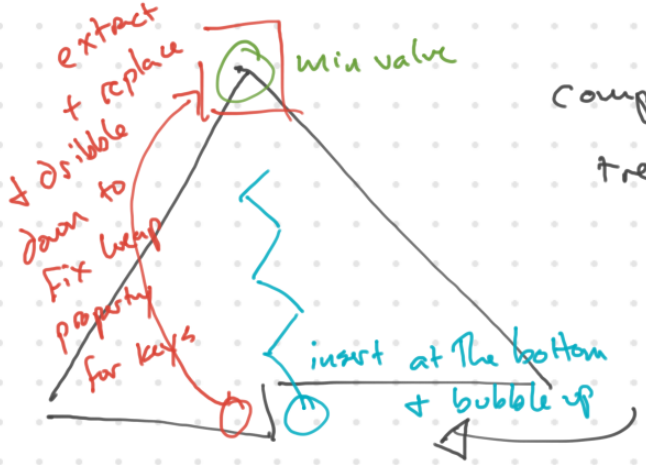
Fibonacci Heaps

Recall a min Heap



complete binary tree except possibly the last row

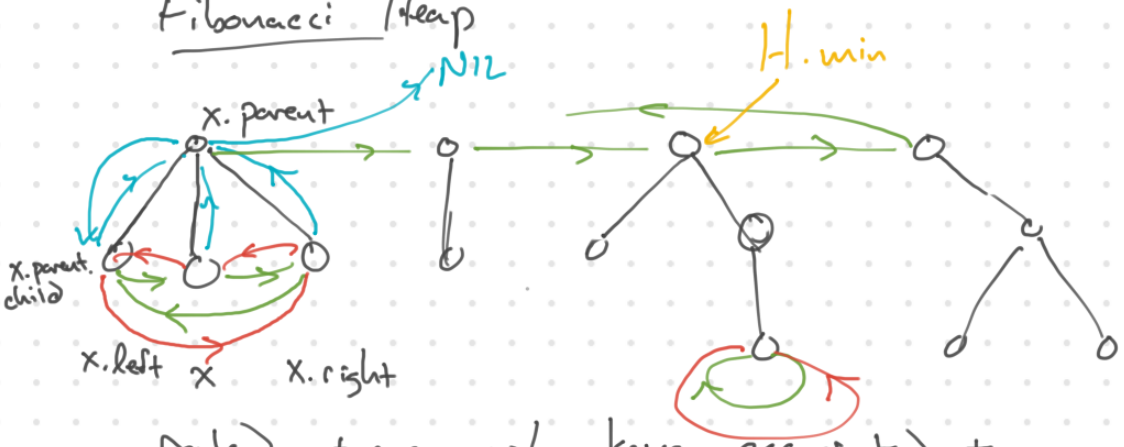
Keys for each node

$x.parent.key \leq x.key \quad \forall x, \neq x$ not the root.

	Heap	Fibonacci Heaps FH
initiate make_heap	$O(1)$	$O(1)$
insert	$O(\log n)$	$O(1)$
min	$O(1)$	$O(1)$
extract_min	$O(\log n)$	$O(\log n)$
union	$O(n)$	$O(1)$
Decrease key	$O(\log n)$	$O(1)$
delete	$O(\log n)$	$O(\log n)$
	↑ worst case complexity	↑ amortized analysis as opposed to worst case

Major caveat: operations have significant constant factors in execution $\Rightarrow$ better only in theory

Fibonacci Heap



rooted trees w/ keys associated to the nodes

$x.key \geq x.parent.key$ if $x.parent \neq NIL$

Obs The min^ key value ~~of any~~ ~~or~~ over all nodes of the ~~no~~ must be at

The root of some component

So we keep H.min - pointer to min key

Keep a double linked list for children of any vertex

parent pointer for each vertex

+ a pointer to one child.

