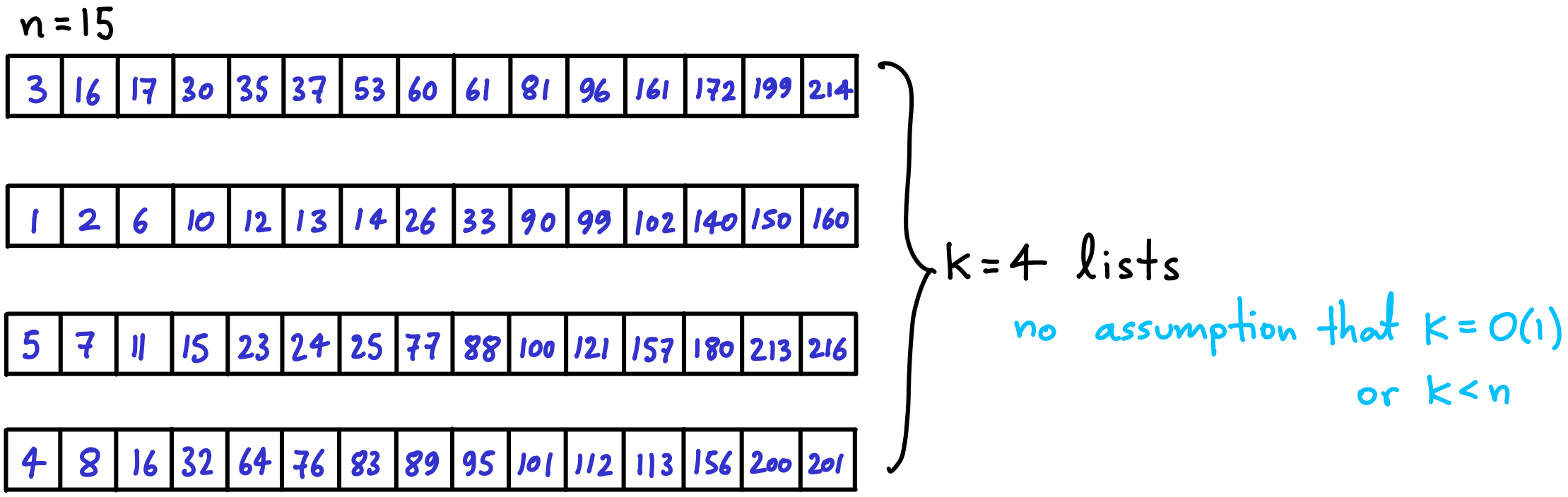
Given k sorted lists, each with n elements

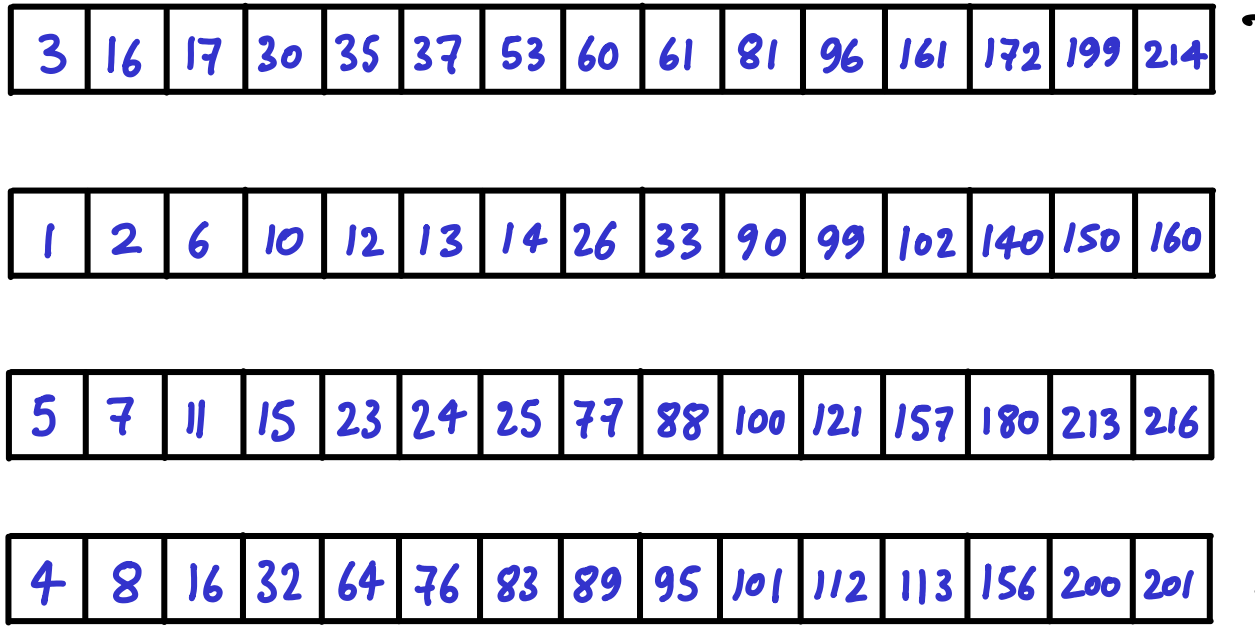


Given k sorted lists, each with n elements

we want to search for x in each list  // output = {YES or NO} per list

time? $O(\ )$ $\Omega(\ )$

n = 15




no assumption that $k = O(1)$ or $k < n$

Given k sorted lists, each with n elements

we want to search for x in each list

For k=1, $\Omega(\log n)$ time.

To report output, $\Omega(k)$ time

} $\Omega(k+\log n)$

Easy to do $O(k \cdot \log n)$

n = 15

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 16 | 17 | 30 | 35 | 37 | 53 | 60 | 61 | 81 | 96 | 161 | 172 | 199 | 214 |
| 1 | 2 | 6 | 10 | 12 | 13 | 14 | 26 | 33 | 90 | 99 | 102 | 140 | 150 | 160 |
| 5 | 7 | 11 | 15 | 23 | 24 | 25 | 77 | 88 | 100 | 121 | 157 | 180 | 213 | 216 |
| 4 | 8 | 16 | 32 | 64 | 76 | 83 | 89 | 95 | 101 | 112 | 113 | 156 | 200 | 201 |

k = 4 lists

no assumption that $k = O(1)$ or $k < n$

$\Omega(k+\log n)$ Easy $O(k \cdot \log n)$

Make one sorted array ?

$\Omega(k+\log n)$ Easy $O(k \cdot \log n)$

Make one sorted array ? binary search : $O(\log nk) = O(\log n) + O(\log k)$

$\Omega(k+\log n)$ Easy $O(k \cdot \log n)$

Make one sorted array ? binary search : $O(\log nk) = O(\log n) + O(\log k)$

↳ if every element points to its original list: handle duplicates in $O(k)$

got $O(k+\log n)$

$\Omega(k+\log n)$ Easy $O(k \cdot \log n)$

Make one sorted array ? binary search : $O(\log nk) = O(\log n) + O(\log k)$

↳ if every element points to its original list: handle duplicates in $O(k)$

→ got $O(k+\log n)$

...but... Merge: $k$ objects, divide & conquer $\log k$ levels $O(nk \cdot \log k)$

preprocessing

$\Omega(k+\log n)$ Easy $O(k \cdot \log n)$

Make one sorted array ? binary search : $O(\log nk) = O(\log n) + O(\log k)$

↳ if every element points to its original list: handle duplicates in $O(k)$

got $O(k+\log n)$

...but... Merge: $k$ objects, divide & conquer $\log k$ levels $O(nk \cdot \log k)$

preprocessing

---

Extra requirement: report predecessor & successor in each list

# $\Omega(k+\log n)$ Easy $O(k \cdot \log n)$

Make one sorted array? binary search : $O(\log nk) = O(\log n) + O(\log k)$

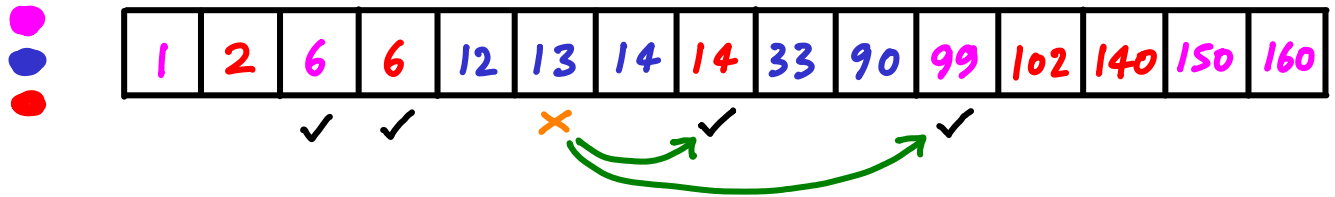
↳ if every element points to its original list: handle duplicates in $O(k)$

↳ got $O(k+\log n)$

...but... Merge: $k$ objects, divide & conquer $\log k$ levels $O(nk \cdot \log k)$ preprocessing

---

Extra requirement: report predecessor & successor in each list




// length $nk$

Store pointer from every position to $k$ elements? → $O(nk^2)$ space

Given $k$ sorted lists, each with $n$ elements

we want to search for $x$ in each list & report pred./succ.

For $k=1$, $\Omega(\log n)$ time.
To report output, $\Omega(k)$ time
} $\Omega(k+\log n)$

Easy to do $O(k \cdot \log n)$

Sort all: $O(nk \log k)$ preprocess
$O(nk^2)$ space
$O(k+\log n)$ search

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 16 | 17 | 30 | 35 | 37 | 53 | 60 | 61 | 81 | 96 | 161 | 172 | 199 | 214 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 6 | 10 | 12 | 13 | 14 | 26 | 33 | 90 | 99 | 102 | 140 | 150 | 160 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 7 | 11 | 15 | 23 | 24 | 25 | 77 | 88 | 100 | 121 | 157 | 180 | 213 | 216 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 8 | 16 | 32 | 64 | 76 | 83 | 89 | 95 | 101 | 112 | 113 | 156 | 200 | 201 |

Given $k$ sorted lists, each with $n$ elements

we want to search for $x$ in each list & report pred./succ.

For $k=1$, $\Omega(\log n)$ time.
To report output, $\Omega(k)$ time
} $\Omega(k+\log n)$

Easy to do $O(k \cdot \log n)$

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 16 | 17 | 30 | 35 | 37 | 53 | 60 | 61 | 81 | 96 | 161 | 172 | 199 | 214 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 6 | 10 | 12 | 13 | 14 | 26 | 33 | 90 | 99 | 102 | 140 | 150 | 160 |

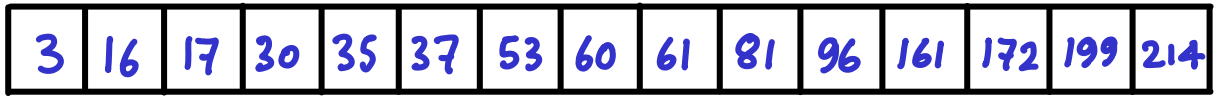
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 7 | 11 | 15 | 23 | 24 | 25 | 77 | 88 | 100 | 121 | 157 | 180 | 213 | 216 |

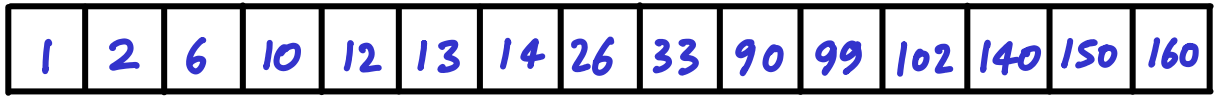
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 8 | 16 | 32 | 64 | 76 | 83 | 89 | 95 | 101 | 112 | 113 | 156 | 200 | 201 |

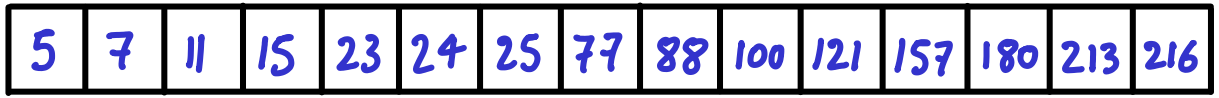
Sort all: $O(nk \log k)$ preprocess
$O(nk^2)$ space
$O(k+\log n)$ search

