Design and Analysis of Algorithms

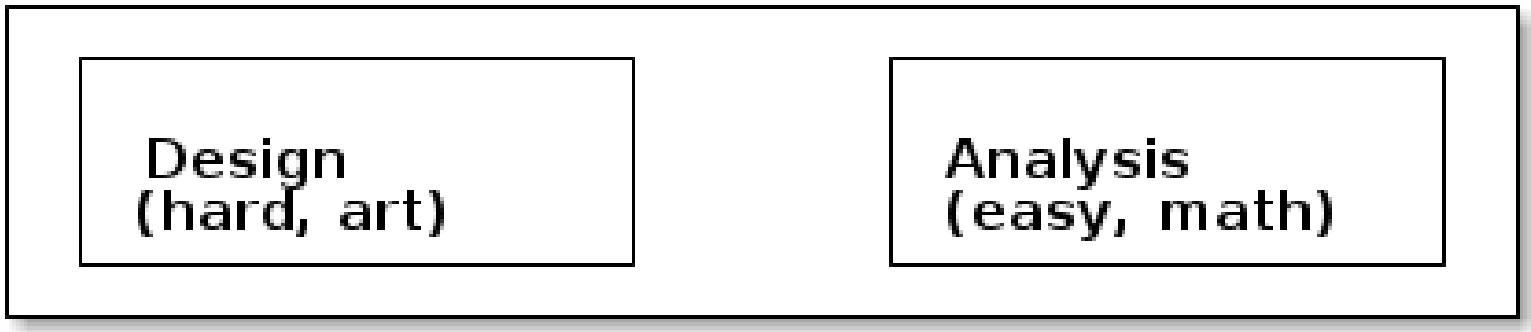
Serge Kruk

May 4, 2022

Synonyms

- Introduction to algorithms
- Design and analysis of algorithms
- The mathematics of Computer Science
- Complexity theory

Each with a different focus.



What are we talking about?

Given a sequence of numbers, return a sequence with the same numbers ordered from smaller to larger.

Note that the problem or the solution needs to specify

- Input

What are we talking about?

Given a sequence of numbers, return a sequence with the same numbers ordered from smaller to larger.

Note that the problem or the solution needs to specify

- Input
- Output

What are we talking about?

Given a sequence of numbers, return a sequence with the same numbers ordered from smaller to larger.

Note that the problem or the solution needs to specify

- Input
- Output
- Process

What do we want from an algorithm?

- Correct

What do we want from an algorithm?

- Correct
  - By testing exhaustively

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?
  - Is it the best possible process?

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?
  - Is it the best possible process?
- Implementable

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?
  - Is it the best possible process?
- Implementable
  - Given language X?

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?
  - Is it the best possible process?
- Implementable
  - Given language X?
  - Given choice of language?

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?
  - Is it the best possible process?
- Implementable
  - Given language X?
  - Given choice of language?
  - By a genius programmer?

What do we want from an algorithm?

- Correct
  - By testing exhaustively
  - By proving correctness
- Efficient
  - With respect to time (which), space (which)?
  - Best case, worst case, average case?
  - Is it the best possible process?
- Implementable
  - Given language X?
  - Given choice of language?
  - By a genius programmer?
  - By a team of drones?

What do we want from an algorithm?

We will focus on

- Correctness
- Run time efficiency as measured via the RAM

And ignore

- Almost everything else

Roughly the boundary between Computer Science and Software Engineering.

An algorithm (Typical exam questions)

```
def bs(a=[]):
    n = len(a)
    for i in range(n-1):
        for j in range(i+1,n):
            if a[i] > a[j]:
                a[i],a[j] = a[j],a[i]
    return a
```

- What does it do?
- Is this algorithm correct?
- How fast is it?

Same algorithm ?

```
bs(int a[], int n) {
    int i,j,t;
    for (i=0; i<n-1; i++){
        for (j=i+1; j<n; j++){
            if (a[i] > a[j]) {
                t = a[i]; a[i]=a[j]; a[j]=t;
            }
        }
    }
    return a
}
```

- Just as correct (or incorrect)?
- Just as fast?

What about this one?

```
(defun bs (sequence)
  (let ((n (length sequence)))
    (loop :for i :from 0 :to (- n 2) do
          (loop :for j :from (1+ i) :to (1- n) do
                (unless (<(elt sequence i) (elt sequence j))
                  (rotatef (elt sequence i) (elt sequence j)))))
    sequence))
```

BS

Three algorithms?

