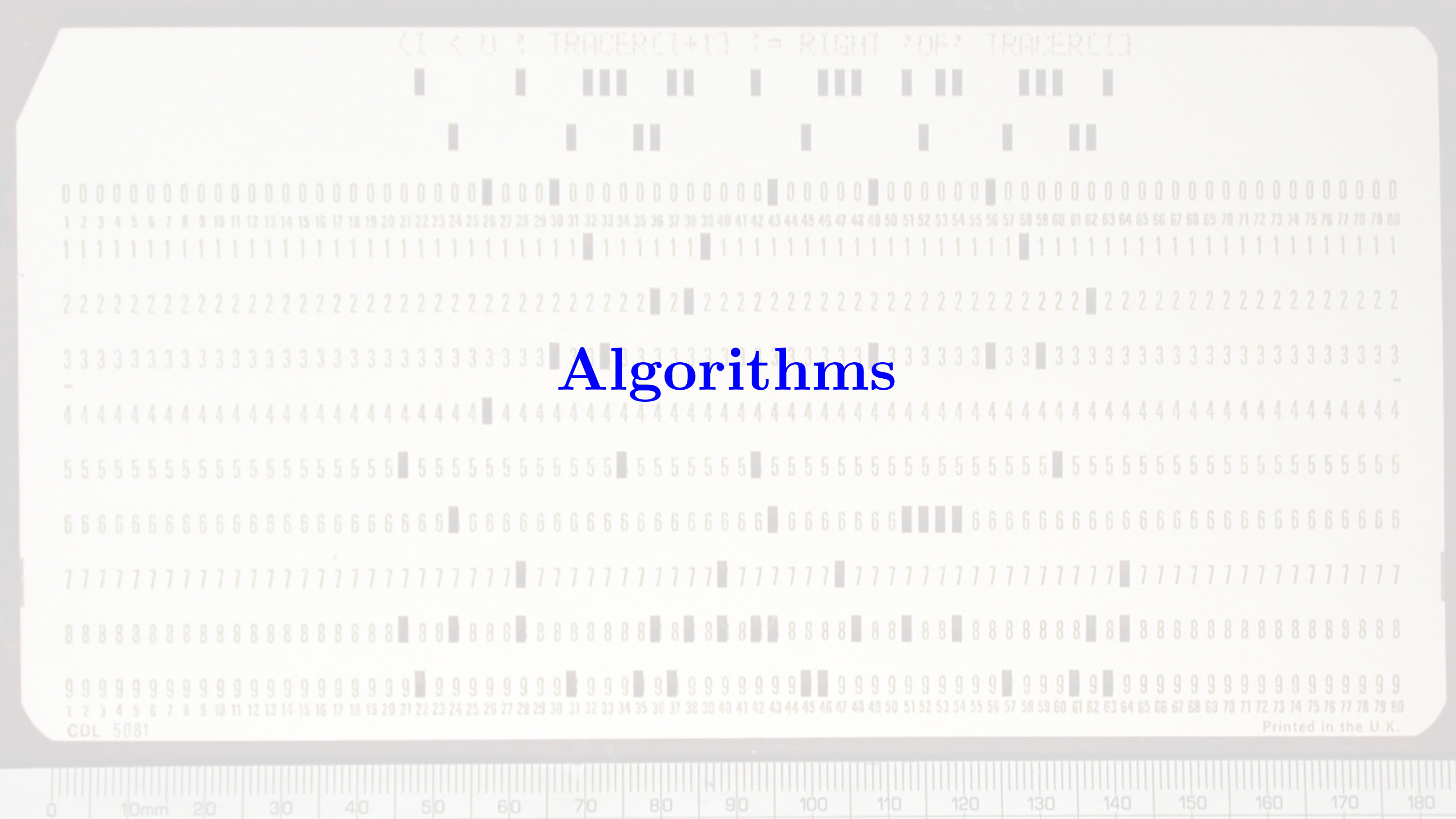
Algorithms



Reading

Mandatory.

Section 2.1-2.4 of the textbook.

Chapter 2 Solved Exerises.

Solve exercises 1,3,6 of Chapter 2.

Suggested.

Sections 2.1-2.5 of Roughgarden's video lectures.
http://www.algorithmsilluminated.org/

Notes on Asymptotic Notation (on Brightspace).

All exercises of Chapter 2.

Algorithms

What is an efficient algorithm?

One that uses less resources. Mainly time and space.

Most of the time, we will focus on the running time.

In particular, running time as a function of input size.

Given two algorithms, how do we know which is better?

Option 1: Find out empirically.

Option 2: Find out analytically.

Empirical Approach

- We need to implement both algorithms
- The results depend on quality of implementation and platform.
- Can only be tried on a limited number of inputs.
- Does not tell us how to design efficient algorithms.