Fractional cascading

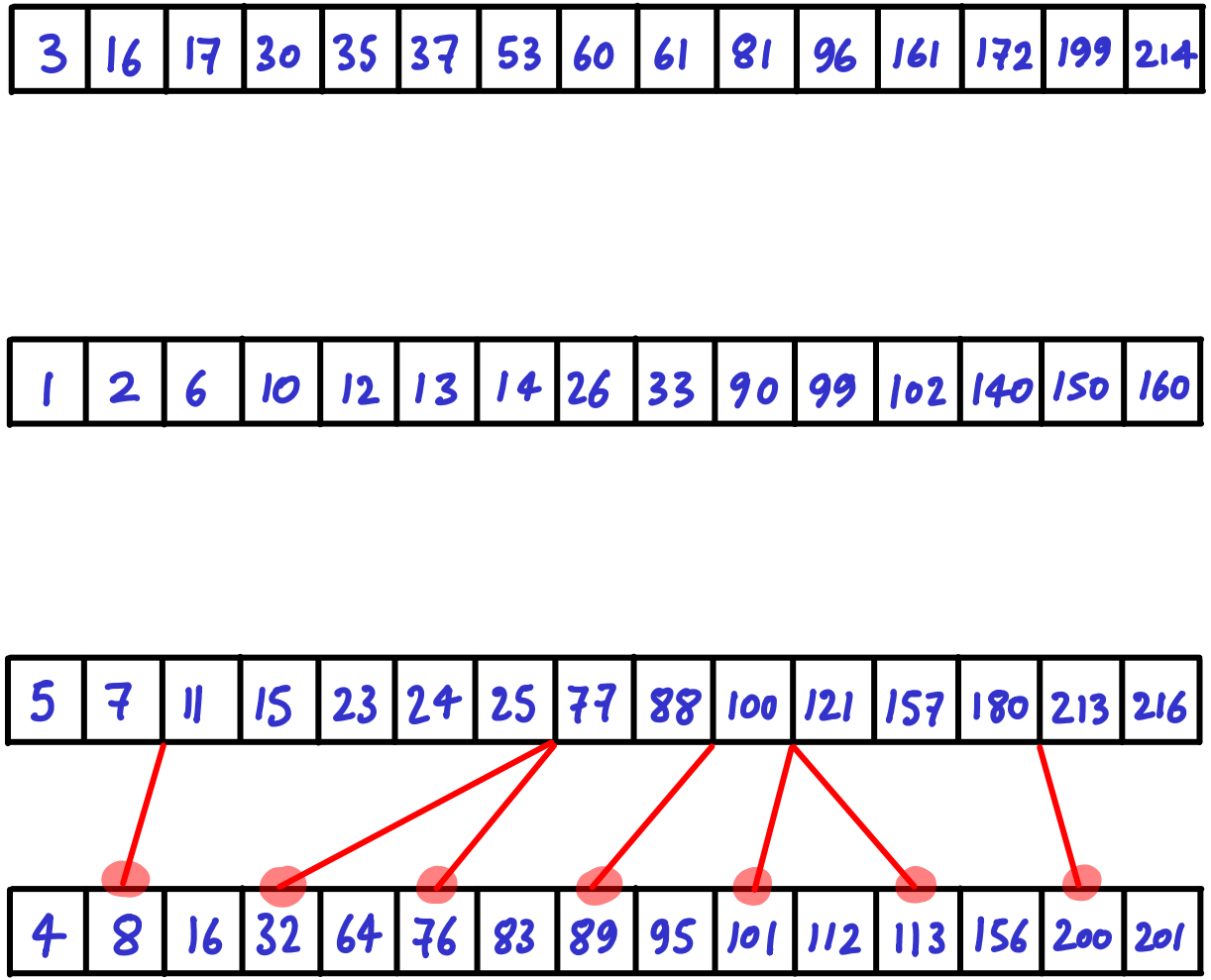
! $O(nk)$ preprocess
! $O(nk)$ space
✓ $O(k+\log n)$ search

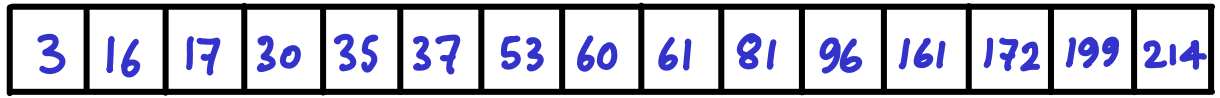

3 16 17 30 35 37 53 60 61 81 96 161 172 199 214

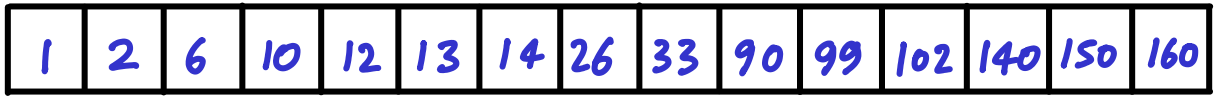

1 2 6 10 12 13 14 26 33 90 99 102 140 150 160

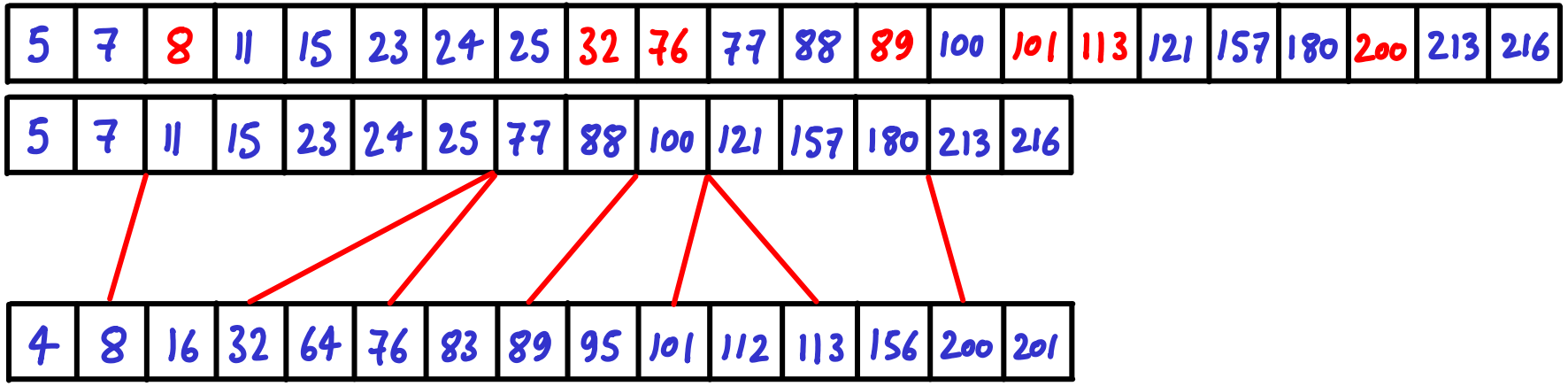

5 7 11 15 23 24 25 77 88 100 121 157 180 213 216

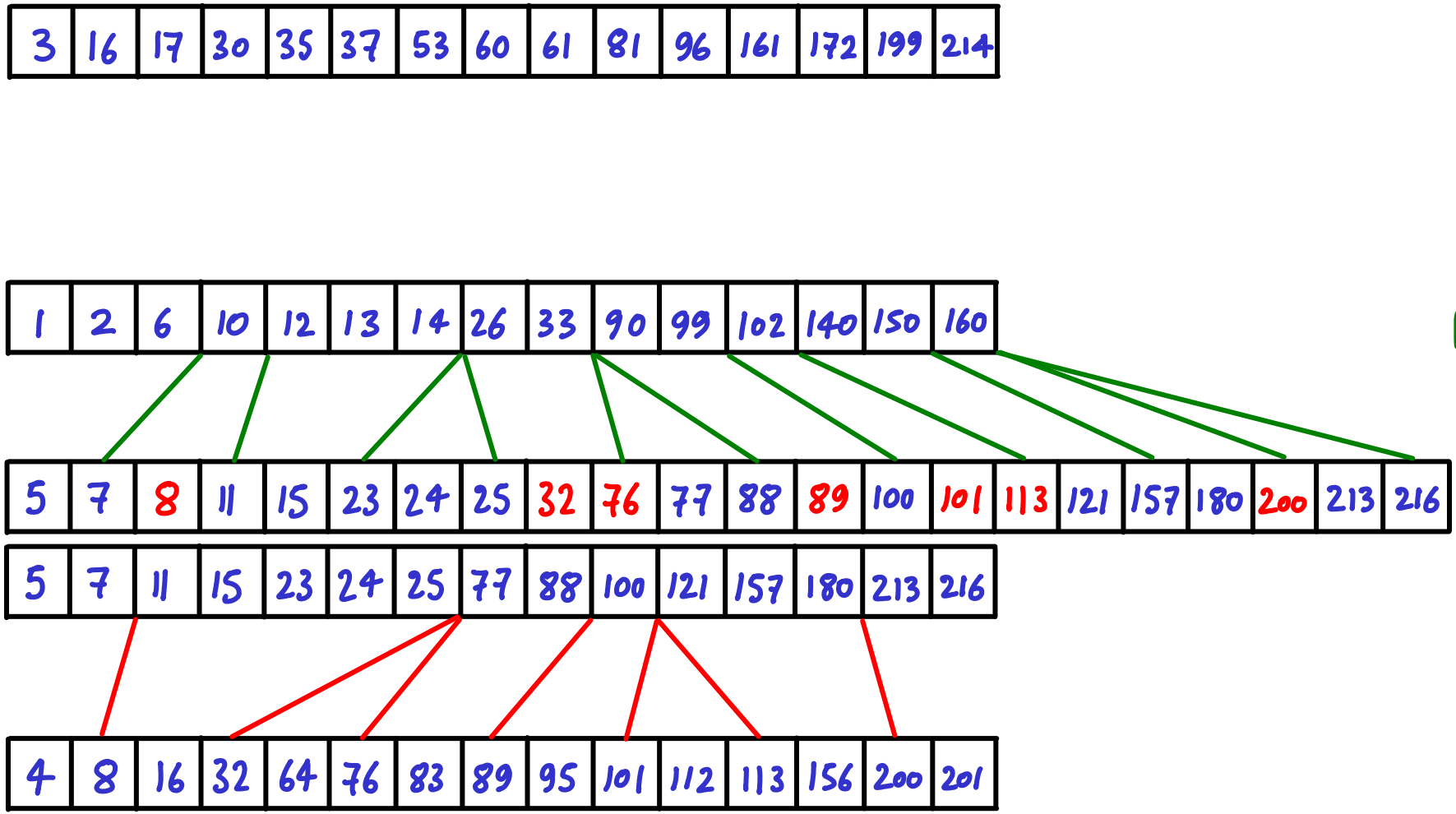

4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

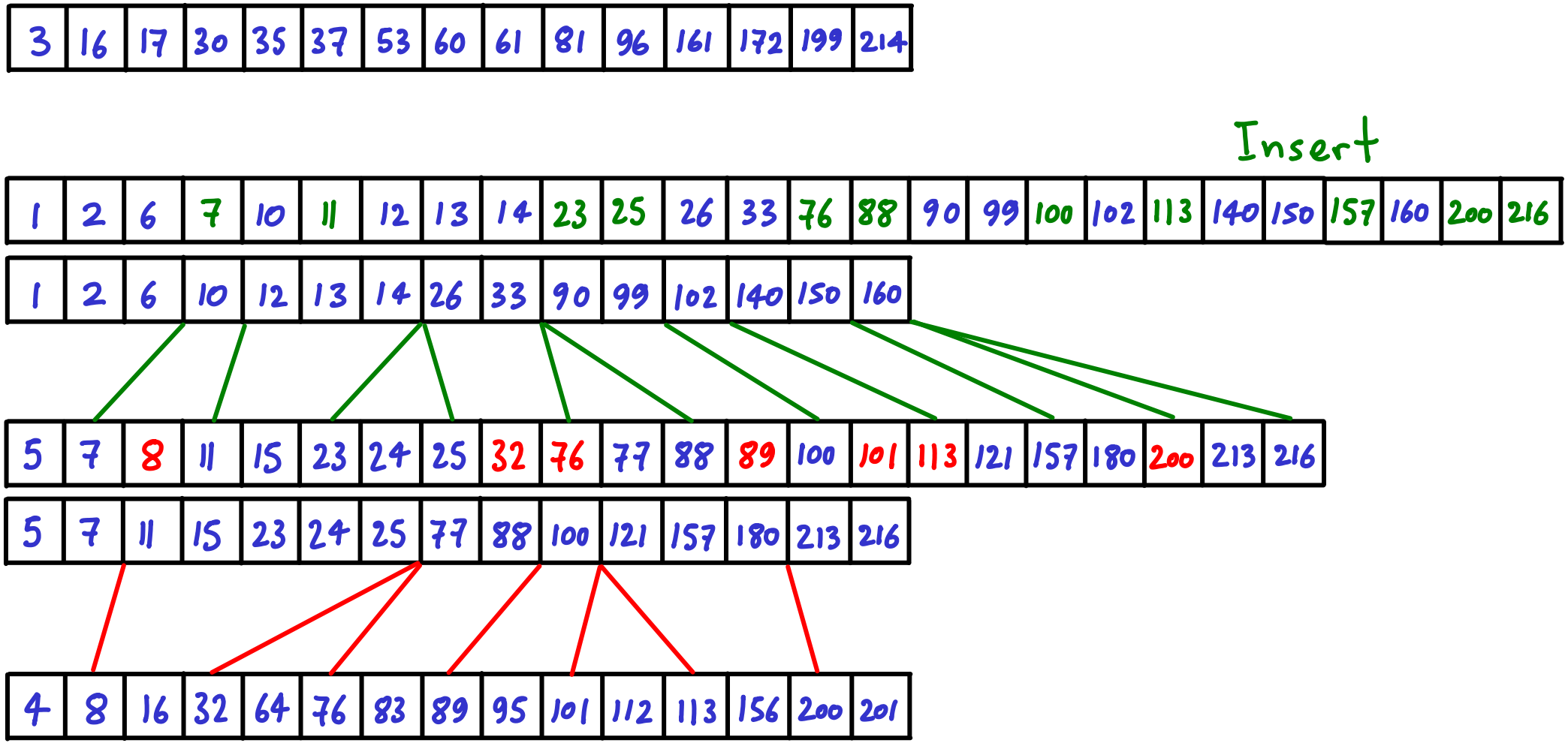

3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
list k-1
list k
Every other element from list k gets mapped to list k-1