Note x.parent.child can be distinct from x

linked list for the roots

Note roots are not ordered in any particular way.

a leaf has x.left = x.right = x

Def $H.n$:= # of nodes in H

we also have a marker $x.mark$ for each node x = T or F

Note $x.mark$ is only used to ensure the proper bounds on the complexity.

We use potential method to prove amortized bounds on the complexity w/ potential function

$$\Phi(H) = T(H) + 2M(H)$$

$T(H)$: # of components

$M(H)$: $\sum_{x \in V} (x.mark = T)$

Operations

for amortized complexity, we need to bound

$$c_i + \Phi(H_i) - \Phi(H_{i-1})$$

where c_i is cost of going from H_{i-1} to H_i

1 creating a new FH

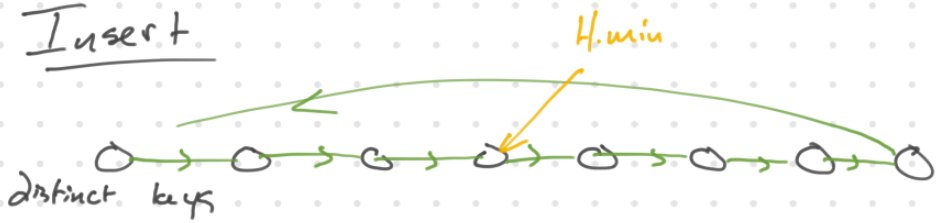
to make an empty FH

set $H.n = 0$ $H.min = NIL$

$T(H) = 0$ $M(H) = 0$

$\Phi(H_i) = 0$ + amortized cost = $O(1)$

Insert



We could even have a FH where every component is a single vertex.

FH.insert (H, x)

x.deg = 0 ← note we're also counting the deg of every vertex.
x.p = NIL
x.child = NIL
x.mark = False

Insert x in list of roots

if H.min = NIL

H.min = x

else

if x.key < H.min.key

H.min = x

$H_{an} = H_{an} + 1$

H' is the new heap

to calculate amortized cost

$T(H') = T(H) + 1$

$m(H') = m(H)$

amortized cost $\leq O(1) + \Phi(H') - \Phi(H)$

$= O(1) + T(H) + 1 + 2m(H) -$

$- T(H) - 2m(H)$

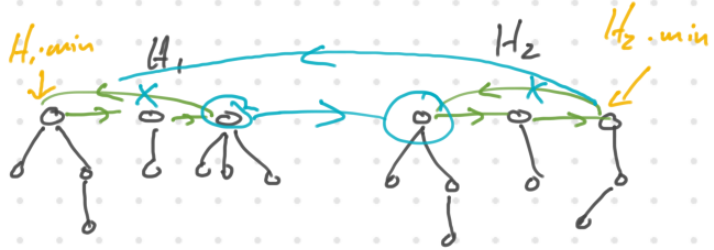
$= O(1) + 1 = O(1)$

✓

3. Find_min

return H.min

4. merging two FH



This is why we wanted a double linked list of the roots by rewriting a constant # of pointers, we can concatenate the list of roots (Note that this wouldn't be possible if the roots were stored as arrays)

FH_union(H_1, H_2)

H = make FH()