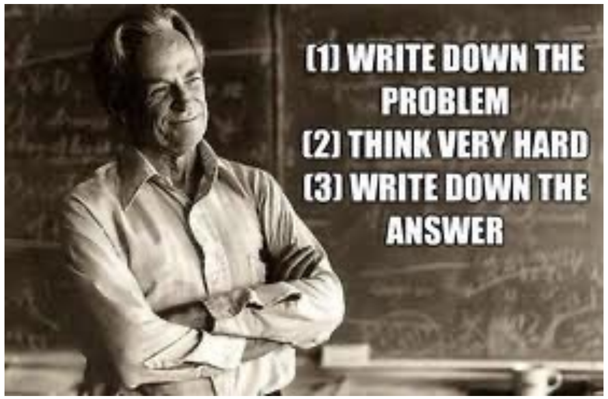
Analytical Approach

The goal is to ignore superficial differences between programs and concentrate on large components of running time.

Step 1: Start with a high level description of the algorithm

Step 2: Identify "basic operations" in the algorithm

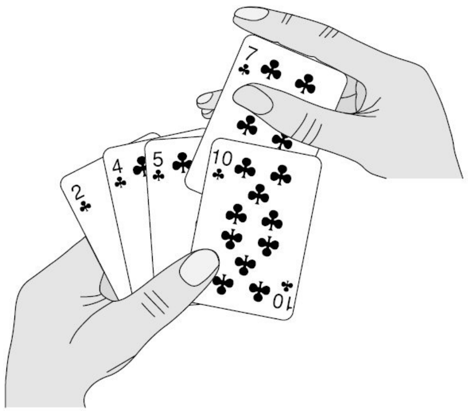
Step 3: Count the number of basic operations required for an input of size n by **staring** at the algorithm.