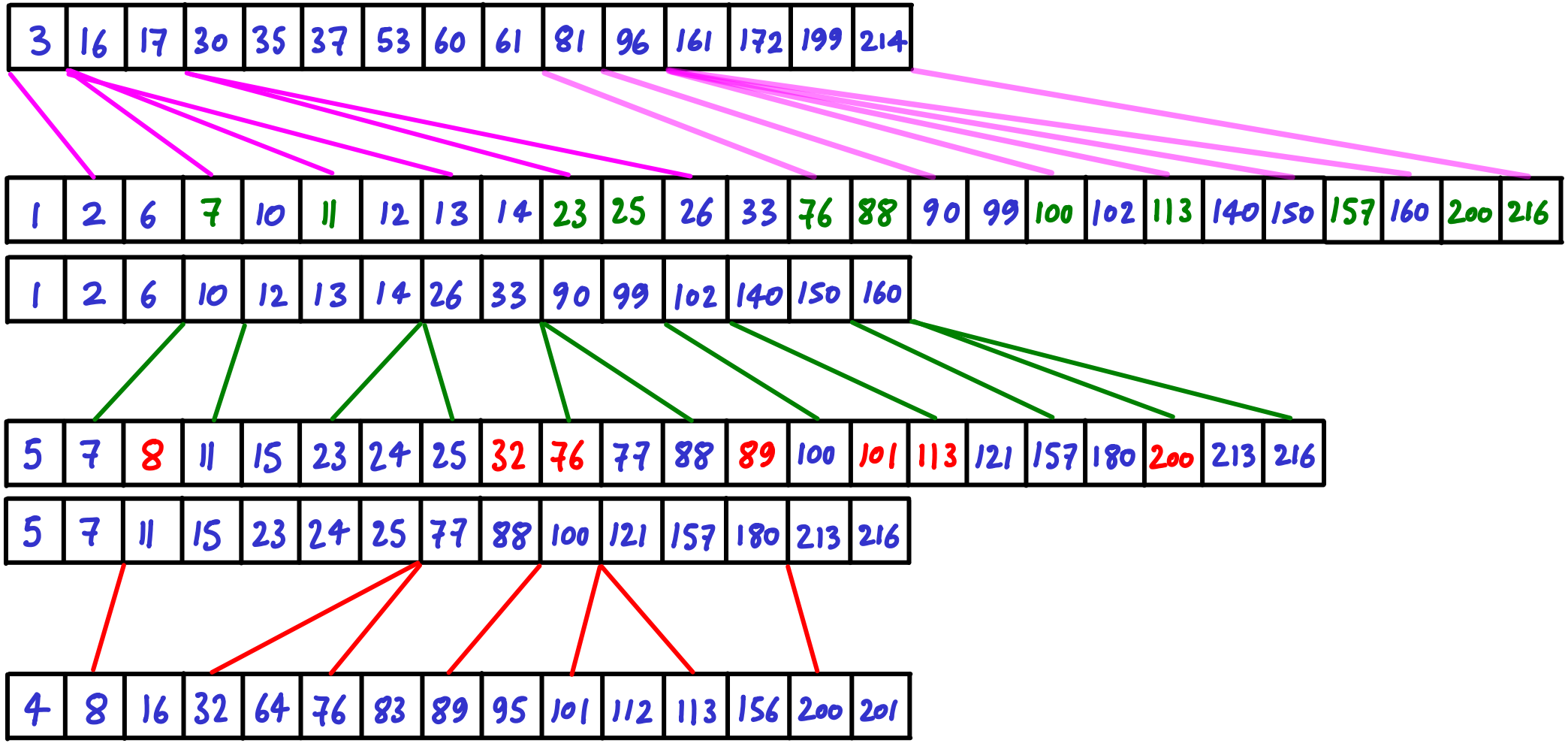

3 16 17 30 35 37 53 60 61 81 96 161 172 199 214

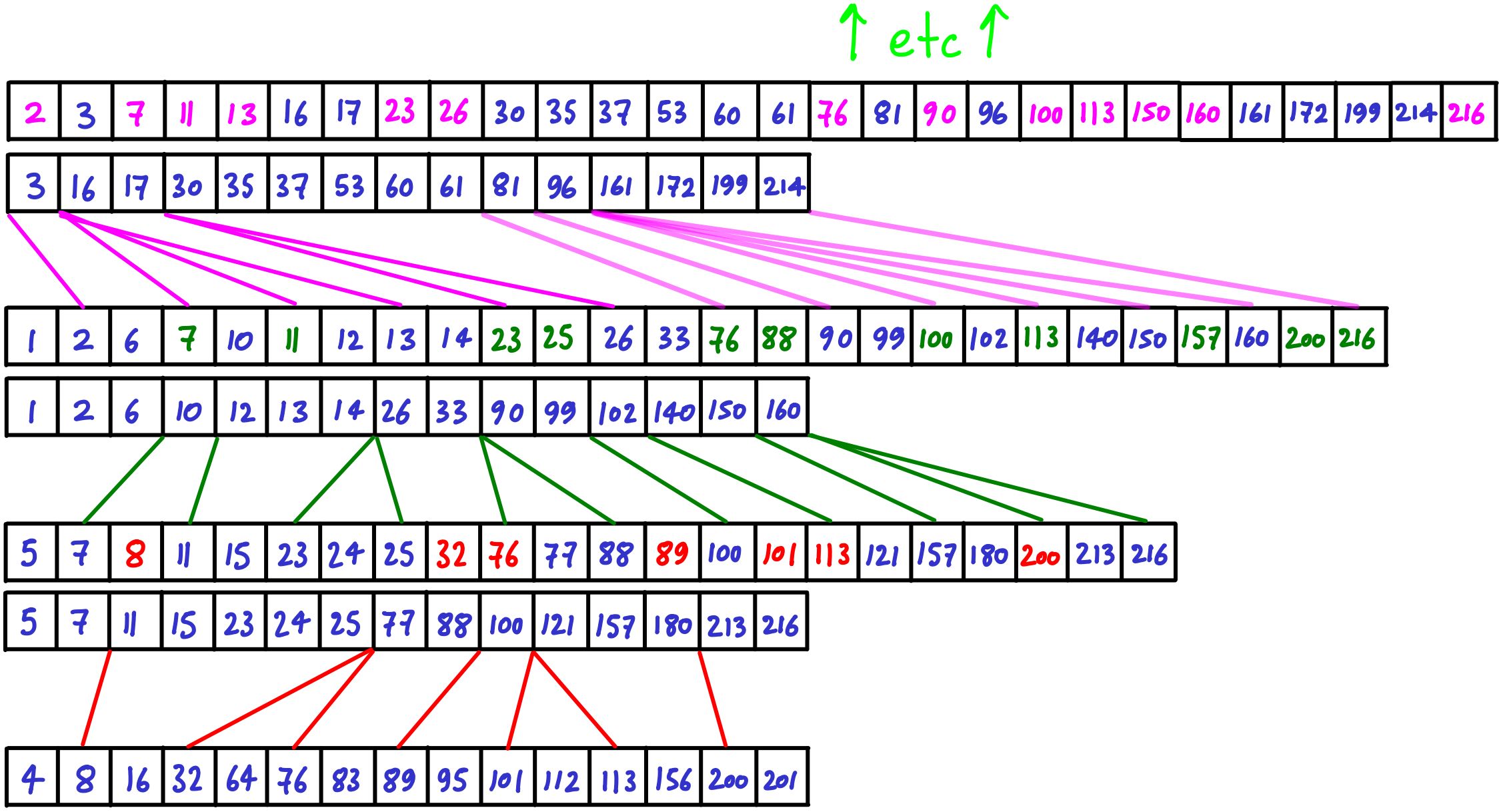

1 2 6 10 12 13 14 26 33 90 99 102 140 150 160

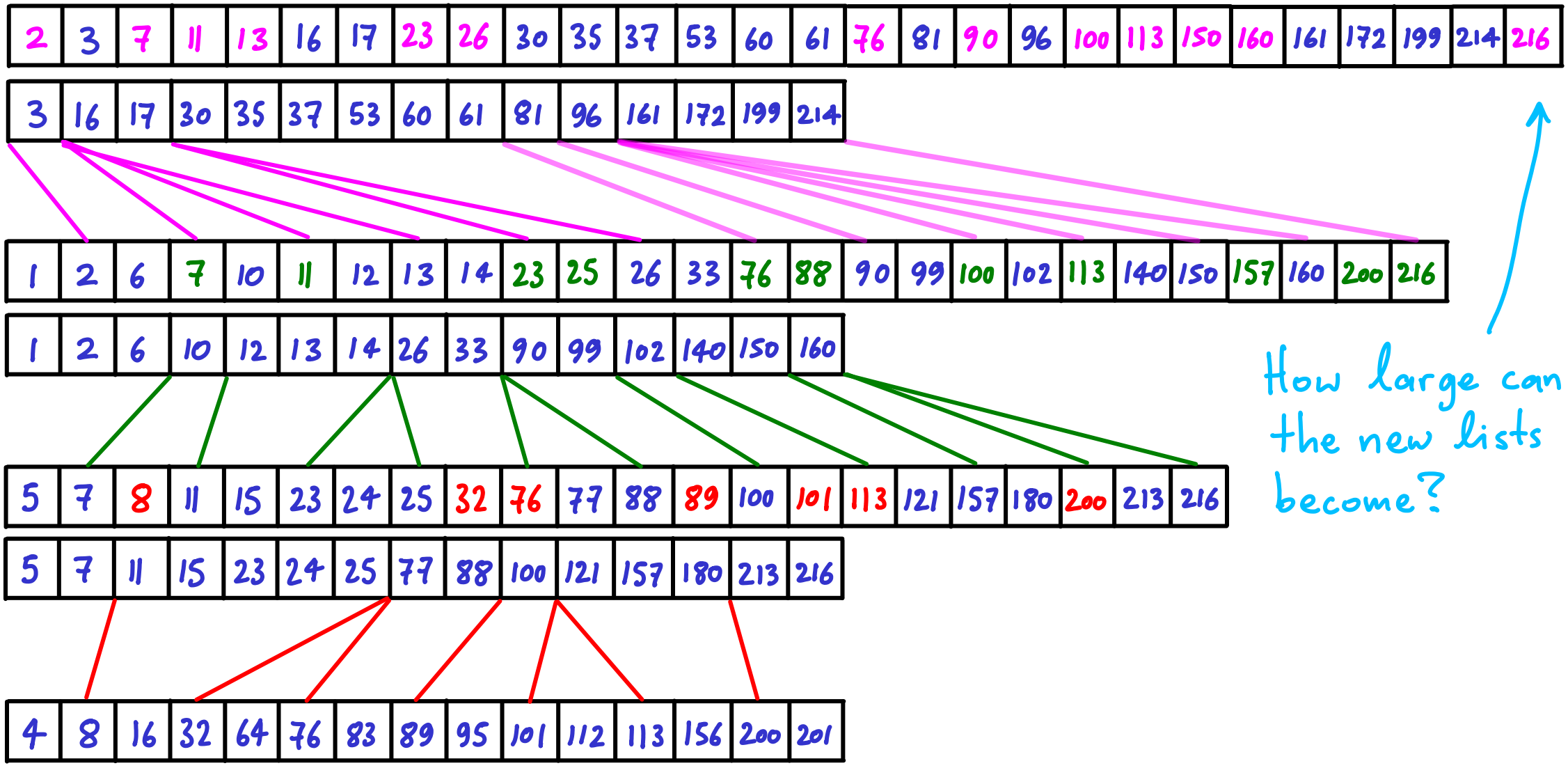

5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Insert mapped elements

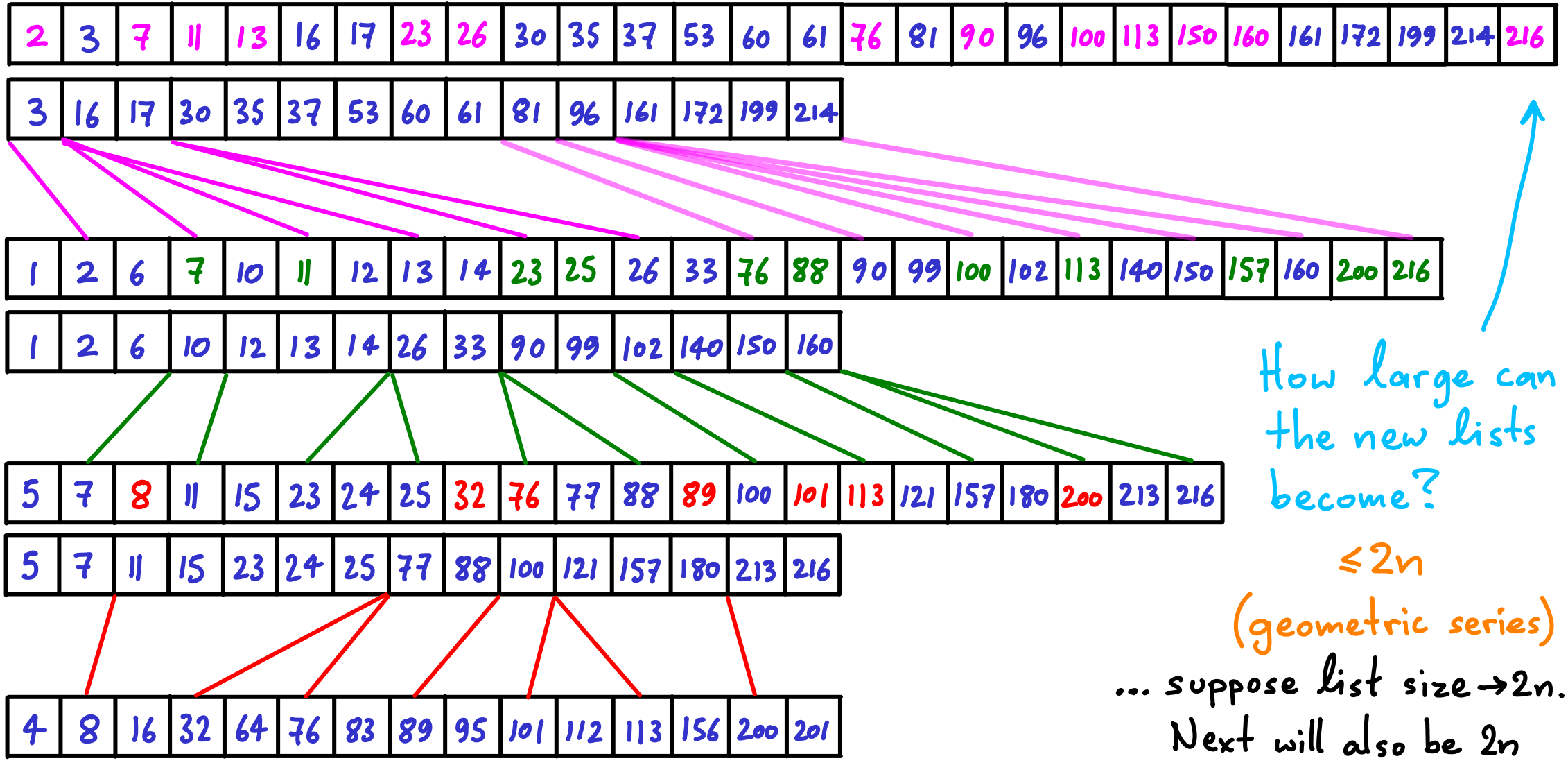

3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Map from list k-1 to list k-2

