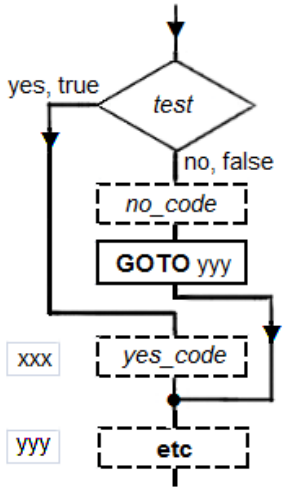
Computer code

```
int Multiply(int input1, int input2)
{
        int theProduct = 0;
        for(int i = 0; i < input2; i = i + 1)
                theProduct = theProduct + input1;
        return theProduct
}
```

Pseudocode

For(each node n_i in N)
  For (each node n_j in N, $n_j \neq n_i$)
    d_{ij} = Euclidean distance (n_i, n_j)
    apply expansion force $X_{ij} = -G/d_{ij}$
    If(edge (n_i, n_j) in E)
      apply string force $S_{ij} = s(d_{ij} - d_o)$

Flow charts

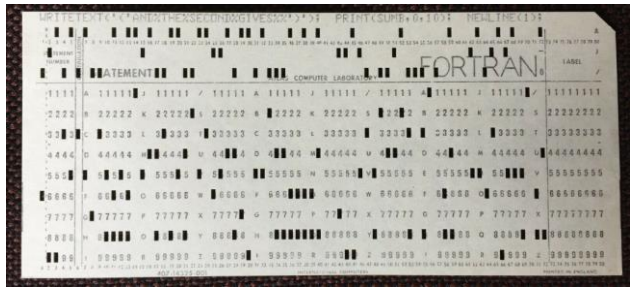


Text/sentences

Goal: Write a program that guesses an integer number that the user is thinking of

It should repeatedly ask the user if some number 'guess' is their hidden number and continue until they respond: "yes"