H.min = H_1.min

concatenate root list of H_1 & H_2 to that of H

if H_1.min = NIL or H_2.min < H_1.min

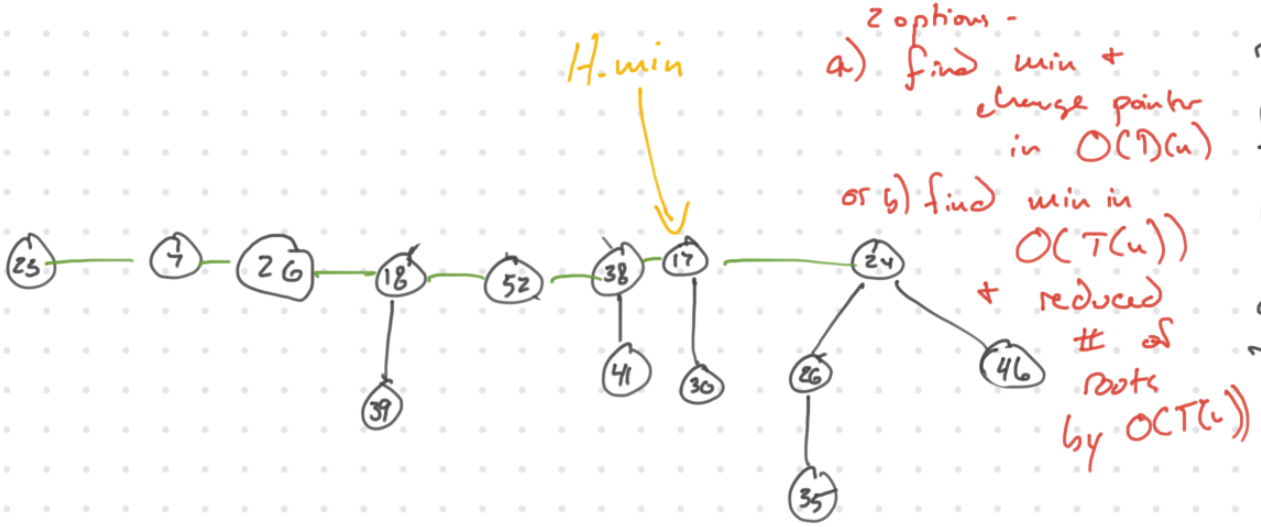
H.min = H_2.min

H.n = H_1.n + H_2.n

return H.

everything is constant

amortized cost = $O(1) + T(H) + 2m(H) - T(H_1) - T(H_2) - 2m(H_1) - 2m(H_2)$



2 options -

a) find min + change pointer in $O(D(n))$

or b) find min in $O(T(n))$ + reduced # of roots by $O(T(n))$

We're going to assume that $\deg(v) \leq D(n)$ for every vertex v because our runtime will actually be a function of $D(n)$

~~cost~~ amortized cost

a) $O(D(n)) + T(H) + O(D(n)) + 2m(H) - T(H) - 2m(H)$

works! we get $O(D(n))$

Since we have a list of roots - to find $\min$ forces option b

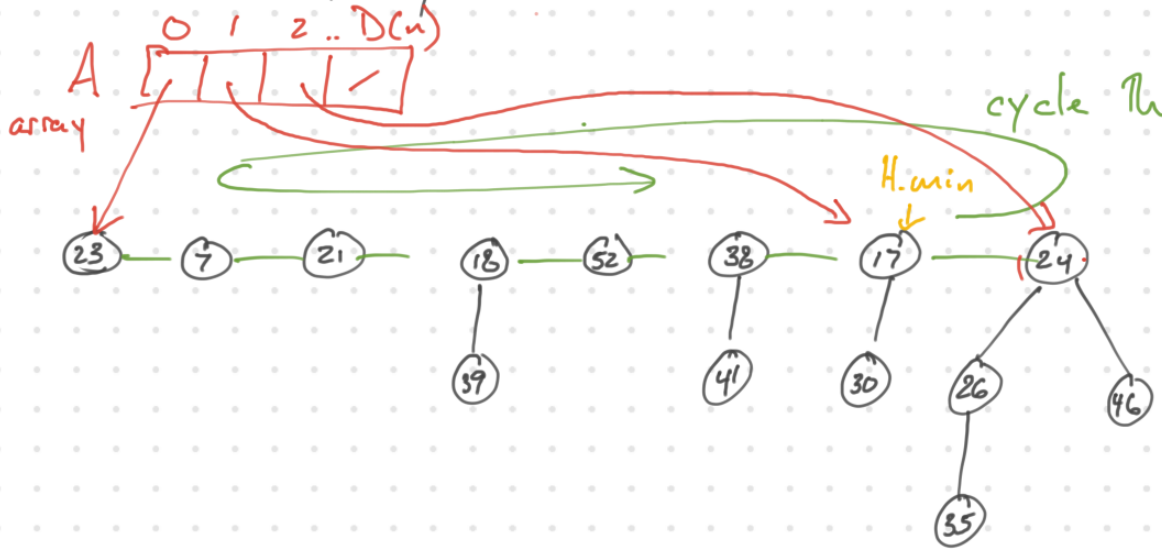
b) $O(T(n)) + O(D(n) + 2m(H) - T(n) - 2m(H)$