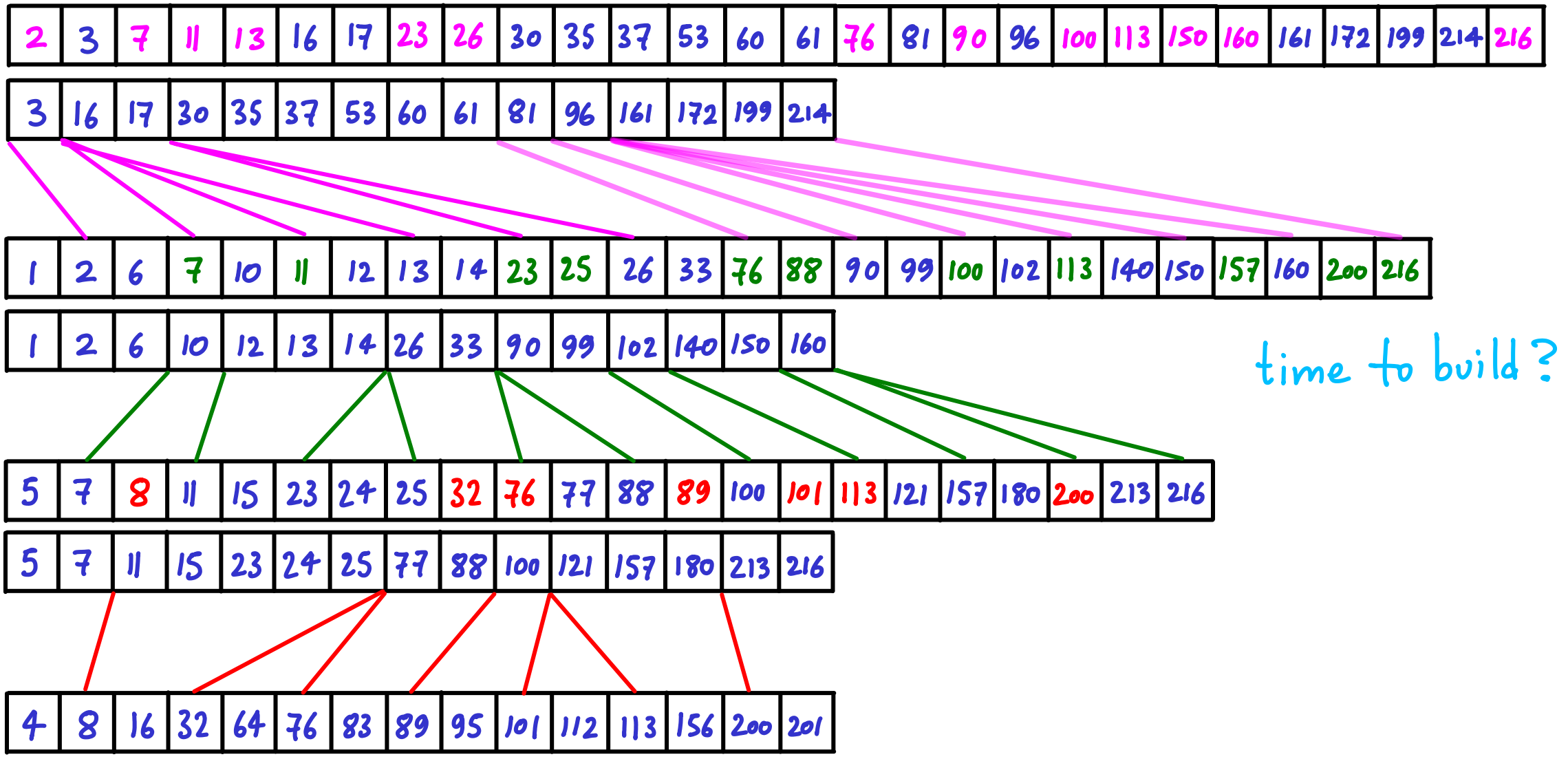

3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
Insert
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

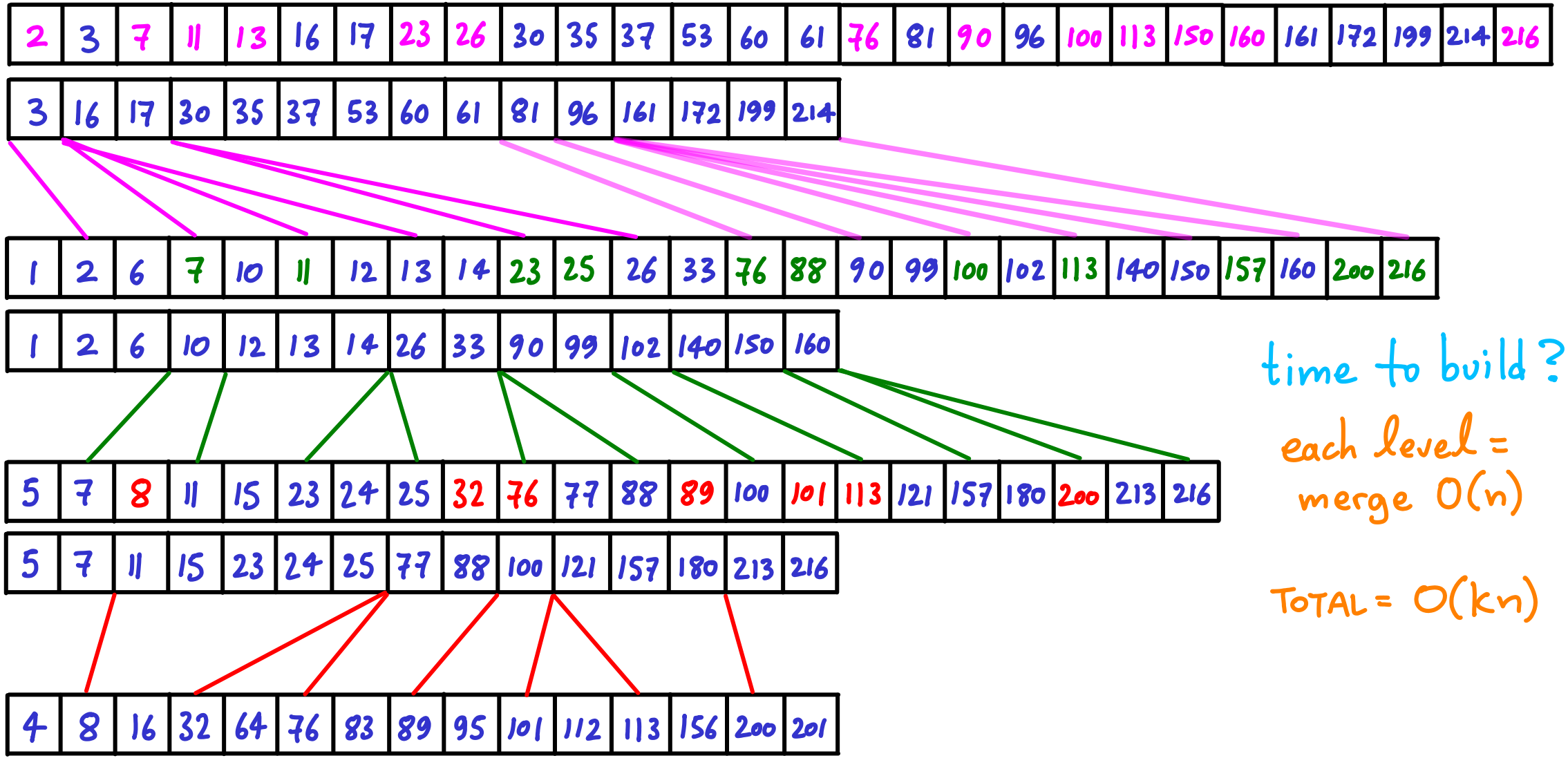

3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201


↑ etc ↑
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201


2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
How large can the new lists become?


