

# Algorithms and Data Structures

Saul Pierotti

March 16, 2020

## Introduction

- Exam will be written, but you can do oral if you want
  - After lectures, around June 25
- An algorithm is a finite series of steps that solves a problem
  - In CS, an algorithm is a well defined computational procedure
- For a sorting algorithm, the input is a series of numbers and the output is an ordered series of numbers
- Algorithms are for humans, while a program is for a computer
- Algorithms are written in pseudocode, which follow specific conventions
- A problem can be solved by many algorithms
- An algorithm can be implemented in many different programs
- Properties of algorithms
  - Input for an algorithm can have 0 or more inputs
  - It always has 1 or more outputs
  - It should be clearly defined and unambiguous
  - It should terminate after a finite number of steps
  - All operations must be basic
    - \* They can be solved exactly and in finite time
- The correctness of an algorithm is difficult to prove
  - I would need to try all possible inputs (!)
  - Published algorithms have a mathematical proof
- Incorrect algorithms can produce a wrong output or not produce any for some instances
  - In some cases they are still useful, if I can control their error rate
- Efficiency is related to the ability of an algorithm to be executed with available resources
- Resources are time and memory
- Time is measured in running time, not CPU time
  - CPU time is dependent on CPU (!)
  - CPU time is number of instructions divided by number of instructions per unit time
  - Running time is the number of primitive operations to be performed in proportion to the input size
- The algorithm influences time much more than hardware, we do not focus on hardware (!)
- Running time of  $n^2$  is unacceptable for large inputs
- Using the right data structure is important for efficiency
- Decision trees are essential in CS
- An instance of a problem is a specific input for that problem
- In a while and for loop, the test is always executed once more than the body

## Math background

- Finite sums
  - $\sum_{k=1}^n a_k = a_1 + a_2 + \dots + a_n$
- Infinite sums

- $\sum_{k=1}^{\infty} a_k = a_1 + a_2 + \dots$
- Sums are linear
  - $\sum_{k=1}^n (ca_k + b_k) = c \sum_{k=1}^n a_k + \sum_{k=1}^n b_k$
- The arithmetic series
  - $\sum_{k=1}^n k = 1 + 2 + \dots + n = \frac{n(n+1)}{2}$
- The quadratic arithmetic series
  - $\sum_{k=1}^n k^2 = 1 + 4 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$
- The cubic arithmetic series
  - $\sum_{k=1}^n k^3 = 1 + 8 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$
- The geometric series
  - $\sum_{k=1}^n x^k = 1 + x + x^2 \dots + x^n = \frac{x^{n+1}-1}{x-1}$
  - When  $x$  is less than 1
    - \*  $\sum_{k=1}^{\infty} x^k = \frac{1}{1-x}$
    - \*  $\sum_{k=1}^{\infty} kx^k = \frac{x}{(1-x)^2}$
- Other formulas
  - $\sum_{k=1}^n \log k \approx n \log n$
  - $\sum_{k=1}^n k^p \approx \frac{n^{p+1}}{p+1}$
- A set is a non-ordered and non repetitive collection of elements
- Sets can be infinite
- It is described as  $S = \dots$
- 2 sets are equal if they contain the same elements
- The cardinality of a set  $|S|$  is the number of elements it contains
- If  $A$  contains all the elements contained in  $B$  and also other elements, then  $B$  is a proper subset of  $A$
- The power set  $P(S)$  is the set of all subset of  $S$ , including the empty set and  $S$  itself
  - $|P(S)| = 2^{|S|}$
- The cartesian product of 2 sets is a set containing all possible pairs of elements
- The cartesian product of  $n$  sets is a set of  $n$ -tuples
- A tree has only one way to go from one node to the other
- A graph can have cycles
- A forest is made of many trees
- Any non-empty tree with  $n$  nodes has  $n-1$  edges
  - If this is not true, we don't have a tree
- A tree is rooted if one of its nodes is distinguished as root
  - It can be defined recursively such that every non-root node of a rooted tree is itself the root of a subtree
- Tree terminology is similar to that of ancestry trees
- The depth of a node is its distance from the root (number of edges)
- The height of a node is the lenght of the longest path to a leaf
- A binary tree is an order tree with 2 subtrees wich are themselves binary

## Pseudocode

- We start counters from 1 since it is easier to understand
- Bold words are reserved words like **return**
- Variables are always local to the current procedure
- We can have loops like **while** and **for**

for  $i=0$  to/downto  $i=4$  (by 3)

  <statement>

- We have if statements

if <condition>

  <statement>

- Comments are rendered with `//`
- No colon/semicolon at the end of lines
- Use of indented blocks
- Differentiate conditional expressions and assignments (!)
- A slice of an array is indicated as `A[3..5]`
- Attributes of objects are indicated as `object.attribute`
  - The length of an array can be indicated as `A.length`

## Sorting

- Sorting is an intermediate step in many tasks in CS
- There are many sorting algorithms

