**Sorting**

Sort()

List {a1, a2, …, an} -------------------🡪 List{ a1, a2, …, an }

**Knowledge base:** Define a minimal language of only functions to implement sort(): if(), lt(), swap()

Set bounds to reduce search space.

* If(): min lg(n!) times
* Lt(): min lg(n!) times
* Swap(): min lg(n!) times\*

(\*) complexity limit islg(n!)

Proof: Binary decision tree has >= lg(n!) leaves

*(\*\*)lg(n!) information theoretic lower bound found on the* [*net*](http://en.wikipedia.org/wiki/Quicksort)

So there can’t be a correct algo which calls swap() less than log(n!) times.

Given the test a1 < a2 < a3 … < an

can we generate all possible input lists of sort function?

**Function generation**

* **Heuristic**

Getting closer to the result. Fitting better to the test.

Ex: Case of sorting 7 integers. We know the answer. While algo is working if it puts an element to its correct position => (it’s good) reinforce that part of the algo

* **Deterministic**
  + **Recursive**
    - **Known complexity**

Ex: If the complexity is n2 => f():{ f(); for(); } or f():{ f();f(); }

Ex: If the complexity is n(n+1)/2 => every recursion has +n calls; fn+1 = fn + n

Ex: If the complexity is log(n!) => fn+1 = fn + log(n+1)

* + - **Unknown complexity**
  + **Iterative**
    - **Known complexity**

Ex: If the complexity is n2 => f():{ for(); for(); }

* + - **Unknown complexity**

**Numerical:**

f0 = 0

f1 = 0 + 1

f2 = 0 + 1 + 2

f3 = 0 + 1 + 2 + 3

f4 = 0 + 1 + 2 + 3 + 4

f5 = 0 + 1 + 2 + 3 + 4 + 5

fn = 0 + 1 + 2 + … + n

**Recursive:**

f0 = 0

fn+1 = fn+ n *s()*

If you can compute the non-recursive part, you can compute the whole function’s complexity

f(list):{

if(n=0 || n=1) ret list;

if(n=2) s(0); ret list;

f(list-1);

for(i:n-1..1) s(i-1);

}

**Iterative:**

fn = n(n+1)/2 *s()*

n:for loop

f(list):{

for(i:n-2..0)

for(j:0..i) s(j);

}

F: Worst case count

Computational equivalence:

These are different representations of f. They are computationally equivalent. In the search space there may be other equivalent functions.

divide&conquer:

fn= 2f(n+1)/2+ (n+1)2/4 n:odd

fn= 2fn/2+ (n2-1)/2 + n/2 n:pair

Computational equivalence:

**Numerical:**

Fn = log 0 + log 1+ … + log n

**Recursive:**

f0 = 0

fn+1 = fn+ log(n+1) *s()*

If you can compute the non-recursive part, you can compute the whole function’s complexity

**Iterative:**

fn = log n! *s()*

Insertion sort ≡ Merge sort

Is it always possible to convert a rec func to iterative form?

Usage of f: Compilation finds all working solutions for f. For each, we know cpu&memory usage (and it’s distributable or not).

At runtime, we can choose the best fitting f implementation in func of env (cpu, memory, # of cpu).

Can we convert one f to another which is distributable?

For Sum(n) = 1 + 2 + 3 + … + (n-1) + n, c=n.add() => n cycle

For n(n+1)/2, c=add()+mul()+div() => 3 cycle

If we know weight of all functions, we can optimize (choose faster)

If f(n) = n(n+1) defined, Sum(n) = f(n) / 2 => 2 cycle

Sort(): n! info

SwapIfLt(): 2 info

Sort=n(n-1)/2 SwapIfLt ???

Test(): => 2^n-1 info

for(i:0..n-1)

if(a[i]<a[i+1])

ok

n(n-1)/2 f pkoi?

n! -------------------🡪1

n-1 f

n! -------------------🡪(n-1)!

f

2 -------------------🡪1 =>n=2

3f

6 -------------------🡪1 =>n=3

6f

24 -----------------🡪1 =>n=4

10f

120----------------🡪1 =>n=5

swapIfLt(list, i){ *// n!, n-1*

if( list[i] > list[i+1] ) *// 2*

swap(list, i, i+1); *// ?*

}

1 bubleSort() = if() + swap()

* Solving for 9,5

Result: f(0); 0

* Solving for 9,5,4

Result: f(0); f(1); f(0);