2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
How large can the new lists become?
≤2n
(geometric series)
... suppose list size→2n.
Next will also be 2n

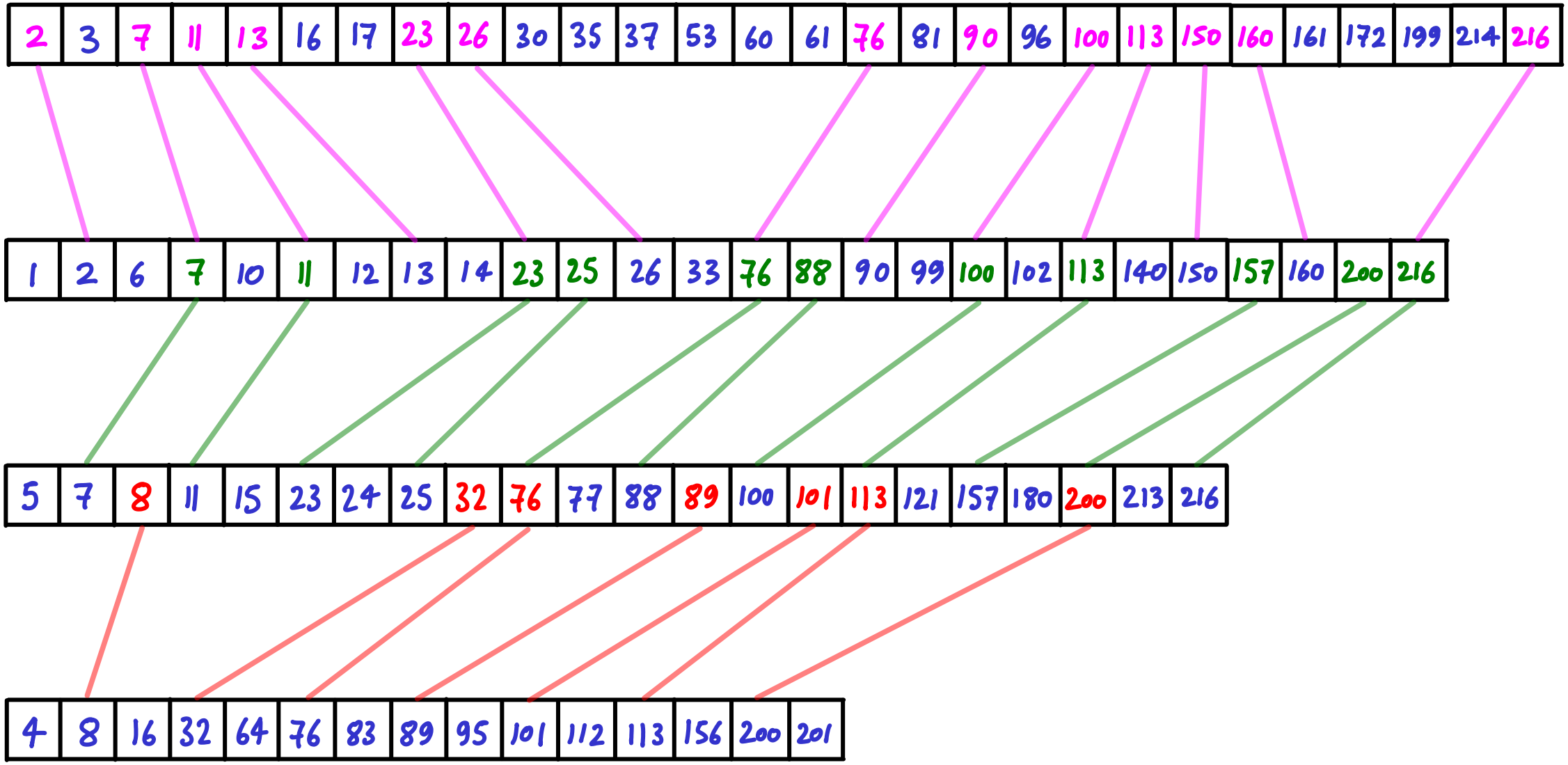

2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
3 16 17 30 35 37 53 60 61 81 96 161 172 199 214
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 6 10 12 13 14 26 33 90 99 102 140 150 160
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
5 7 11 15 23 24 25 77 88 100 121 157 180 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
time to build?

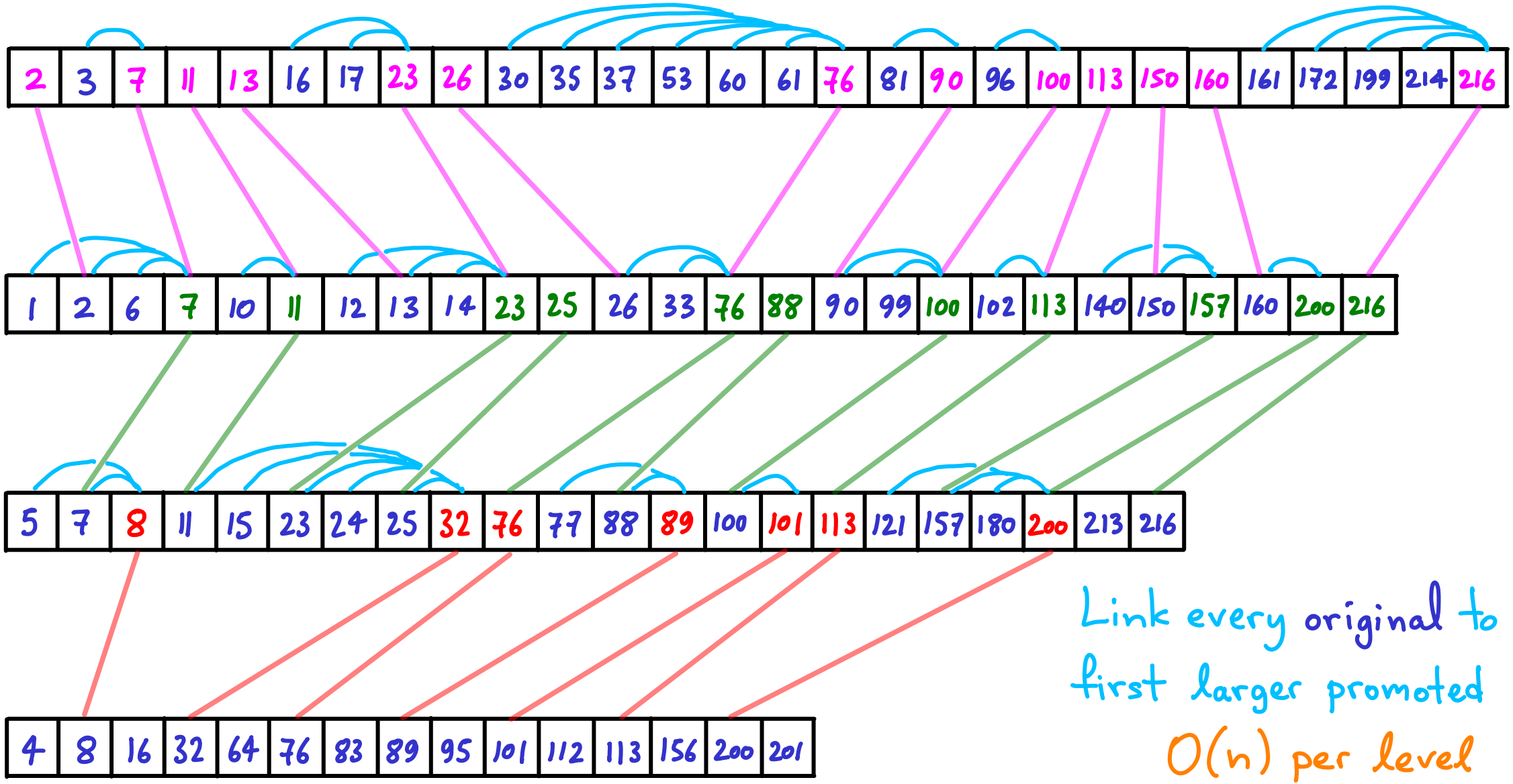


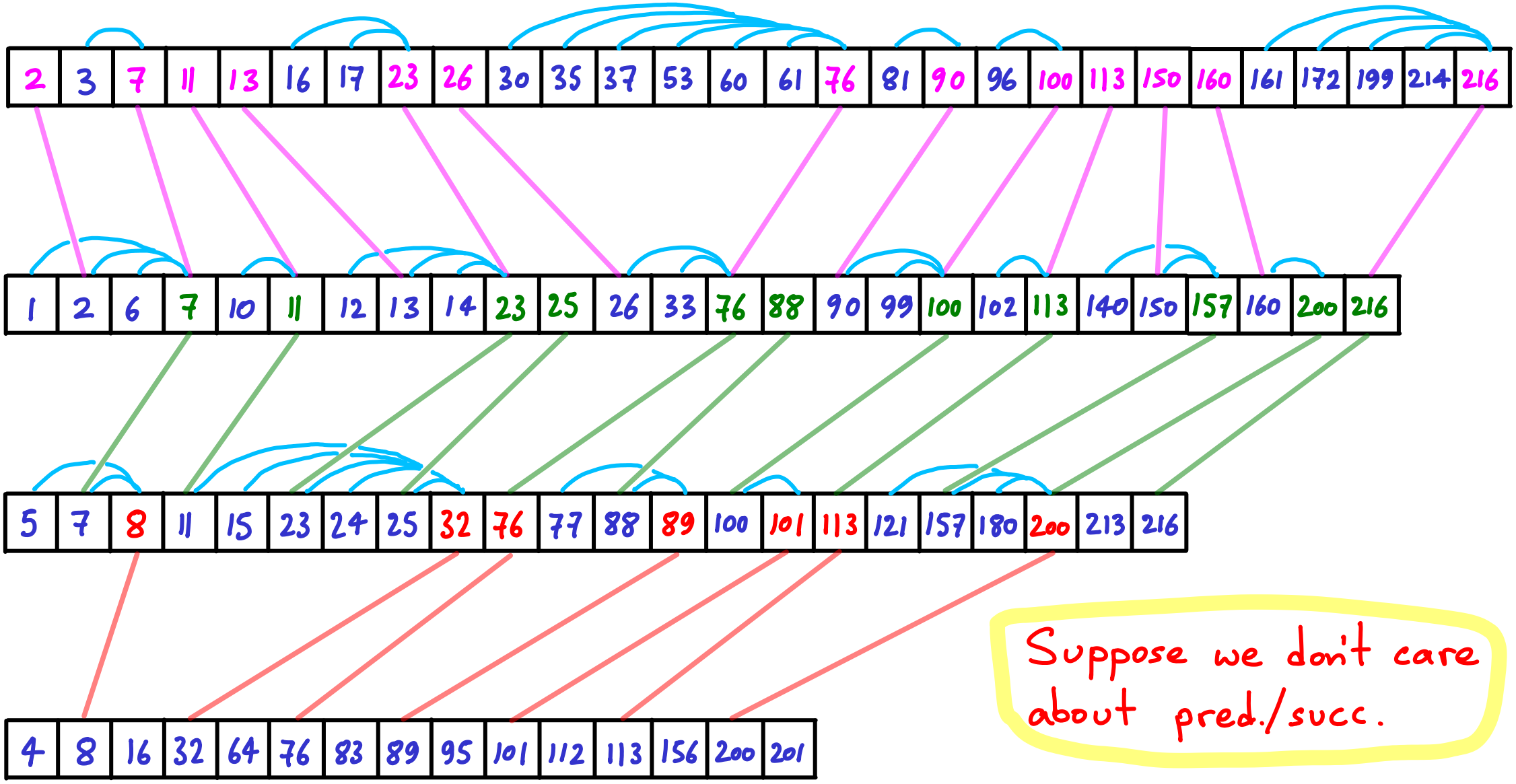

time to build?