Insertion Sort

Input: Sequence of numbers

Output: Non-decreasing permutation of the input

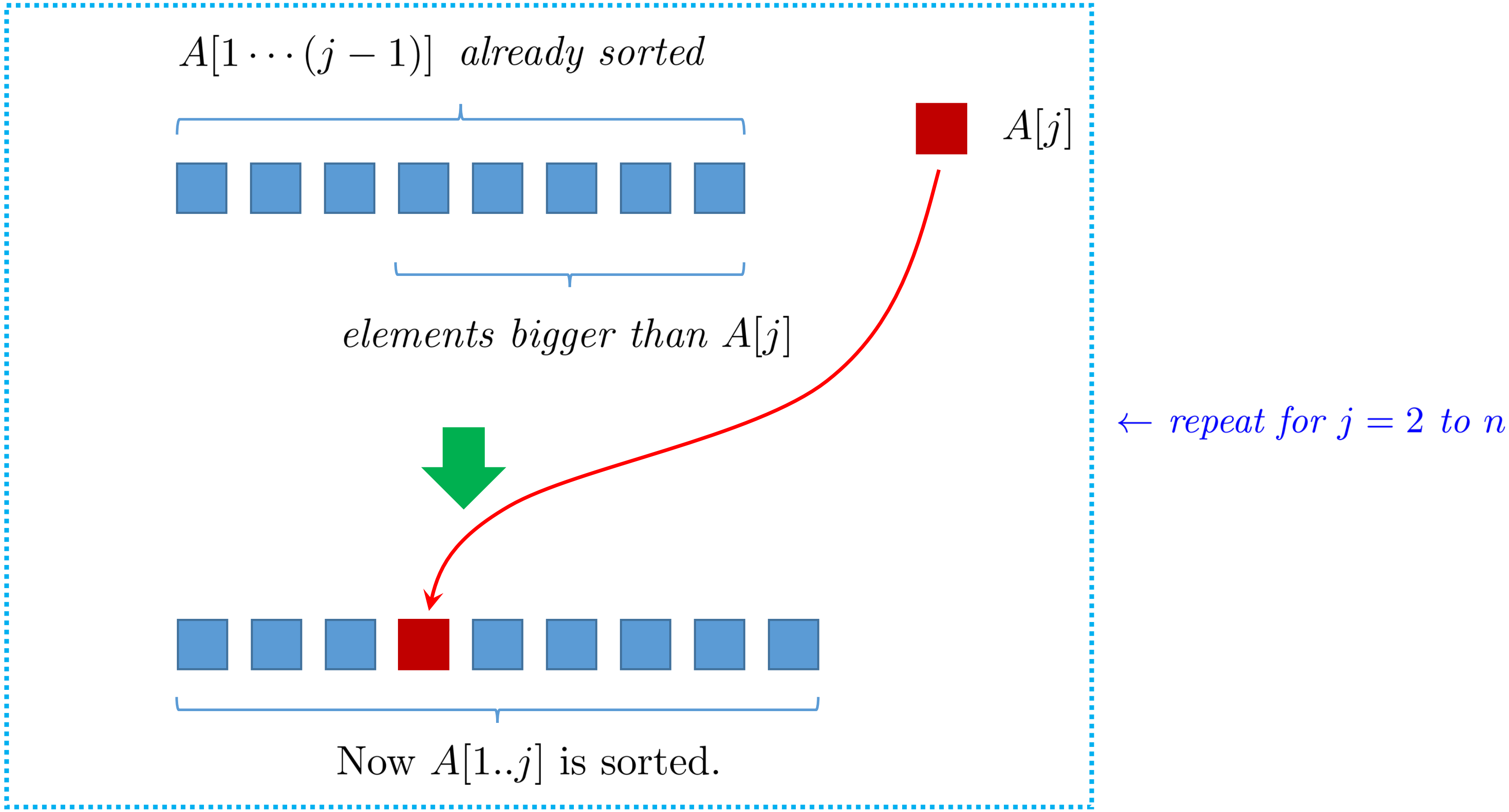


Example input: 6 5 3 1 8 7 2 4.

Can you decipher the algorithm?

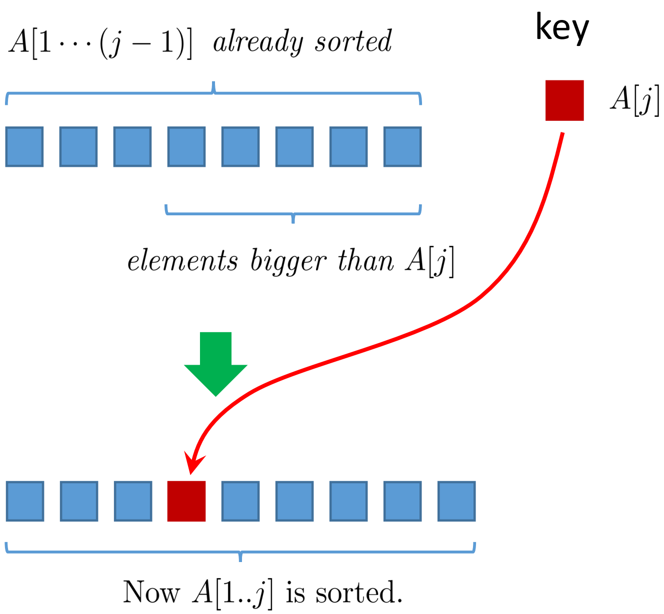
Insertion Sort

Idea. We insert the elements one by one from left to right and keep the set of inserted elements sorted.



Insertion Sort

repeat for $j = 2$ ***to*** n**:**

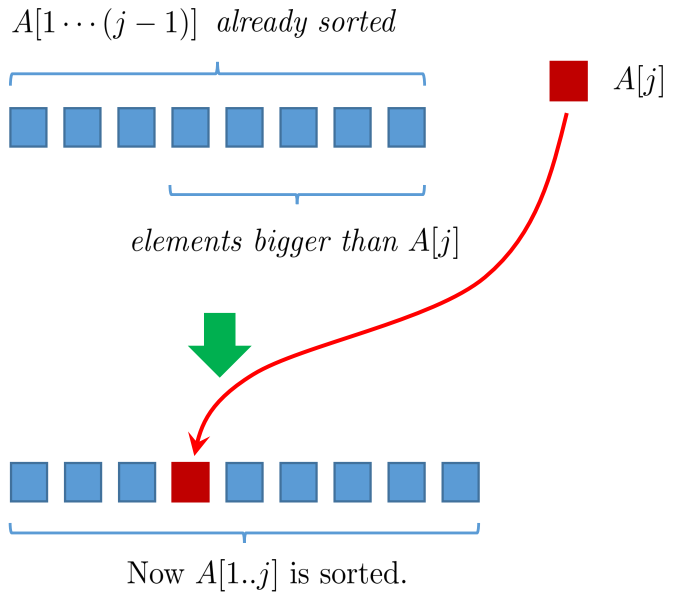



```
for i = 0 to n
    key = A[i]
    j = i - 1
    while j >= 0 and A[j] > key
        A[j + 1] = A[j]
        j = j - 1
    end while
A[j + 1] = key
end for
```

What is the number of operations as a function of n?

Insertion Sort

repeat for $j = 2$ ***to*** n**:**



What is the number of operations as a function of n?

(A) $\propto 1$

(B) $\propto n$

(C) $\propto n^2$

(D) It depends.

Best Case: $\propto n$ *Worst Case:* $\propto n^2$

Which case to use?