Found loop: f(n,m) = -m+n [n <= 2, m <= n]

for(n: 1..2)

for(m: 1..n)

x = -m+n

Found loop: f(n,m) = m-1 [n <= 2, m <= -n+3]

for(n: 1..2)

for(m: 1..-n+3)

x = m-1

Result: f(1); f(0); f(1);

Found loop: f(n,m) = m-n+1 [n <= 2, m <= n]

for(n: 1..2)

for(m: 1..n)

x = m-n+1

Found loop: f(n,m) = -m+2 [n <= 2, m <= -n+3]

for(n: 1..2)

for(m: 1..-n+3)

x = -m+2

* Solving for 9,5,4,2 Full iterative -+1 (3 segments)

Result: f(0); f(1); f(0); f(2); f(1); f(0); 010210 => fn-1 + n 0,10,210

Result: f(0); f(1); f(2); f(1); f(0); f(1); 012101 => n + fn-1 012,10,1

Result: f(0); f(2); f(1); f(0); f(2); f(1); 021021 0,210,21

Result: f(0); f(2); f(1); f(2); f(0); f(1); 021201

Result: f(1); f(0); f(2); f(1); f(0); f(2); 102102 10,210,2

Result: f(1); f(2); f(0); f(1); f(0); f(2); 120102

Result: f(1); f(2); f(1); f(0); f(1); f(2); 121012 1,210,12

Result: f(2); f(1); f(0); f(2); f(1); f(2); 210212 210,21,2

Result: f(0); f(1); f(0); f(2); f(1); f(0);

Found loop: f(n,m) = -m+n [n <= 3, m <= n]

for(n: 1..3)

for(m: 1..n)

x = -m+n

Result: f(0); f(1); f(2); f(1); f(0); f(1);

Result: f(0); f(2); f(1); f(0); f(2); f(1);

Result: f(0); f(2); f(1); f(2); f(0); f(1);

Result: f(1); f(0); f(2); f(1); f(0); f(2);

Result: f(1); f(2); f(0); f(1); f(0); f(2);

Result: f(1); f(2); f(1); f(0); f(1); f(2);

Result: f(2); f(1); f(0); f(2); f(1); f(2);

Found loop: f(n,m) = -m+3 [n <= 3, m <= -n+4]

for(n: 1..3)

for(m: 1..-n+4)

x = -m+3

## How to solve loops

**Solving simple arithmetic loop (order=1)**

n:1,2,…,En

f(n) = kn + d n<En

Ex :

1,2,3,4, 5

3,5,7,9,11

a1=k+d

a2=2k+d

* k=a2-a1 d=2a1-a2
* f(n)=(a2-a1)n+2a1-a2

**Solving embedded arithmetic loop (order=2)**

n:1,2,3,…,En

m:1,2,3,…,(k’’n + d’’)

f(n,m) = knm + dm + k’n + d’ n <= En, m <= k’’n + d’’

Ex :

f:1,2,3,1,2,1 n<=3 m<=-n+4

a1 = f(1,1) = (k+d) +k’+d’

a2 = f(1,2) = (k+d)2 +k’+d’

a3 = f(1,3) = (k+d)3 +3k’+d’

a4 = f(2,1) = (2k+d) +2k’+d’

a5 = f(2,2) = (2k+d)2 +2k’+d’

a6 = f(3,1) = (3k+d) +3k’+d’

k =…

k’=…

d=…

d’=…

* k=0, d=1, k’=0, d’=0
* f(n,m) = m

**Grouping for order 2**

a1,a2,a3,a4,a5,a6 C:count

n<=En, m<=k’’n+d’’

n=1 => m Є {1,2,…k’’+d’’}

n=2 => m Є {1,2,…2k’’+d’’}

n=3 => m Є {1,2,…3k’’+d’’}

…

n=En => m Є {1,2,…Enk’’+d’’} +

---------------------------------------------

=C

1<En<C

k’’+d’’>=1, 2k’’+d’’>=1, …, Enk’’+d’’>=1

**Solving embedded arithmetic loop (order=3)**

f(n,m) = k1 nmp + k2 nm + k3 np + k4 mp + k4 n + k5 m + k6 p + d

n <= En, m <= k’’n + d’’, p <= k’’’nm + d’’’m + k’’’n + d’’’

I don’t find any solution for series 1, 4, 14, 46, 146 with the arithmetic solver. But it is f(n)=5f(n-1)-6f(n-2), f(0)=1,f(1)=4 (closed form:2\*3^n-2^n ). Is there a recursive solution for suite 0,1,0,1,2,3 ?