- No, just one!
  - Same data structure (well, almost)
  - Same process

In this class

- Our view of algorithms is (mostly) independent of language.

Is the algorithm correct? (Discuss)

```
def bs(a=[]):
    n = len(a)
    for i in range(n-1):
        for j in range(i+1,n):
            if a[i] > a[j]:
                a[i],a[j] = a[j],a[i]
    return a
```

Proof by testing

```
print(bs([1,3,2]))
```

['a', 'b', 'c']
Are you convinced?

Testing

Testing shows the presence, not the absence of bugs.
– Dijkstra (1969)

Edsger W. Dijkstra

https://en.wikiquote.org/wiki/Edsger_W._Dijkstra

Is the algorithm correct? (Discuss)

```
def bs(a=[]):
    n = len(a)
    for i in range(n-1):
        for j in range(i+1,n):
            if a[i] > a[j]:
                a[i],a[j] = a[j],a[i]
    return a
```

Proof by argument

- The outer loop isolates as `a[i]` element from first to one before last.

Proof by argument

- The outer loop isolates as `a[i]` element from first to one before last.
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.

Proof by argument

- The outer loop isolates as `a[i]` element from first to one before last.
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.
- Therefore at the end of the inner loop, element `a[i]` is the smallest of all elements from `i` to the end.

Proof by argument

- The outer loop isolates as `a[i]` element from first to one before last.
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.
- Therefore at the end of the inner loop, element `a[i]` is the smallest of all elements from `i` to the end.
- Therefore at the end of the outer loop every element is smaller (or equal) to all elements following it and the array is sorted

Proof by argument

- The outer loop isolates as `a[i]` element from first to one before last.
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.
- Therefore at the end of the inner loop, element `a[i]` is the smallest of all elements from `i` to the end.
- Therefore at the end of the outer loop every element is smaller (or equal) to all elements following it and the array is sorted
- Key: With `i` having value `k`, the elements `0` to `k-1` of the array are the `k` smallest elements of the array and they are sorted.

Proof of correctness

Our goal will be to develop tools to make the proofs simple and convincing.

Problem for the class

Write a function which, given an array of `n` numbers, returns the smallest `k` numbers.

- What is the input?
- What is the output?
- Write down the process.

Smallest k numbers of an array (Solution 0)

- What is the input?
  - The array (let's call it `a`)
  - The number of elements to extract (let's call that `k`)
- What is the output?
  - An array of length `k` with the smallest elements of array `a`

Smallest k numbers of an array (Solution 0)

Put these lines in order to construct a solution.

```
a[i],a[j] = a[j],a[i]
def kmallest0(a, k):
for i in range(k):
for j in range(i+1,n):
if a[i] > a[j]:
n = len(a)
return a[0:k]
```

Smallest k numbers of an array (Solution 0)

In the proper order:

```
def ksmallest0(a, k):
  n = len(a)
  for i in range(k):
    for j in range(i+1,n):
      if a[i] > a[j]:
        a[i],a[j] = a[j],a[i]
  return a[0:k]
```

Testing

```
print(ksmallest0([1,3,2],3))
```

```
[1, 2, 3]
```

Correctness

- Under the assumption that the array has at least `k` elements (Pre-condition)

Correctness

- Under the assumption that the array has at least `k` elements (Pre-condition)
- The outer loop isolates as `a[i]` the first `k` elements of the array

Correctness

- Under the assumption that the array has at least `k` elements (Pre-condition)
- The outer loop isolates as `a[i]` the first `k` elements of the array
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.

Correctness

- Under the assumption that the array has at least `k` elements (Pre-condition)
- The outer loop isolates as `a[i]` the first `k` elements of the array
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.
- Therefore at the end of the inner loop, element `a[i]` is the smallest of all elements from `i` to the end.

Correctness

- Under the assumption that the array has at least `k` elements (Pre-condition)
- The outer loop isolates as `a[i]` the first `k` elements of the array
- The inner loop compares the isolated element to each following and swaps the smaller into the isolated element's position.
- Therefore at the end of the inner loop, element `a[i]` is the smallest of all elements from `i` to the end.
- Therefore at the end of the outer loop each of the first `k` elements is smaller (or equal) to all elements following it and we return a copy of these `k` elements

Efficiency

- What shall we count? The number of comparisons seems good.
- The outer loop executes `k` times.
- The inner loop executes `n-1`, `n-2`, ..., `n-k` times
- So, at most `n` times.
- Therefore at most `nk` comparisons.