$= O(D(n))$

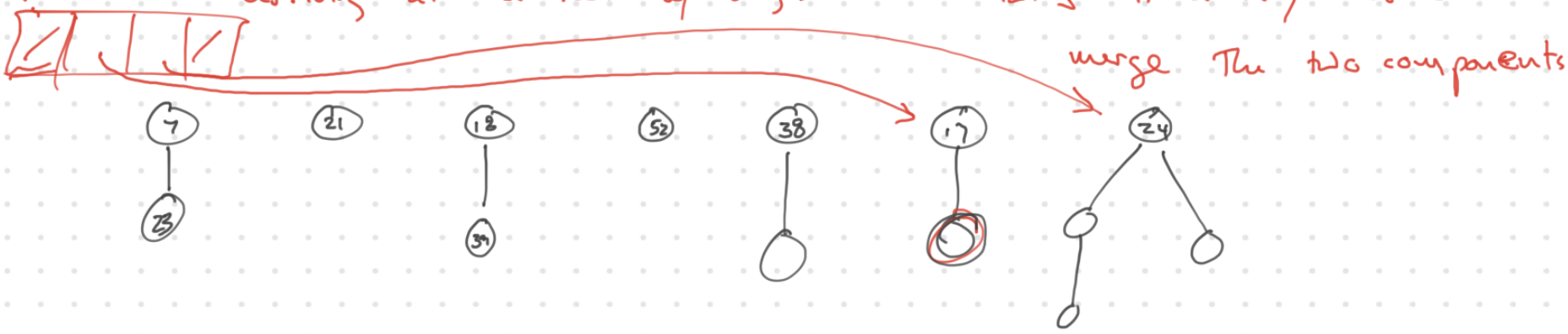
works too!