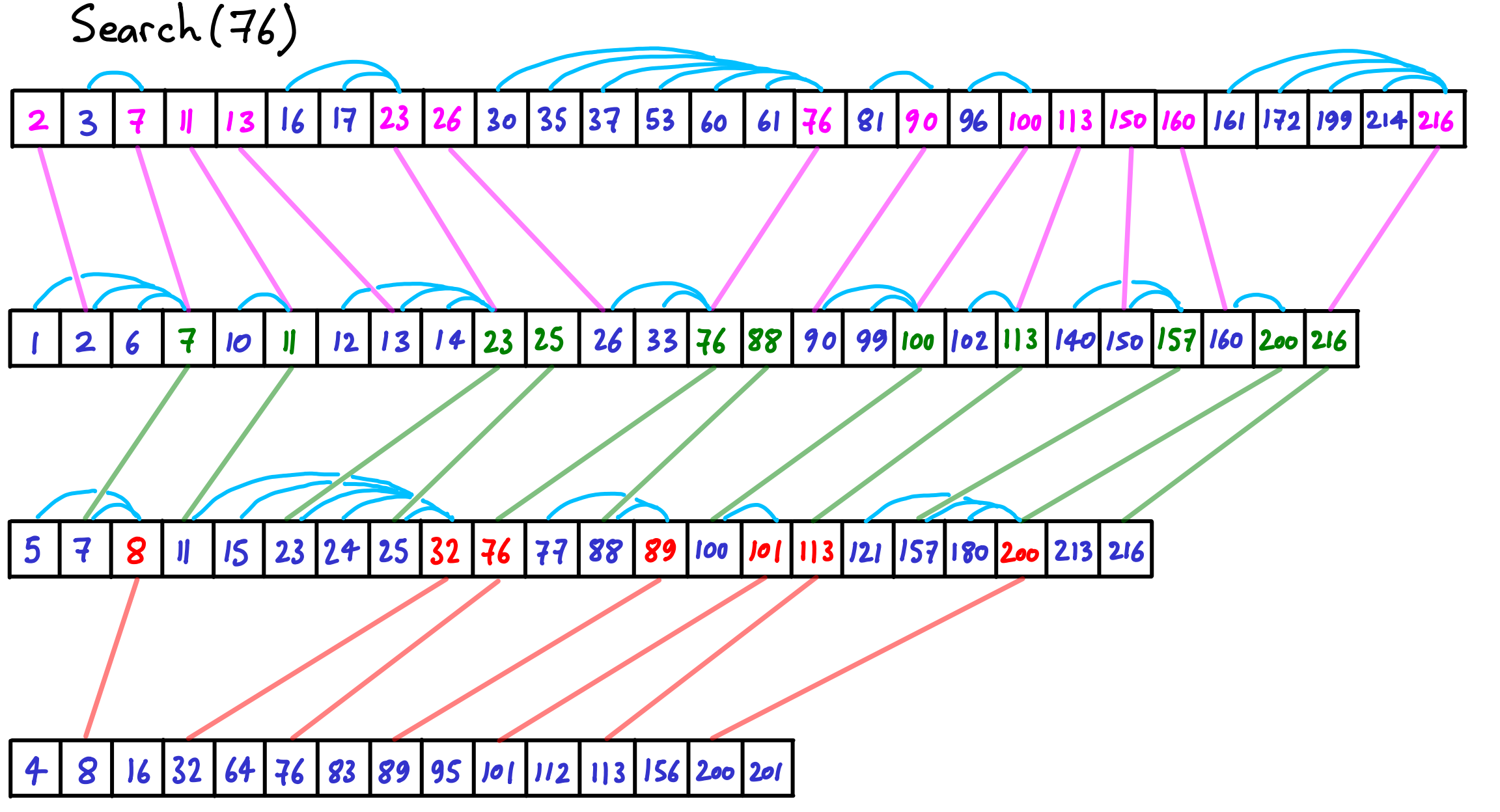
each level = merge O(n)

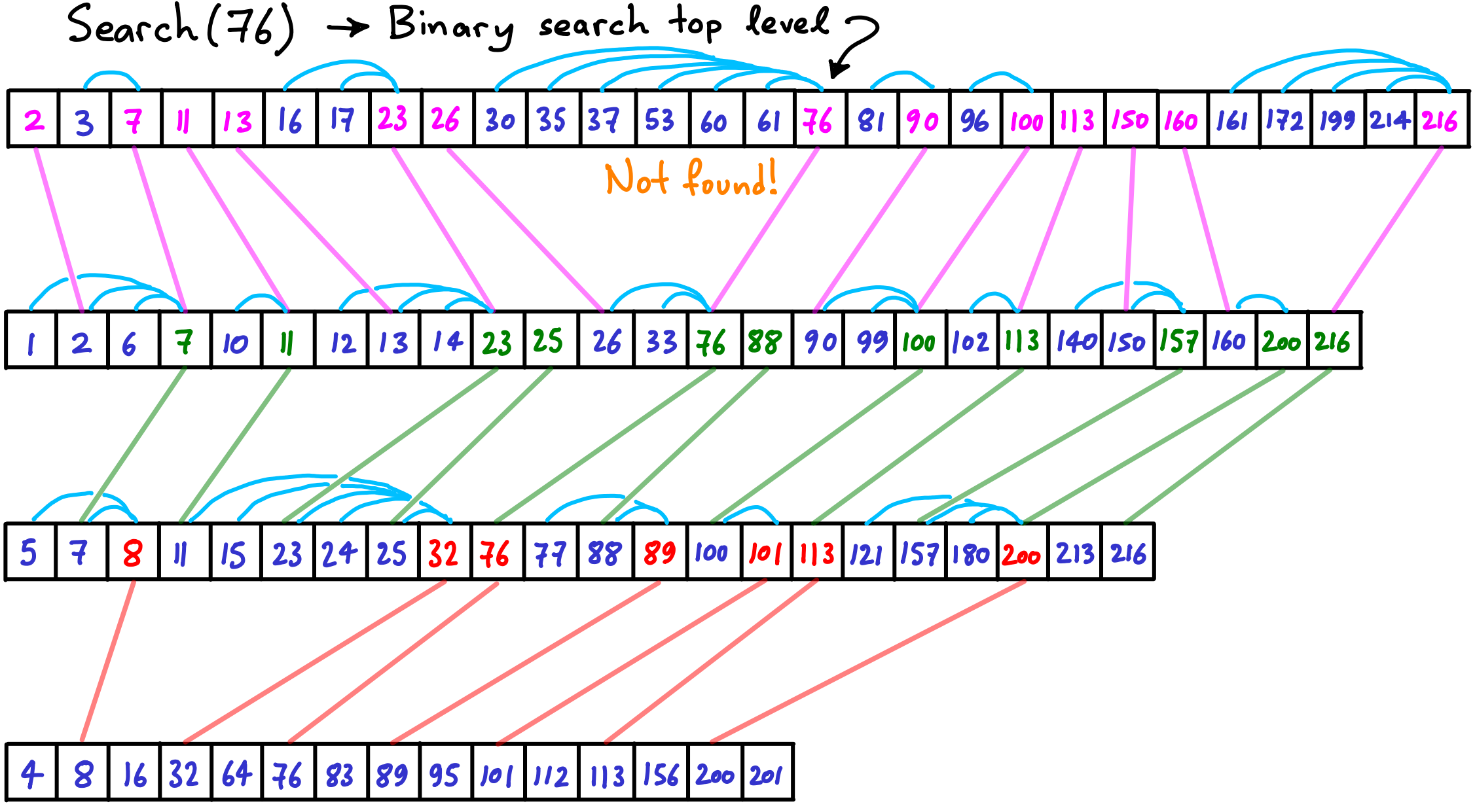
TOTAL = O(kn)

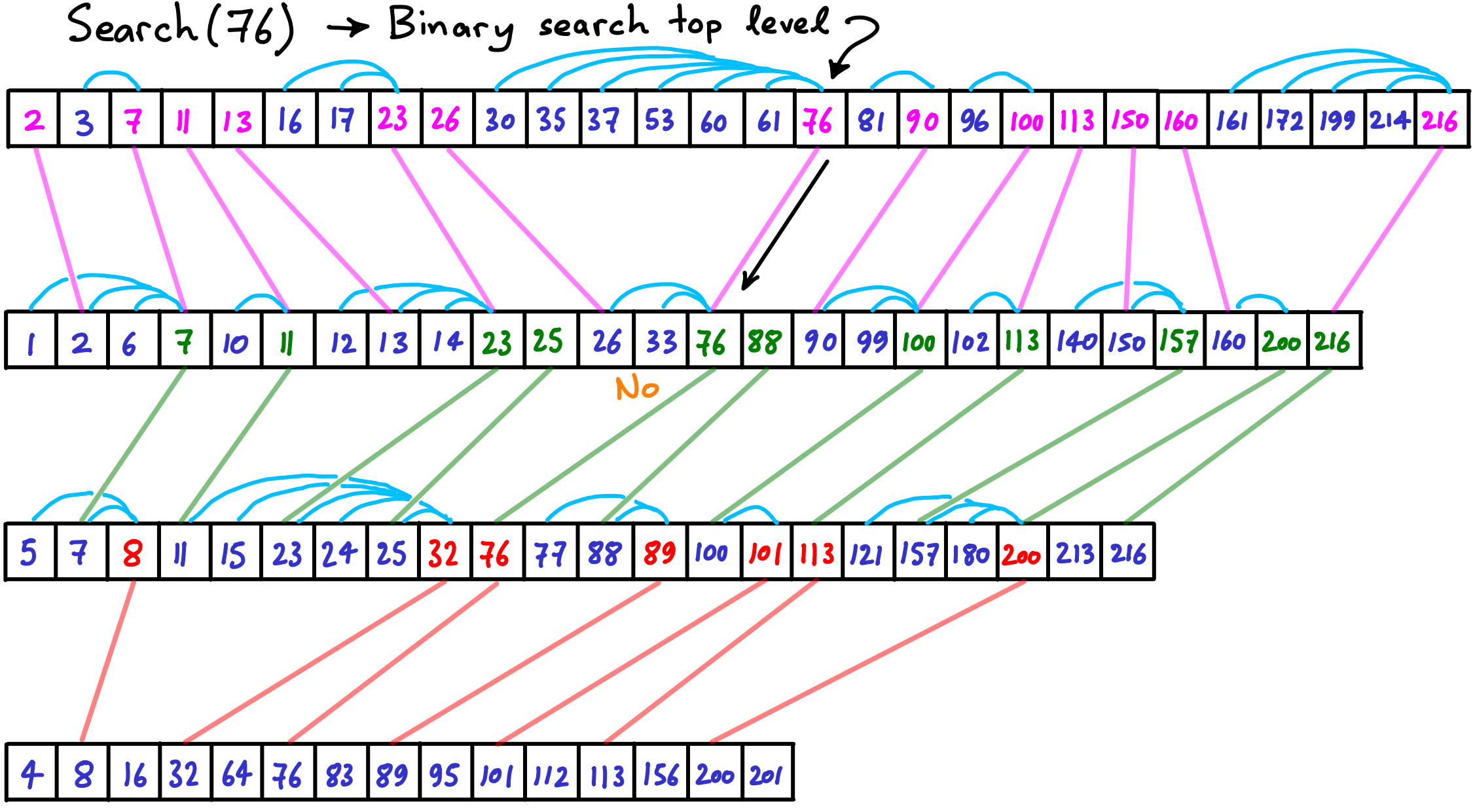

2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

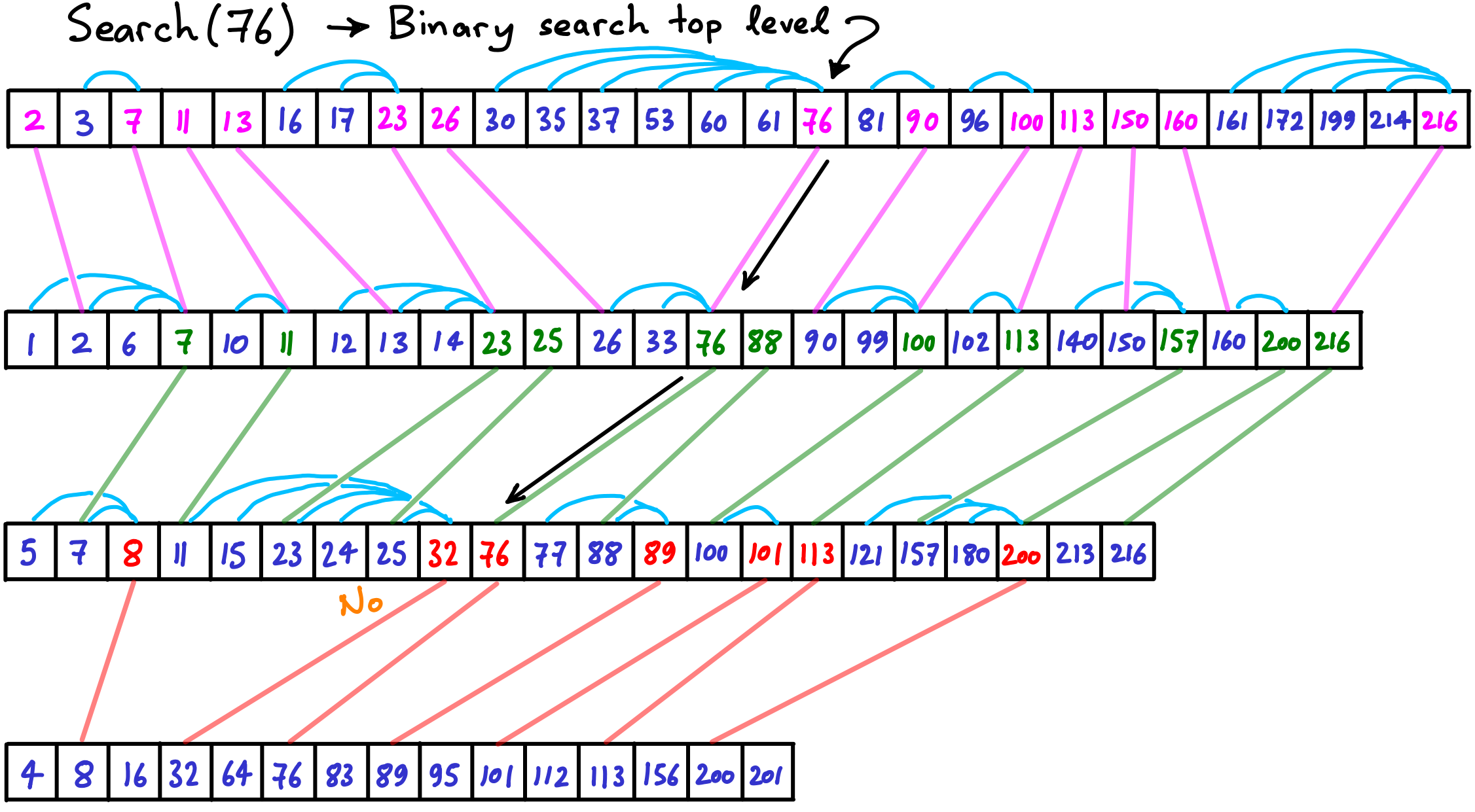

2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Link every original to first larger promoted
O(n) per level