The worst case running time is usually a good indicator of the efficiency of an algorithm. It also gives an upper bound.

The best case running time is almost never a good indicator.

An average case running time is often as bad as worst case, and typically harder to compute.

We will stick with **Worst Case Analysis**.

Asymptotic Analysis

Idea: concentrate on how the running time grows as the input size increases.

What happens when input doubles in size to the following running times?

$2n$	Running time doubles when input size doubles
$10n$	Running time doubles when input size doubles
n^2	Running time quadruples when input size doubles

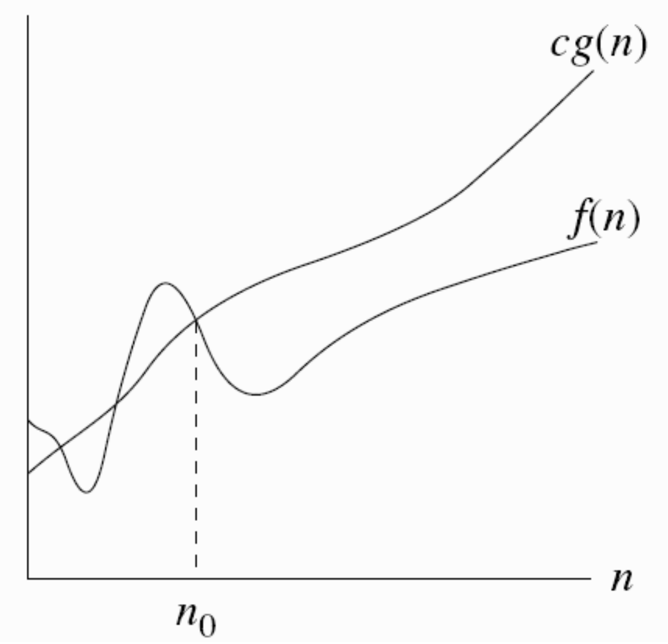
In some sense, $2n \equiv 10n$.

Similarly, $n^2 \equiv 5n^2$.

Asymptotic Analysis: Big-Oh Notation

$f(n)$ is $O(g(n))$ if $\exists$ constants c and n_0 s.t. $f(n) \leq c \cdot g(n)$ for $n \geq n_0$

We are assuming that $f(n)$ and $g(n)$ are positive functions.



Abuse of notation:

When $f(n)$ is $O(g(n))$, we will often write $f(n) = O(g(n))$.

This is incorrect but frequently used notation.

Asymptotic Analysis: Big-Oh Notation

Examples:

$3n + 7$ is $O(n)$

$5n^2 + 4n + 9$ is $O(n^2)$

n^2 is **not** $O(n)$

General Rule: Ignore lower order terms and constants.

$0.1n^3 \log n + 200n^2 \log\log n + 4.2n\sqrt{n} + 9 \quad$ is $O(n^3 \log n)$

$2n$ is $O(n^9)$ but we usually try and give the best bound.

Practice

HOME

Is $n^2 + 10 = O(n^2)$?

Is $10n = O(9n)$?

Is $20 = O(1)$?

Is $1 = O(20)$?

Is $3^n = O(2^n)$?

Is $n^2 = O(n \log n)$

Is $n! = O(2^n)$

Is $n\sqrt{n} = O(n \log^2 n)$?

Exercise. Prove that for non-negative functions $f(n)$ and $g(n)$,

- $\max(f(n), g(n)) = O(f(n) + g(n))$ and
- $f(n) + g(n) = O(\max(f(n), g(n)))$

Practice

Let $T(n) = \frac{1}{2}n^2 + 3n$. Which of the following is true?

1. T(n)=O(n)
2. T(n)= $O(nlog(n))$
3. T(n)=O(n^2)
4. T(n)=O(n^3)

Which of these options is O(2^n)

1. 2^{10+n}
2. 2^{10n}
3. 3^{10+n}

Practice

What is the running time of the following functions in terms of n?

```python
def f(n):
  s = 0
  for i in range(n):
    s += i*i
  return s
```

$O(n)$

```python
def f(n):
  s = 0
  for i in range(n):
    for j in range(i+1):
      s += i*j
  return s
```

$O(n^2)$

```python
def f(n):
  if n<=1:
     return 1
  else:
     return n*f(n-1)
```

$O(n)$

Practice

What is the running time of the following functions in terms of n?

```
def f(n):
  if n<=1: return n
  return 2*f(n//2)
```

$O(\log n)$

```
def f(n):
  if n<=1: return n
  return f(n//2) + f(n//2)
```

$//$: *integer division*

$O(n)$

```
def f(n):
  if n<=1: return n
  return f(n-1) + f(n-1)
```

$O(2^n)$

Solve these as a homework

Read the substitution method in Reccurence relations documents on brightspace