Punch cards

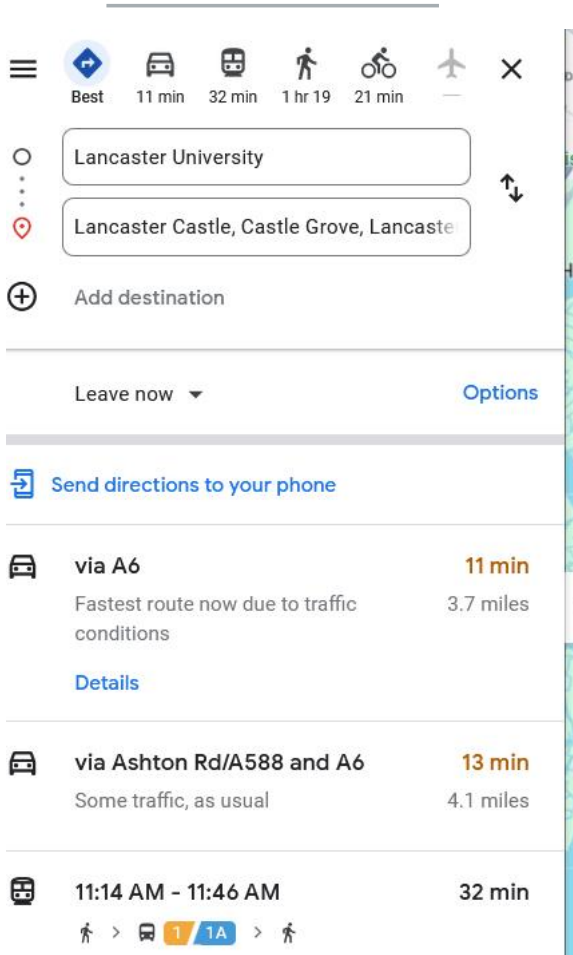


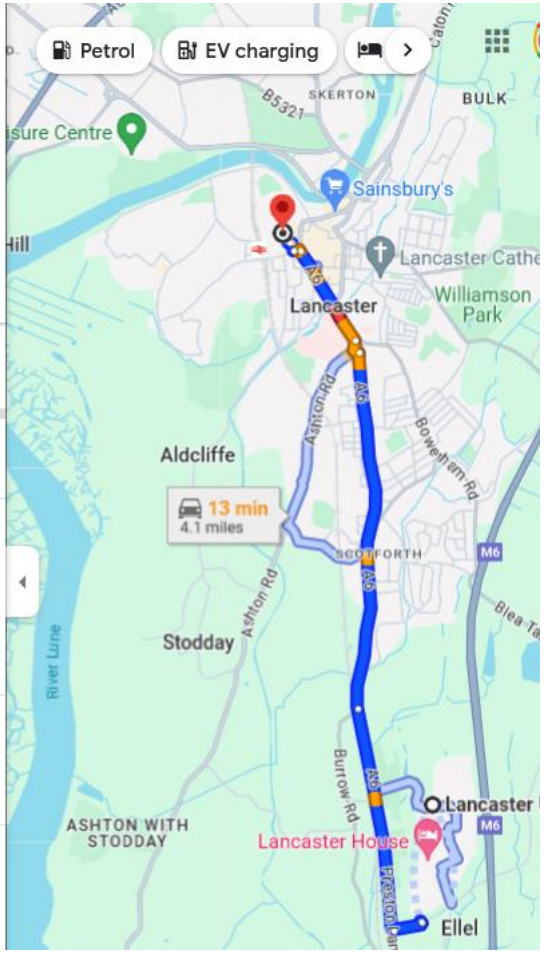
Today's Lecture

Lancaster University

- What is an algorithm?
- **Examples of algorithms**
- Why study algorithms?

Algorithm Example: Map Directions





Input: start and end location,

Output: Path suggestions.

Algorithm:

- Finds the quickest path(s) between the start and end inputs based on current traffic conditions
- Treat map as graph (more on graph algorithms in “Abstract Data Types”).
- Number of options, e.g. Dijkstra’s Algorithm

Algorithm Example: Automatic Vacuum Cleaner

Inputs: Sensor signals – wheel drop, obstacle detection, dirt sensors, docking station detection

Outputs: Path - motors drive wheels to move robot

Algorithm:

- Path planning algorithm, e.g. Random walk, spiral, wall following

Algorithm Example: Vending Coin Change

Lancaster University

Input: Change required (initial_value)

Output: Coins

Algorithm:

- Finds the coins to give as change
- Option: Greedy method, at each step chose the coin with largest possible value that does not exceed the amount of change left
  1) Set remval=initial_value
  2) Choose largest available coin that is less than remval
  3) Add coin to set of coins (output) and set remval:=remval-coin value
  4) Repeat 2 and 3 until remval = 0
- Greedy – best option available at the moment

Algorithm Example: Dataset searching, sorting, processing

Lancaster University

	A	B	C	D	E
1	College Enrollment 2007 - 2008				
2					
3	Student ID	Last Name	Initial	Age	Program
4	ST348-245	Walton	L.	21	Drafting
5	ST348-246	Wilson	R.	19	Science
6	ST348-247	Thompson	G.	18	Business
7	ST348-248	James	L.	23	Nursing
8	ST348-249	Peterson	M.	37	Science
9	ST348-250	Graham	J.	20	Arts
10	ST348-251	Smith	F.	26	Business
11	ST348-252	Nash	S.	22	Arts
12	ST348-253	Russell	W.	20	Nursing
13	ST348-254	Robitaille	L.	19	Drafting

Searching: **Input:** Array, search criteria, **Output:** Array indexes

- **Algorithm:** Many types of searching algorithms, e.g. Linear Search,

Sorting: **Input**: unsorted list, criteria, **Output:** sorted list

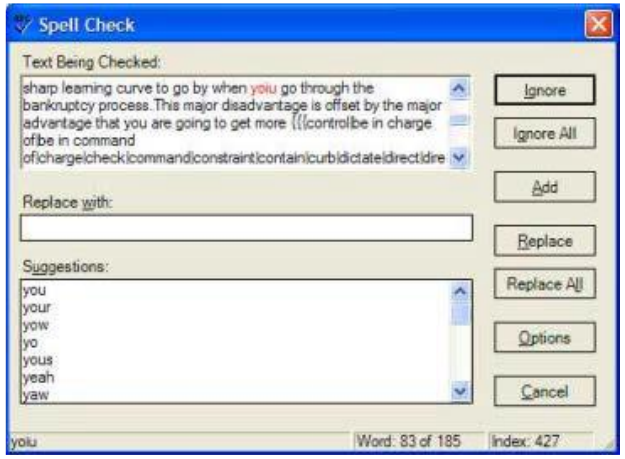
- **Algorithm:** Many options, e.g. quicksort, merge sort