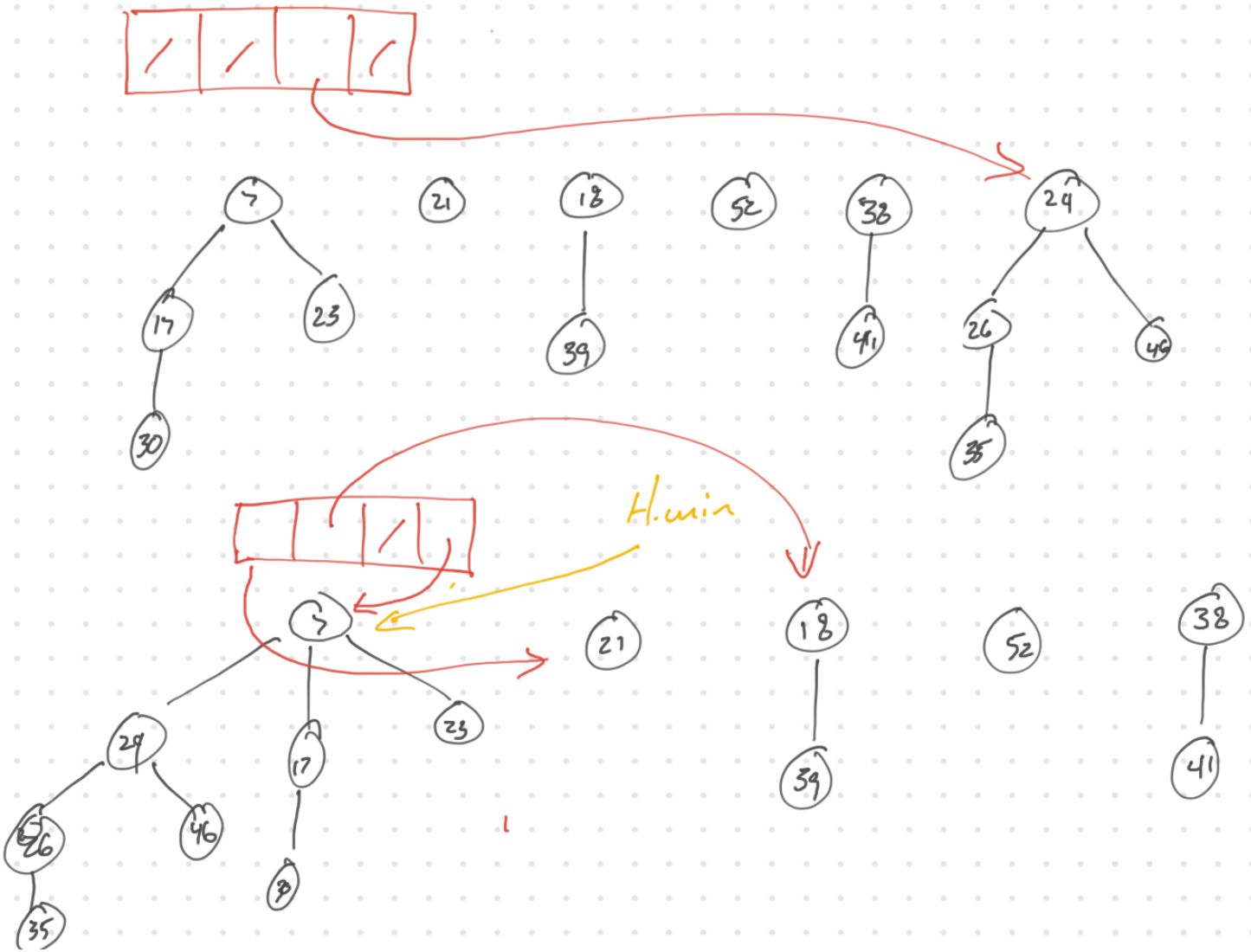
{ we wanted final runtime to be $O(\log n)$ - so we need a theoretical result that $\deg \leq \log n$)

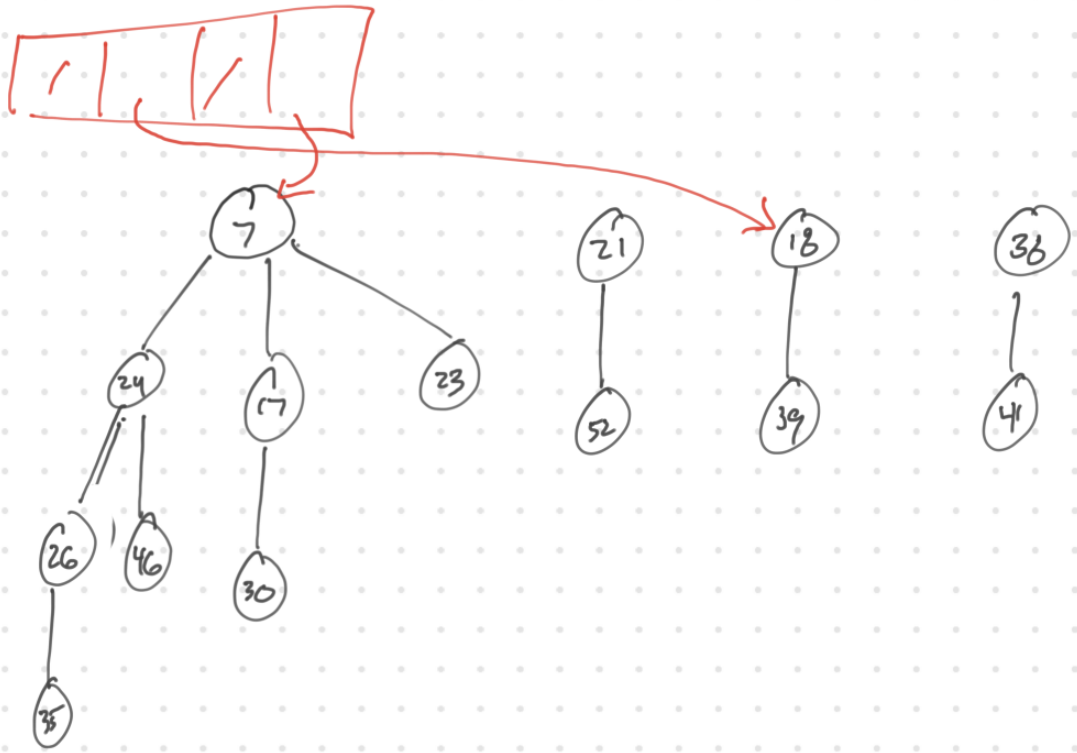
So for complexity to work out, we need to reduced the # of roots