## Insertion sort

- It is like arranging cards in order in your hand by picking one at a time
- I take 1 unsorted object at a time and I insert it in the correct position in the sorted array
  - I compare with all the objects in the sorted array, until I find the right position
- I start from the first element of the array and I don't do anything
- I take the second element, and if it is smaller than the first I swap them
- I take the third, and if it is smaller than the second I compare it with the first and I swap in the right position
- I continue like this for all the elements

### Pseudocode

```

INSERTION-SORT(A)
  for j = 2 to A.length
    key = A[j]
    i = j-1
    while i > 0 and A[i] > key
      A[i+1] = A[i]
    A[i+1] = key

```

### Running time

- Nearly sorted numbers can be sorted much faster with insertion sort
- The input size is the length of the array
  - $n = A.length$
- The initial **for** test is executed  $n$  times
  - It is  $n$ , not  $n-1$  because even when it is false we still have to check ones (!)
  - The body of the **for** is executed  $n-1$  times
- The assignment of `key` is therefore executed  $n-1$  times
- The **while** test is executed  $\sum_{j=2}^n t_j$  times
  - The body of the **while** is executed  $\sum_{j=2}^n t_j - 1$
  - There are 2 assignments on the while body
- The final assignment after the **while** inside the **for** is executed  $n-1$  times

### Best case

- The array is already sorted
- I never enter the while, but I do completely the for
- This means that  $t_j$  is 1, I only do the test
- The time is linear

### Worst case

- The array is in reverse sorted order
- The time is quadratic

#### Average case

- It is really difficult to do, we prefer to focus on the worst case

#### Evaluation

- For almost sorted sequences its running time is almost linear
- Can be online
  - It can sort sequences as they arrive
- In the worst and average cases it is quadratic
  - Quadratic is really bad (!)

## Merge sort

- It is a divide and conquer algorithm
  - Divide a problem in subproblems of smaller size
  - Solve the subproblems recursively (conquer)
  - Combine the solution to solve the original problem
- The complicated part is the merging process
- It runs always as  $n \cdot \log(n)$ , there is not worst or best case
- It requires a lot of memory to store all the sub-arrays
- It is worse than insertion sort in the best case, but better in most cases
- It cannot work online (!)

#### Idea

- I want to sort the array A
- I split the array using the indices p,q,r such that  $p \leq q < r$
- I want to produce a single sorted subarray
- I call initially on A with p=1 and r=A.length
- The index q is the one that best splits the array in 2
- For merging I always have sorted arrays to merge
  - The first element of each array is guaranteed to be the smallest one of the entire array
  - I compare the first element of the 2 arrays to be merged, and I put the smallest in the output array
  - I repeat until one of the arrays is empty
  - I finish by putting what remains of the other array in the output
  - I put an imaginary infinite at the end of any array
    - \* This is so that when I finish the elements of an array, whatever remains in the other is smaller and so it is inserted in the output

#### Pseudocode

```

MERGESORT(A,p,r)
  if p < r
    q = (p+r)/2
    MERGESORT(A,p,q)
    MERGESORT(A,q+1,r)
    MERGE(A,p,q,r)

```

```

MERGE(A,p,q,r)
  n1 = q - p + 1
  n2 = r - q
  for i = 1 to n1
    L[i] = A[p+i-1]

```

```

for j = 1 to n2
    R[i] = A[q+j]
L[n1+1] = \infty
R[n2+1] = \infty
i = 1
j = 1
for k = p to r
    if L[i] <= R[j]
        A[k] = L[i]
        i += 1
    else A[k] = R[j]
        j += 1

```

MERGESORT(A,1,A.lenght)

### Running time

- MERGE
  - Copying the elements into the subarrays takes  $\Theta(n)$
  - Adding elements to the final array takes  $n$  iterations of that themselves take constant time  
\*  $\Theta(n)$
  - In total, the merging takes  $\Theta(n)$
- MERGE-SORT
  - Let  $T(n)$  be the unknown running time of MERGE-SORT
  - Calculating  $q$ :  $\Theta(1)$
  - Solve recursively 2 subproblems of size  $n/2$ :  $2T(n/2)$
  - Call to MERGE:  $\Theta(n)$
  - So,  $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(1) + \Theta(n) & \text{if } n > 1 \end{cases}$
  - The recursive equation can be solved and we find that  $T(n) = \Theta(n \log n)$

## Limiting behaviour of functions

- There are different notations to define the behaviour of functions
- The  $\Theta$  (theta) notation signifies asymptotic equality
  - Formally, for a function  $f(n)$  having a certain  $\Theta$  notation there are 2 constant that multiplied for the  $\Theta$  function are constantly greater or smaller than  $f(n)$  for  $n > n_0$   
\* This is defined as a tight bound
  - If I say that  $f(n) = \Theta(g(n))$  I mean that  $f(n)$  belongs to the family of functions with order of growth  $g(n)$
- The  $O$  (big-O) notation indicates an upper bound for the asymptotic behaviour
  - The formal definition is similar to that of  $\Theta$ , but instead of a tight bound I only search for an upper bound  
\* I only want a constant, not 2 (!)
- The  $o$  (small-o) notation indicates a lower bound for the function