Smallest k numbers of an array (Solution 1)

Can you think of another solution?

Smallest k numbers of an array (Solution 1)

```
def ksmallest1(a, k):
    return bs(a)[0:k]
```

What can we say about correctness and efficiency?

Correctness

- If `bs` is correct
- and if we assume that it will never be called with `k` larger than array length
- then yes, this version is correct.

Efficiency?

- In terms of programmer's time, it must be ideal (one-liner!)
- `ksmallest` is as runtime efficient as our `bs` sorting routine.
  - Plus time to copy `k` elements

Contrast runtime of both algorithms

- Algorithm `ksmallest0` runs in time proportional to kn
- Algorithm `ksmallest1` runs in time proportional to n^2

Conclusion

We have a clear and quantitative comparison of two algorithms!

Can we ask anything else?

- Can we do better than kn?

Can we ask anything else?

- Can we do better than kn?
  - Yes, using a priority heap (more on that later).

Can we ask anything else?

- Can we do better than kn?
  - Yes, using a priority heap (more on that later).
- What is a theoretical minimal time constraint?

Can we ask anything else?

- Can we do better than kn?
  - Yes, using a priority heap (more on that later).
- What is a theoretical minimal time constraint?
  - We need to read the whole array.

Can we ask anything else?

- Can we do better than kn?
  - Yes, using a priority heap (more on that later).
- What is a theoretical minimal time constraint?
  - We need to read the whole array.
  - So minimal time has to be proportional to length of array.

Can we ask anything else?

- Can we do better than kn?
  - Yes, using a priority heap (more on that later).
- What is a theoretical minimal time constraint?
  - We need to read the whole array.
  - So minimal time has to be proportional to length of array.
  - Is this achievable?

The main goals of this course

- Develop algorithms to solve problems.

The main goals of this course

- Develop algorithms to solve problems.
  - Not find the right library or glue the right calls together.

The main goals of this course

- Develop algorithms to solve problems.
  - Not find the right library or glue the right calls together.
- Prove the algorithms correct.

The main goals of this course

- Develop algorithms to solve problems.
  - Not find the right library or glue the right calls together.
- Prove the algorithms correct.
- Compute asymptotic runtime.

The main goals of this course

- Develop algorithms to solve problems.
  - Not find the right library or glue the right calls together.
- Prove the algorithms correct.
- Compute asymptotic runtime.
- Contrast different algorithms solving the same problem.

The main goals of this course

- Develop algorithms to solve problems.
  - Not find the right library or glue the right calls together.
- Prove the algorithms correct.
- Compute asymptotic runtime.
- Contrast different algorithms solving the same problem.
- Learn the major design approaches.

The main goals of this course

- Develop algorithms to solve problems.
  - Not find the right library or glue the right calls together.
- Prove the algorithms correct.
- Compute asymptotic runtime.
- Contrast different algorithms solving the same problem.
- Learn the major design approaches.
- Learn the well-known algorithms.

Class project (for fun)

- This is NOT the object of the course.

Micro optimization

All current processors have very fast instruction caches. But, to take full advantage of those caches, code must limit jumps. Write the following three exercises without any loops or if statements.

- Determine the absolute value of an integer.
- Determine if an integer is a power of 2.
- Determine the minimum and maximum of two integers.

Potential solutions (in C)

Absolute value of integer

```
(v>0)*v+(v<0)*-v);
```

Power of 2?

```
v && !(v & (v-1));
```

Min and Max of two integers

```
min = y ^ ((x ^ y) & -(x < y));
max = x ^ ((x ^ y) & -(x < y));
```

If you like these

Hacker's Delight by Henry Warren.

Class project (for real)

We will solve this together as I want to see your homework done.

Squaring without multiplication

Write a function that takes a positive integer n and returns n-squared. You may use addition and subtraction but not multiplication or exponentiation.

- Provide algorithmic solution.
- Provide tests.
- Prove algorithm correct.
- Provide runtime.

Homework/Test questions

- What is the asymptotic runtime of the following

```
def f(a):
  n = len(a)
  for i in range(n):
    s=0
    for j in range(i,n):
      s = s + a[j]
      if s == 0:
         return a[i:j]
  return []
```

Homework/Test questions

For each function $f(n)$ in the table and time t, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

$f(n)$	1 second	1 minute	1 hour	1 day	1 month	1 year
$\log_2 n$						
$\sqrt{n}$						
n	10^6					
$n \log_2 n$						
n^2						
n^3						
2^n						
$n!$						