arriving at a root w/ deg = k ST A[k] is already defined




7
21
18
52
38
24
17
23
39
41
26
46
30
35
H.min
7
21
18
52
38
24
17
23
39
41
26
46
35
8


7
24
17
23
26
46
30
38
21
52
18
39
36
41

```
FIB-HEAP-EXTRACT-MIN(H)
 1  z = H.min
 2  if z ≠ NIL
 3       for each child x of z
 4            add x to the root list of H
 5            x.p = NIL
 6       remove z from the root list of H
 7       if z == z.right
 8            H.min = NIL
 9       else H.min = z.right
10            CONSOLIDATE(H)
11       H.n = H.n − 1
12  return z
```

move each child up to be a root

found a min-temp to proceed with

long complicated argument we made reducing the comps.

```
CONSOLIDATE(H)
1  let A[0..D(H.n)] be a new array
2  for i = 0 to D(H.n)
3      A[i] = NIL
4  for each node w in the root list of H
5      x = w
6      d = x.degree
7      while A[d] ≠ NIL
8          y = A[d]        // another node with the same degree as x
9          if x.key > y.key
10             exchange x with y
11         FIB-HEAP-LINK(H, y, x)
12         A[d] = NIL
13         d = d + 1
14     A[d] = x
15 H.min = NIL
16 for i = 0 to D(H.n)
17     if A[i] ≠ NIL
18         if H.min == NIL
19             create a root list for H containing just A[i]
20             H.min = A[i]
21         else insert A[i] into H's root list
22             if A[i].key < H.min.key
23                 H.min = A[i]

FIB-HEAP-LINK(H, y, x)
1  remove y from the root list of H
2  make y a child of x, incrementing x.degree
3  y.mark = FALSE
```

array of possible root degrees A in the example

cycle through each root

updating A - try to add a pointer from ☆ A[deg(w)] to w

+ if A[deg(w)] is already defined, merge w + ~~the other~~ the comp indicated by ☆ A[deg(w)]

after considering all possible roots w, when this stops, every comp is indicated by an elt. of A ie we have ≤ D(n) comps

search through the comps to find min in $O(D(n))$

amortized cost: The amt of work done over the for loop

split into

work in while loop

work outside while loop

$O(T(H) + D(u))$

summed over all passes total

$O(1)$ for each pass of while $= O(T(H) + D(u))$

$$\Rightarrow \text{total } O(T(H) + D(u)) + \overline{\Phi}(H') - \overline{\Phi}(H)$$

$O(T(H) + D(u))$ $=$ c_i

$\overline{\Phi}(H') = T(H') - M(H') = D(u) - M(H')$

$\overline{\Phi}(H) = T(H) - M(H)$

we didn't talk about marked vertices because the # can only decrease

amortized cost = $O(T(n) + D(n)) + D(n) - T(n) + \Phi(H') - \Phi(H)$

cancels out

$O(D(n))$