A better embedded loop solver

Ex: 1,2,3,1,2,1

for n :1..3

for m:1..2

for k :1..1

* for n :3..1

for m:1..n

Find groups => Search for loop 1 => Goto Start

How do we sort a list of N numbers ?

There is a variable in this question; N. So we will try to solve it for different values of N. i.e N=1, N=2, N=3,etc… and then we will try to generalize the solution for any N.

These solutions are all same; different representations of En(En+1)/2 iterations. 2 embedded for loops have always 4 different representations. If you know one, you can reproduce the 3 others.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Iter** | **Ex** | **Solution** | **Solution in loop** | **En** | **k’’** | **d’’** | **k** | **d** | **k’** | **d’** |
| N= 1 | 9 |  |  |  |  |  |  |  |  |  |
| N= 2 | 9,5 | f(0); |  |  |  |  |  |  |  |  |
| N= 3 | 9,5,4 | f(0); f(1); f(0);  f(1); f(0); f(1) ; | for(n: 1..2) for(m: 1..n) f(-m+n);  for(n: 1..2) for(m: 1..-n+3) f(m-1);  for(n: 1..2) for(m: 1..n) f(m-n+1);  for(n: 1..2) for(m: 1..-n+3) f(-m+2); | 2  2  2  2 | 1  -1  1  -1 | 0  3  0  3 | 0  0  0  0 | -1  1  1  -1 | 1  0  -1  0 | 0  -1  1  2 |
| N= 4 | 9,5,4,2 | f(0); f(1); f(0); f(2); f(1); f(0);  f(0); f(1); f(2); f(0); f(1); f(0);  f(2); f(1); f(2); f(0); f(1); f(2);  f(2); f(1); f(0); f(2); f(1); f(2); | for(n: 1..3) for(m: 1..n) f(-m+n);  for(n: 1..3) for(m: 1..-n+4) f(m-1);  for(n: 1..3) for(m: 1..n) f(m-n+2);  for(n: 1..3) for(m: 1..-n+4) f(-m+3); | 3  3  3  3 | 1  -1  1  -1 | 0  4  0  4 | 0  0  0  0 | -1  1  1  -1 | 1  0  -1  0 | 0  -1  2  3 |
| N= 5 | 9,5,4,2,1 | f(0); f(1); f(0); f(2); f(1); f(0); f(3); f(2); f(1); f(0);  f(0); f(1); f(2); f(3); f(0); f(1); f(2); f(0); f(1); f(0);  f(3); f(2); f(3); f(1); f(2); f(3); f(0); f(1); f(2); f(3);  f(3); f(2); f(1); f(0); f(3); f(2); f(1); f(3); f(2); f(3); | for(n: 1..4) for(m: 1..n) f(-m+n);  for(n: 1..4) for(m: 1..-n+5) f(m-1);  for(n: 1..4) for(m: 1..n) f(m-n+3);  for(n: 1..4) for(m: 1..-n+5) f(-m+4); | 4  4  4  4 | 1  -1  1  -1 | 0  5  0  5 | 0  0  0  0 | -1  1  1  -1 | 1  0  -1  0 | 0  -1  3  4 |

We see that there is a linear relation between N and En,d’’,d’ at levels N=3 and N=4

All these solutions are equivalent? Complexity n

*Yes! All solutions running in complexity n are equivalent. If you find 1 of them, you can generate mechanically all other iterative and recursive ones.*

How many equivalent solutions out there?

We can generalize a solution at the level of n=4.

1M elements: n! is too big for 1M but the solution is so simple at n=4. Runtime of the solution is another question.

SO, for other hard algos may be the solution can be generalized at a very early level.

TSP algo may be generalized at n=5,6 ??

The time needed to find a solution does not depend on the size of the question (n). But it depends on how complex is the pb (how the pb is modelled). SwapIfLt() gives a solution at n=4.

Connect4,TSP,Sorting,HT => Easy to model. These are reducible questions; Connect3,sort5, TSP7…

Chess is not reducible.

Is bg reducible to 8 cases,2 sided dices and 5 pawns (instead of 24,6,15)? Can you solve for 3 and find invariants, and then you can search for the big real bg?

How many ways to put 5 balls to 8 bags?