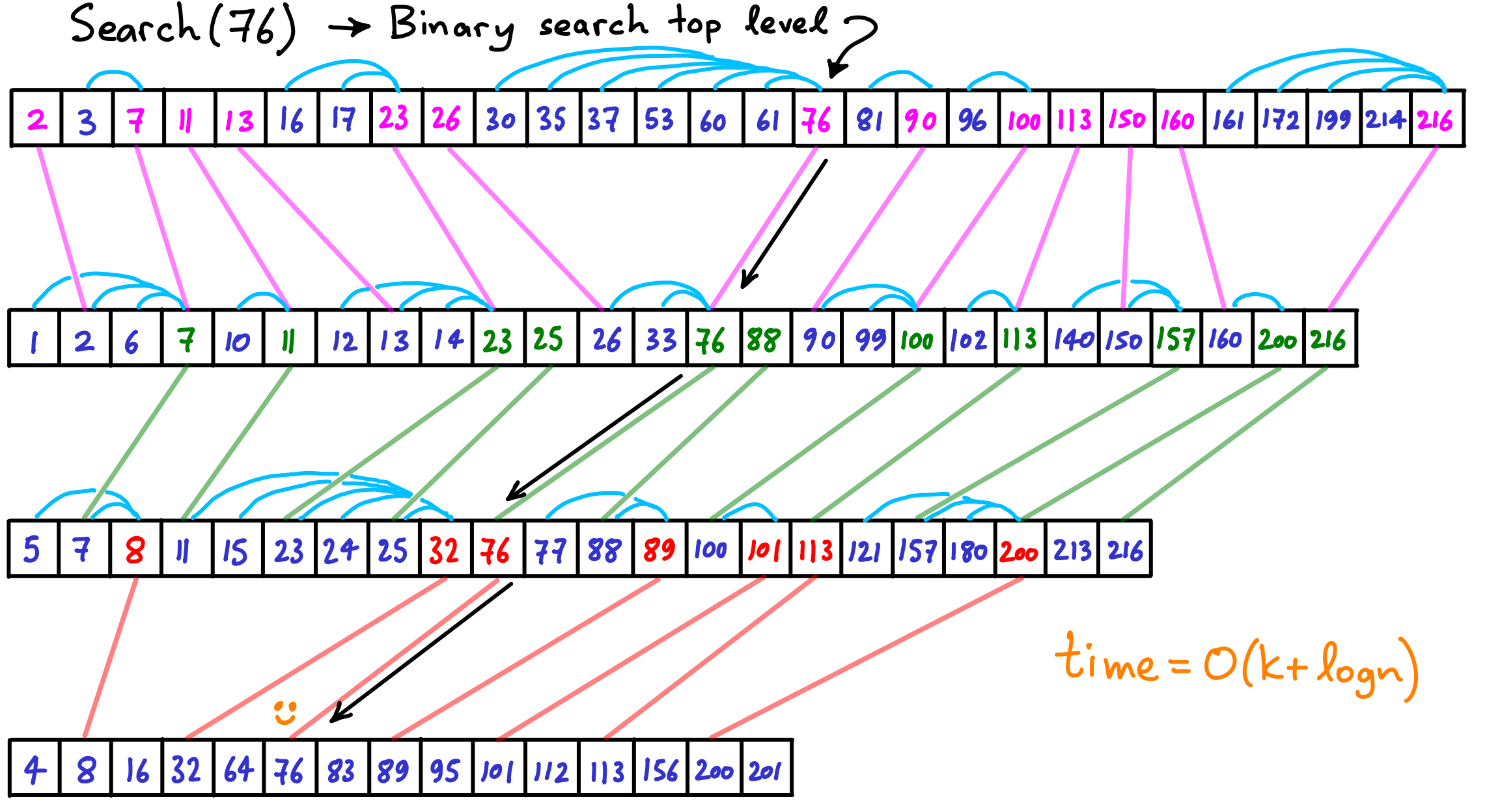

2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Suppose we don't care about pred./succ.

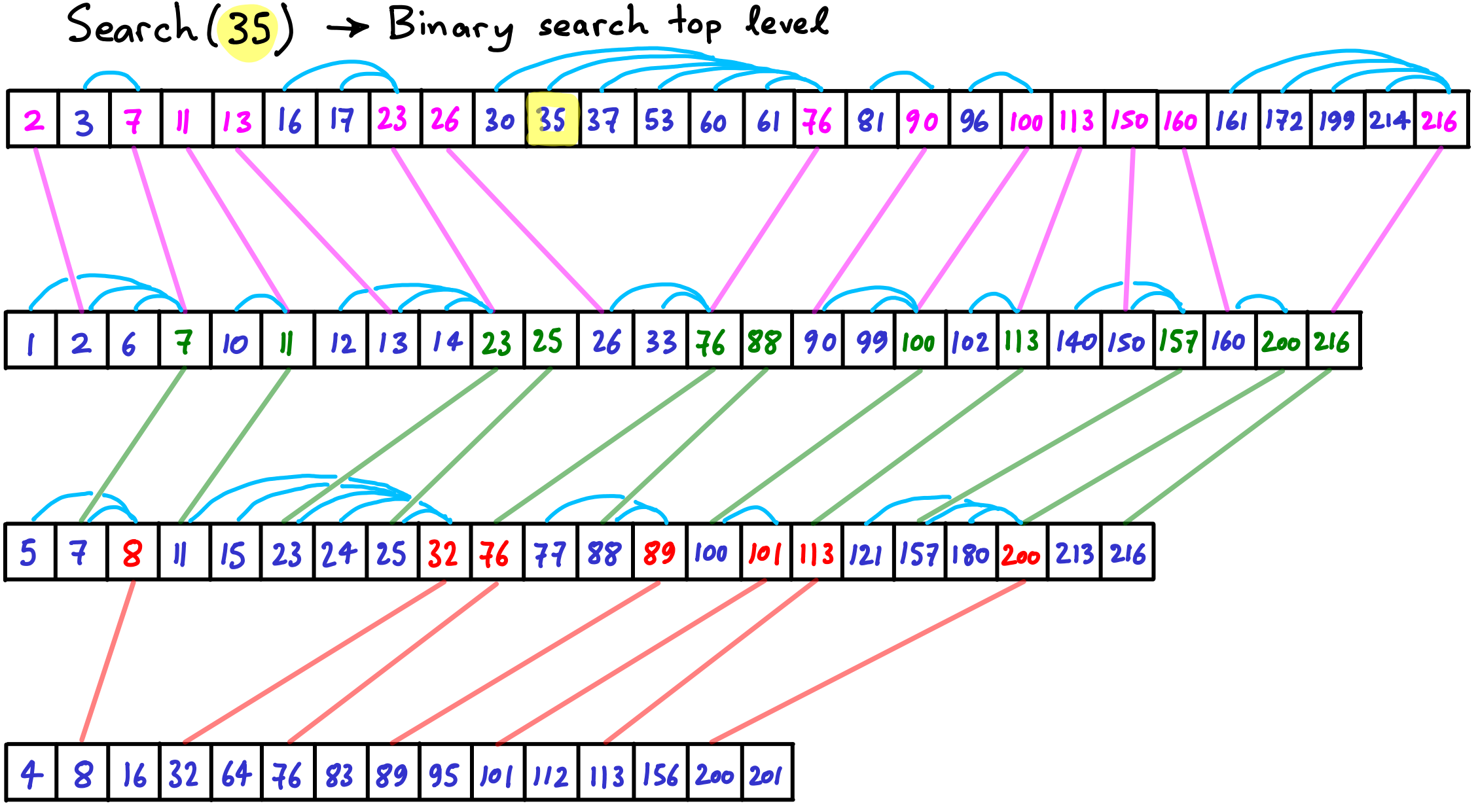

Search(76)
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

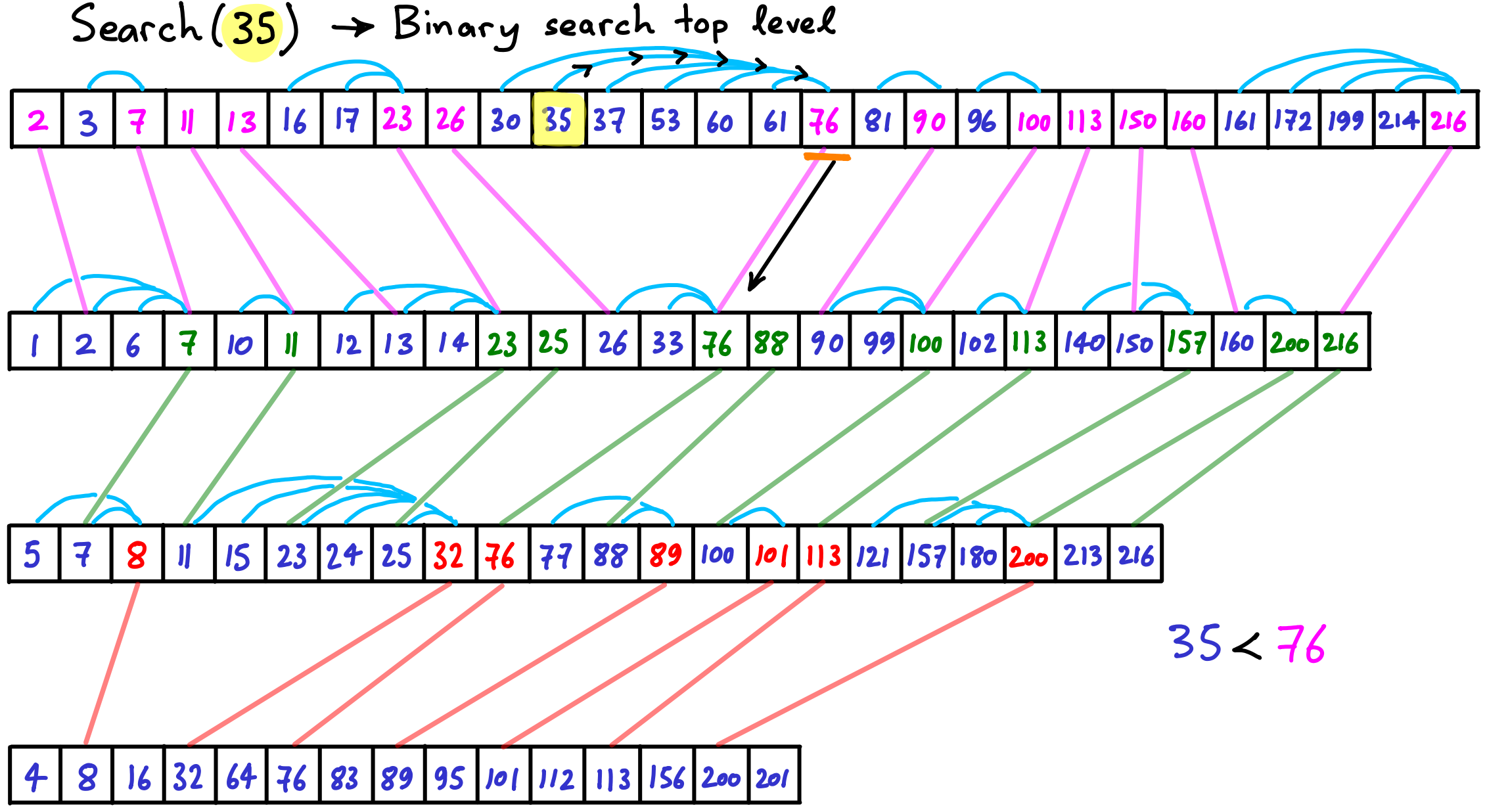

Search(76) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
Not found!
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