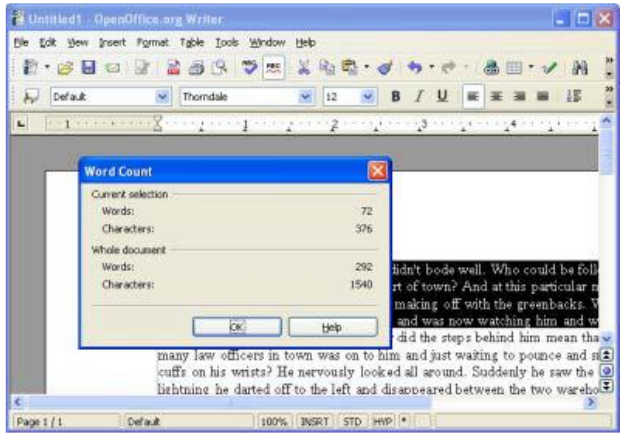
Calculations, e.g. average: **Input**: numeric array (arr) with length n, **Output:** Average value (av)

- **Algorithm:**

```
sum=0
for(i = 0 ; i < n ; i ++ )
        sum=sum+arr[i];
av=sum/n
```

Algorithm Example: Word Processing





Word processing: spell check, count number of words:

- Spell check: e.g. approximate string matching algorithms such as **Levenshtein distance**
- Word count – e.g count any string of “things” between two spaces as a word. Again different ways to do this, e.g. some may also read comma, or colon as a separator of words even if no space.

Algorithm Examples

Lancaster University

Many different approaches to solving same problems

- Inputs and outputs may be the same, but the steps taken to compute the output (e.g. the algorithm different).
- Algorithm may produce exactly the same numerical result – but the specific steps taken to get results (e.g. algorithm) differs.

Key types of Algorithms:

- **Sorting algorithms**: Used to sort the data in a particular format. Many types, e.g. quicksort, mergesort, etc.
- **Searching algorithms**: Used to find a value or record that user requests. Many types, e.g. Linear search, binary search, etc.
- **Graph Algorithms**: Used to find solutions to problems such as finding the shortest path between cities, e.g. breadth-first search, Dijkstra's algorithm, etc.

- **What makes one algorithm *'better'* than another?**
- Ways of comparing algorithms– time/ space complexity of algorithms

Today's Lecture

Lancaster University

- What is an algorithm?
- Examples of algorithms
- **Why study algorithms?**

Study of Algorithms

Lancaster University

- Algorithms require resources during computation to solve problem
  - **Space:** how much memory does it take to solve a problem
  - **Time:** how many steps (relative to input size) does it take to solve a problem
    - Performance analysis is actual running time on some computers (in seconds/minutes)
    - Theoretically: As opposed to actual time
- Other than performance, correctness, robustness, and simplicity are important

slido

Suppose computers were infinitely fast and memory was free. Would you have any reason to study algorithms?

ⓘ Start presenting to display the poll results on this slide.

Study of Algorithms

Lancaster University

- Other than performance, correctness, robustness, and simplicity are important
  - Still want to demonstrate solution terminates with correct solution
  - Want solution to be easy to implement (and within the bounds of good software engineering practice)
- In reality: Computers are fast, but not infinitely fast and memory is inexpensive, but not free
  - Computing time and space in memory bounded resources
  - Algorithms should be efficient in terms of space or time: Space and Time Complexity

Space Complexity

Lancaster University

- Space complexity = The amount of memory required by an algorithm to run to completion

- Some algorithms may be more efficient if data completely loaded into memory
  - Need to look also at system limitations
  - e.g. Classify 2GB of text in various categories [politics, tourism, sport, natural disasters, etc.] – can I afford to load the entire collection?

Space Complexity

Lancaster University

1. **Fixed part:** The size required to store certain data/variables, that is independent of the size of the problem:
   - e.g. name of the data collection
   - same size for classifying 2GB or 1MB of texts

2. **Variable part:** Space needed by variables, whose size is dependent on the size of the problem:
   - e.g. actual text
   - load 2GB of text VS. load 1MB of text

Space Complexity: Example 1

Lancaster University

```
#include<stdio.h>
int main()
{
    int a = 5, b = 5, c;
    c = a + b;
    printf("%d", c);
}
```

3 integer variables '*a*', '*b*', '*c*' are used

- The size of the integer data type is 2 or 4 bytes which depends on the compiler. Now, let's assume the size as 4 bytes.
- So, the total space occupied by the above given program is 4 * 3 = **12 bytes**.