5: C(8,1) 8=8

4,1: C(8,2) 8.7=56

3,2: C(8,2) 8.7=56

3,1,1: C(8,3) / 2! 8.7.6/2=168

2,2,1: C(8,3) / 2! 8.7.6/2=168

2,1,1,1: C(8,4) / 3! 8.7.6.5/6=420

1,1,1,1,1:C(8,5) / 5! 8.7.6.5.4/5!=56

=932

You search for a reversing algo (i.e. convert 9,4,2,1 to 1,2,4,9) with swapNext(list, x).

You call swapNext conditionally and you find the sorting(!) algo.

|  |  |
| --- | --- |
| Reverse | Sort |
| for i: 0..n  for j: i..0  swapNext(list, i); | for i: 0..n  for j: i..0  if(list[i] > list[i+1])  swapNext(list, i); |

Sequential pattern mining, sequence mining

Generating functionology

Function optimization process:

1. swapIfLt()
2. n(n+1)/2 s()
3. fn+1 = fn+ n *s()*
4. fn+1 = fn+ log n *insertAt()*

insertBinary(start, end, x){

if(start==end) insertAt(n, x);

if(x < a[n/2]) insertBin(0, n/2, x);

else insertBinary(n/2, n, x);

}

in = in/2 + 1 insertAt() => in = log n insertAt()

1 : 0 => n=2

3 : 0 1 0 => n=3

6 : 0 1 2 0 1 0 => n=4

10: 0 1 2 3 0 1 2 0 1 0 => n=5

Proof by recursion: Assuming fn = n(n-1)/2, fn+1 = fn + n ??

LoopSolver:

* SequencialLoopSolver
  + Arithmetic an+k
  + Geometric an\*k
* RecursiveLoopSolver

We don’t want to find a way to compute, we want just to specify (and let the tool find a way to compute)

|  |  |  |
| --- | --- | --- |
| Environment | Knowledge base & rules(what is doable?) | Objective (what we want?) |
| 9  5  4  2 | Swap box with the next one | 2  4  5  9 |
| int[] list | swapIfLt() | for(i:1..n) list[i]<list[i+1] |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |   Connect 4 | Add pawn | Make 4 in a line |
|  | +, \*, / | 1+2+3+…+n |
| Hanoi towers: | Move disk  (small disk over big one) |  |
| stack[3]; | Move(a,b)  Stack[b].Push(stack[a].pop());  Rules:  -Small over big  -no move(s) that bring to the first (or preceding) state. Ex: move(x,x) or move(x,y);move(y,x); |  |
| [Dots and boxes:](https://www.youtube.com/watch?v=KboGyIilP6k) |  |  |
| Knights tour |  |  |

Redefine your algo as a search pb. Then can compute how optimal your algo is.

What is the complexity bounds of HasDoublicate(list of n):bool function?

Sort&search: lg(n!)+n

Program verification

Contracts c#

Dafny language

Interactive theorem proving => isabelle

Automating theorem proving with SMT

The very basic function is a transistor.

With that you can construct and, or, xor… gates. And with gates you can construct complex functions.

Instead of having all 2 digit binary numbers; 00, 01, 10, 11

You can define their position in 0011. 00=>0, 01=>1, 11=>2, 10=>3

2 bits: 0011

3 bits: 10001110

4 bits: 1101000011110010

You can not compress the info. The preceding transformation conserves the same amount of bits.

A prime number P has the info of all primes up to sqrt(P). Which is equal to the info of all composite numbers.

To accelerate algo search you can eliminate algos that a swap series comes back to a previous state. You will be still finding (in 5.5s) a result for(n: 1..4) for(m: 1..n) f(-m+n);

**Solving for 9,5,4**

Leaf Count: 2

Result: f(0); f(1); f(0);

Found loop: f(n,m) = -m+n [n <= 2, m <= n]

for(n: 1..2) for(m: 1..n) f(-m+n);

Found loop: f(n,m) = m-1 [n <= 2, m <= -n+3]

for(n: 1..2) for(m: 1..-n+3) f(m-1);

Result: f(1); f(0); f(1);

Found loop: f(n,m) = m-n+1 [n <= 2, m <= n]

for(n: 1..2) for(m: 1..n) f(m-n+1);

Found loop: f(n,m) = -m+2 [n <= 2, m <= -n+3]

for(n: 1..2) for(m: 1..-n+3) f(-m+2);

**Solving for 9,5,4,2**

Leaf Count: 15

Result: f(0); f(1); f(0); f(2); f(1); f(0);

Found loop: f(n,m) = -m+n [n <= 3, m <= n]

for(n: 1..3) for(m: 1..n) f(-m+n);

Result: f(0); f(1); f(2); f(1); f(0); f(1);

Result: f(0); f(2); f(1); f(0); f(2); f(1);

Result: f(0); f(2); f(1); f(2); f(0); f(1);

Result: f(1); f(2); f(0); f(1); f(0); f(2);

Result: f(2); f(0); f(1); f(2); f(0); f(1);

Result: f(2); f(1); f(0); f(1); f(2); f(1);

**Solving for 9,5,4,2,1**

Leaf Count: 672

Result: f(0); f(1); f(0); f(2); f(1); f(0); f(3); f(2); f(1); f(0);

Found loop: f(n,m) = -m+n [n <= 4, m <= n]

for(n: 1..4) for(m: 1..n) f(-m+n);

Construct the function Sort() with swapIfGt()

There are N number of optimal solutions: So={Sort1(), Sort2(),…SortN}

For a max complexity n(n-1)/2 how many different algos you can construct with swapIfGt() ?

= n-1 ^ n(n-1)/2

Prop2: Given a solution SortX() you can find all other solutions (by using the counting function method)

Prop3: There are several reducibility rules for swapIfGt() (how many?)

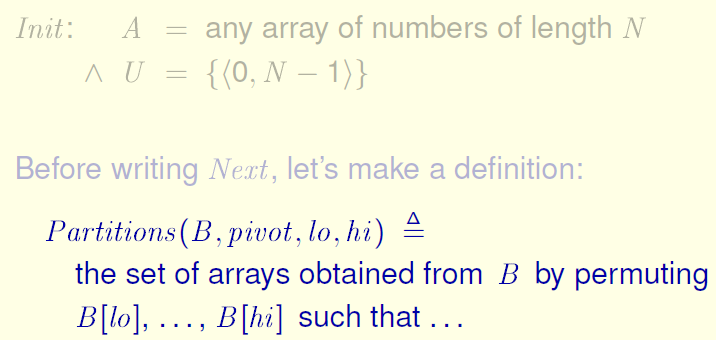
If you construct any function f() respecting reducibility rules then f € So

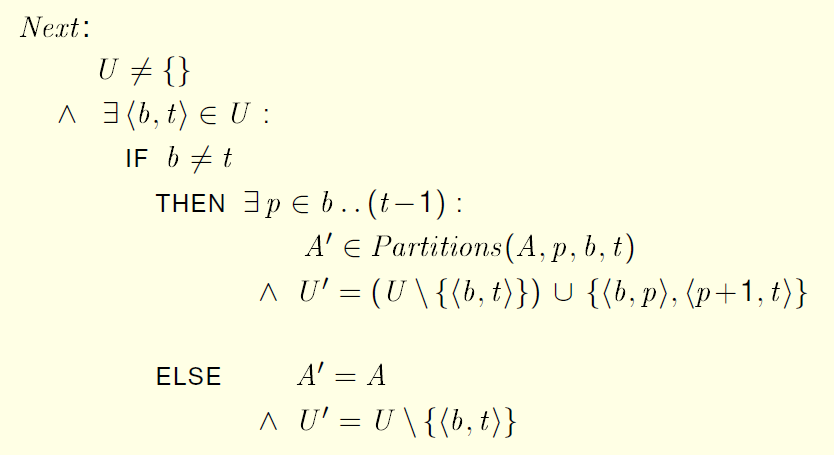
If a solution is optimal then it can’t have any reducible block

So instead of solving Sorting pb you can find all reducibility rules.

If you construct a function f() respecting all reduc rules, f() necessarily solves the pb and f() is optimal.

SortSolver’s complexity is compl of finding reduc rules





You can decompile assembly or IL and examine results. It can be useful as different compilers (and different compiler options for optimizations) give different results.

<http://www.azillionmonkeys.com/qed/sort.html>

Sort algos compiled with Intel C/C++ with options /O2 /G6 /Qaxi /Qxi /Qip are x2 faster than others.

Machine Learning déf: Arthur Samuel (1959):Field of study that gives computers the ability to learn without being explicitly programmed.

Machine learning => pattern recognition

Read about Formal Concept analysis

* <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.6746&rep=rep1&type=pdf>
* https://www.youtube.com/watch?v=Xuxm929tIRY