Fixed part is 12 **Variable part is 0**

Space Complexity: Example 2

Lancaster University

```
#include <stdio.h>
int main()
{
    int n, i, sum = 0;
    scanf("%d", &n);
    int arr[n];
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
        sum = sum + arr[i];
    }
    printf("%d", sum);
}
```

What is the space complexity of this algorithm?

Fixed part:

- Integer variables such as *'n'*, *'i'* and *'sum'*.
  - Assuming 4 bytes for each variable, the total space occupied by *n, i* and *sum* is **12 bytes**

Space Complexity: Example 2

Lancaster University

```
#include <stdio.h>
int main()
{
    int n, i, sum = 0;
    scanf("%d", &n);
    int arr[n];
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
        sum = sum + arr[i];
    }
    printf("%d", sum);
}
```

What is the space complexity of this algorithm?

Variable Part:

- The array *'arr'* consists of '*n*' integer numbers.
  - So, the space occupied by the array is **4n bytes**

Space Complexity: Example 2

Lancaster University

```
#include <stdio.h>
int main()
{
    int n, i, sum = 0;
    scanf("%d", &n);
    int arr[n];
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
        sum = sum + arr[i];
    }
    printf("%d", sum);
}
```

What is the space complexity of this algorithm?

- The total space occupied by the program is:

12 + 4n bytes

Fixed part is 12

Variable part is 4n

Space Complexity: Question

Lancaster University

Pseudo-code

```
int freq[n];
int a[n];
int i
for(int i = 0; i<n; i++)
{
        freq[a[i]]++;
}
```

- What is the space complexity of this algorithm?
- Assume that the size of the integer data type is 4 bytes

slido

What is the space complexity of this algorithm (assume size of integer data type is 4 bytes)

ⓘ Start presenting to display the poll results on this slide.

Space Complexity: Question

Lancaster University

Pseudo-code

```
int freq[n];
int a[n];
int i
for(int i = 0; i<n; i++)
{
        freq[a[i]]++;
}
```

- What is the space complexity of this algorithm?
- Assume that the size of the integer data type is 4 bytes
- Fixed part: one integer (i) = 4 bytes
- Variable part: two arrays of length n (freq, a) = 2*n*4 bytes
- **Total: 4 + 8n bytes**

Space vs Time

Lancaster University

- An optimal algorithm requires less space in memory and also takes less time to compute the output. e.g. optimises both space and time complexity
- In general, it is not always possible to optimise both space and time complexity simultaneously.
- In practice - trade-off time vs space complexity, e.g.
  - Look up table vs recalculation
  - Compressed vs uncompressed data
  - Memoization (can be used to store the results of functional calls to reuse and reduce time complexity. However, increases space complexity)

Why is Runtime Important?

Lancaster University

- Want runtime to be as fast as possible
- Many possible algorithms for doing the same task, and some can be much slower than others
- Software requirements may specify certain performance time criteria

Running Time

Lancaster University

- The running time **depends on the size of the input**
  - Why?
- The running time **depends on the organization of the input**
  - Why?
- Generally, we seek **upper bounds (worst case) on the running time**,
  - Why?
- **Optimal operating temperature**
- **Number of processor cores**

Space vs Time

Lancaster University

- Undoubtedly, both time and space complexity are two important parameters for evaluating a solution.
- Nevertheless, with the current evolution in hardware technologies, in some circumstances space complexity is no longer as essential because almost all machines have enough memory.
- However, the **time complexity is still a crucial way** to evaluate algorithms and thus is the focus of this module (Week 12 to Week 15)

slido

Please download and install the Slido app on all computers you use

Audience Q&A

ⓘ Start presenting to display the audience questions on this slide.

Summary

Lancaster University

Today's lecture: introduced the concept of algorithms and the space and time complexity of algorithms

- An algorithm is an effective method for solving a class of problems
- Algorithms are used everywhere e.g. mobile phones, ATM, etc.
- Space and **Time** complexity
- The running time depends on the size of the input, the organization of the input, optimal operating temperature, number of processor cores
- **Next Lecture:** Introduction to operation counting
- **Any questions?**