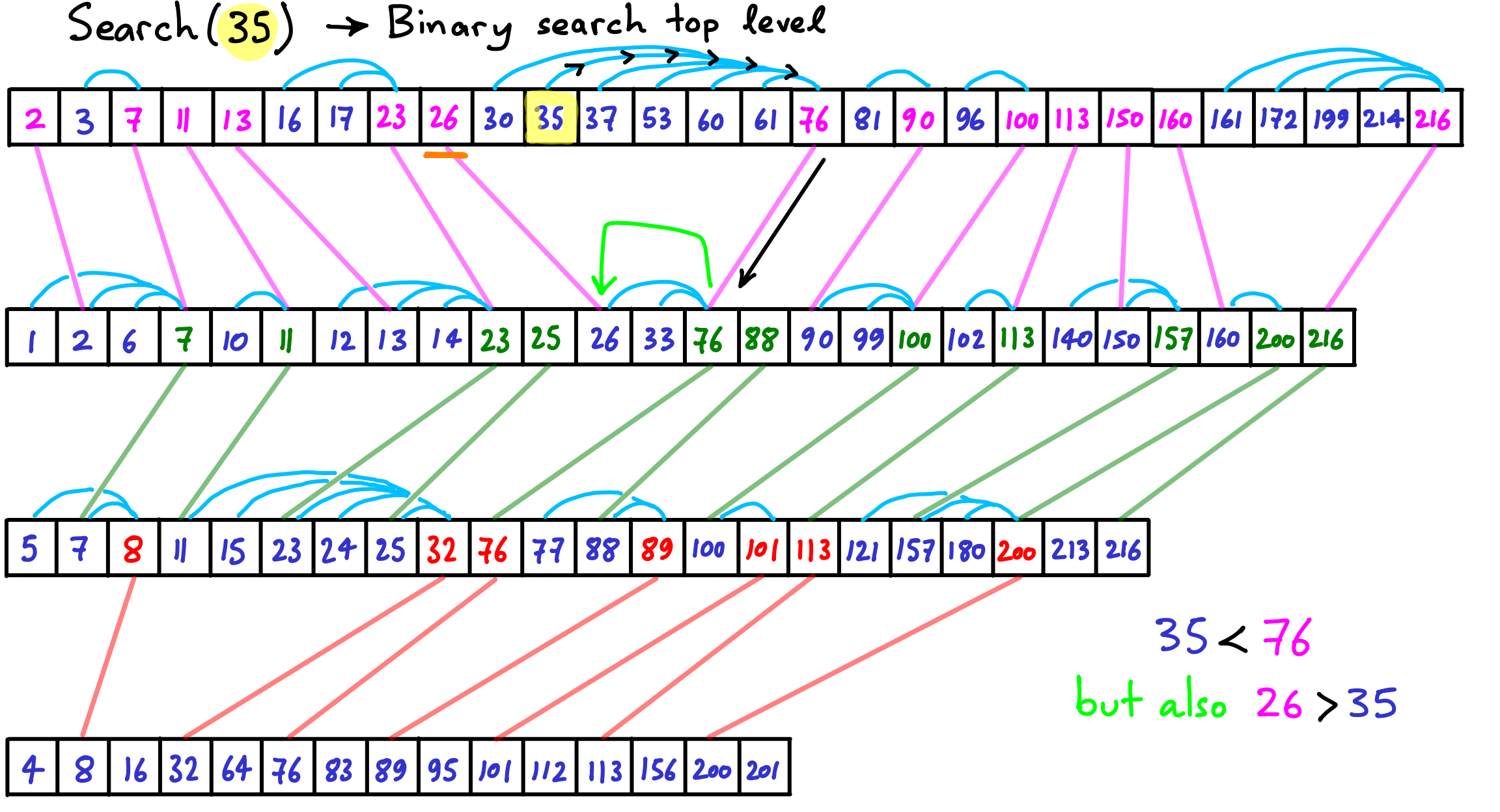

Search(76) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
No
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

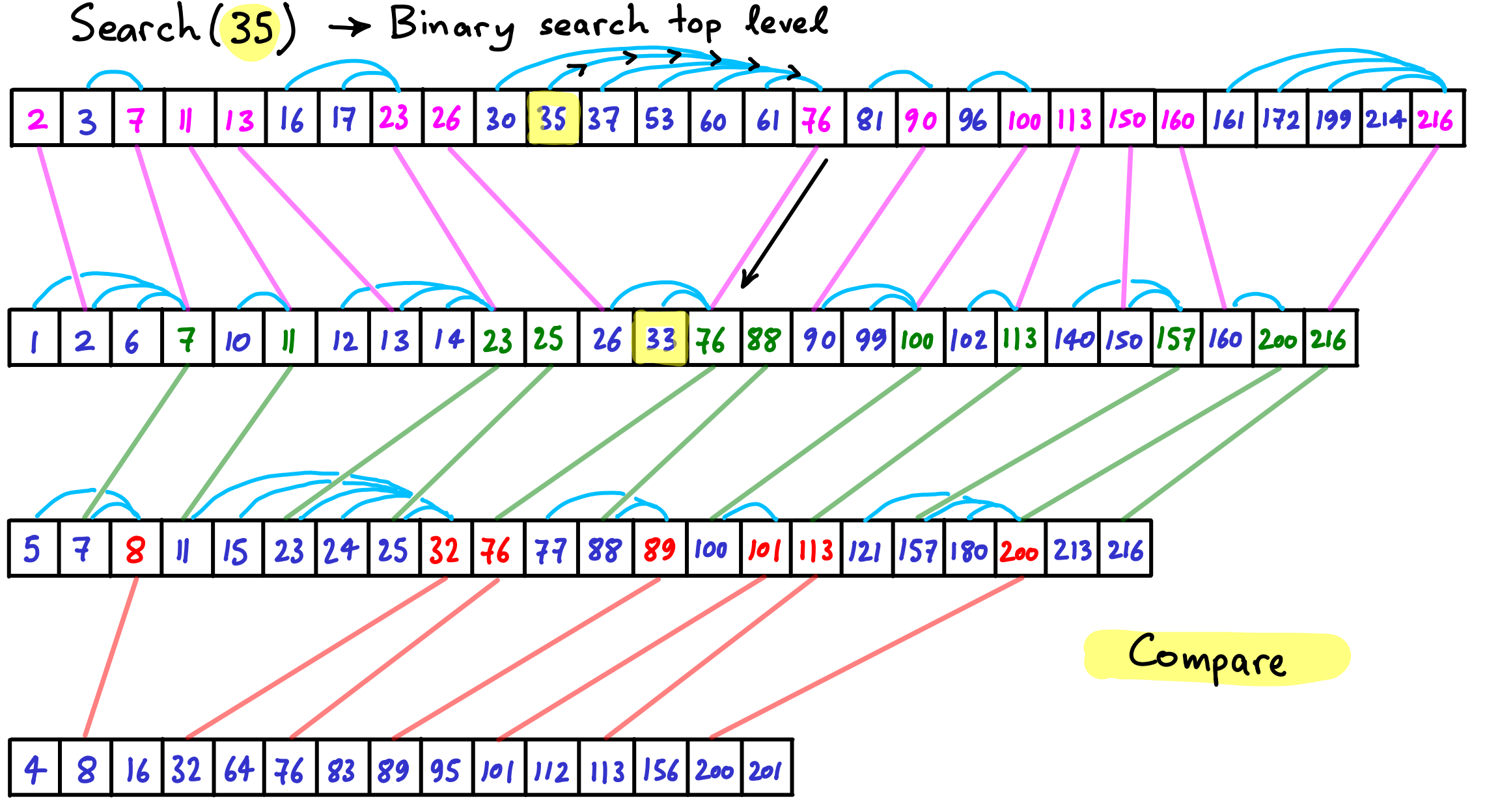

Search(76) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
No
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

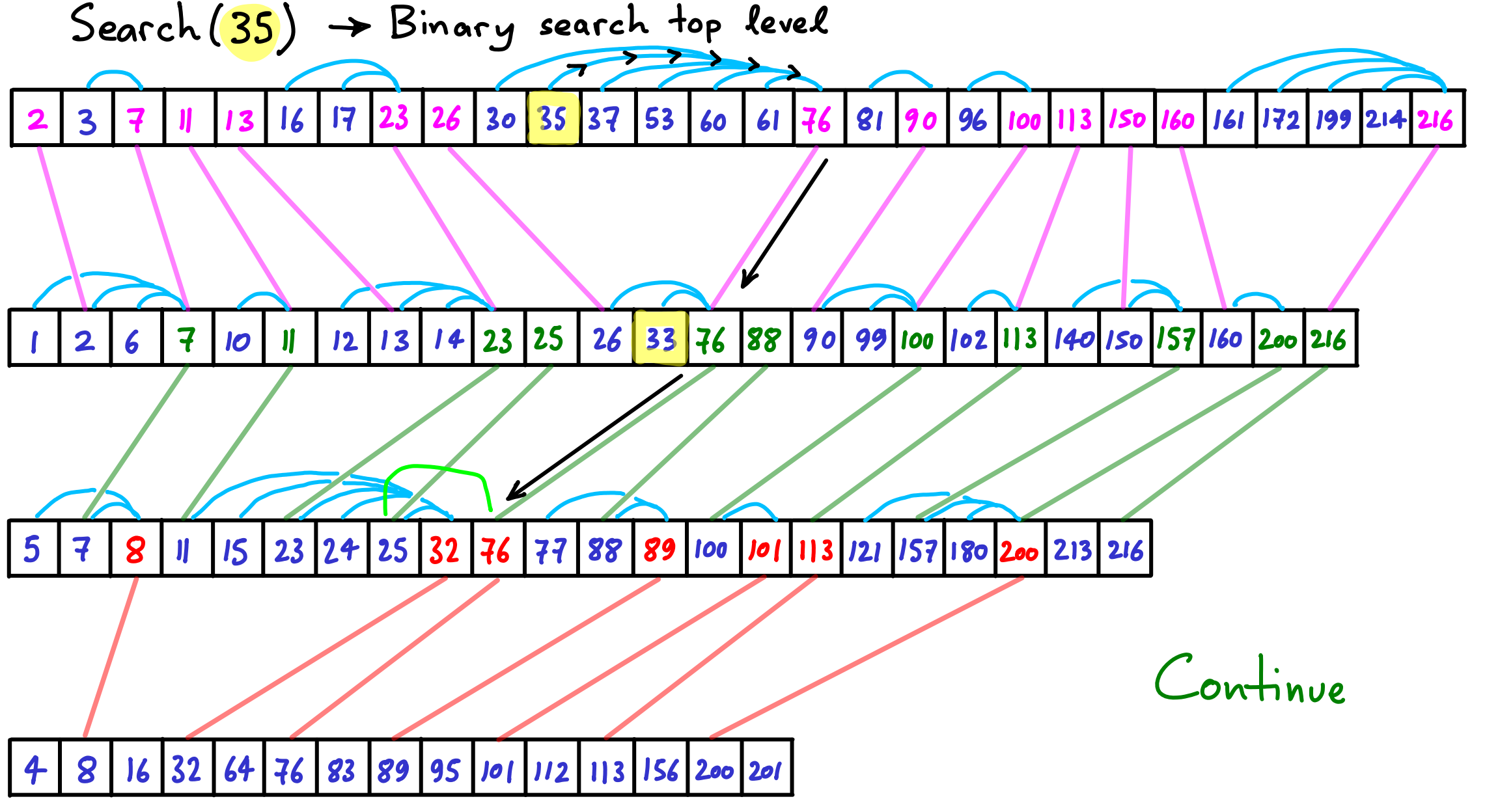

Search(76) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
time = O(k+logn)

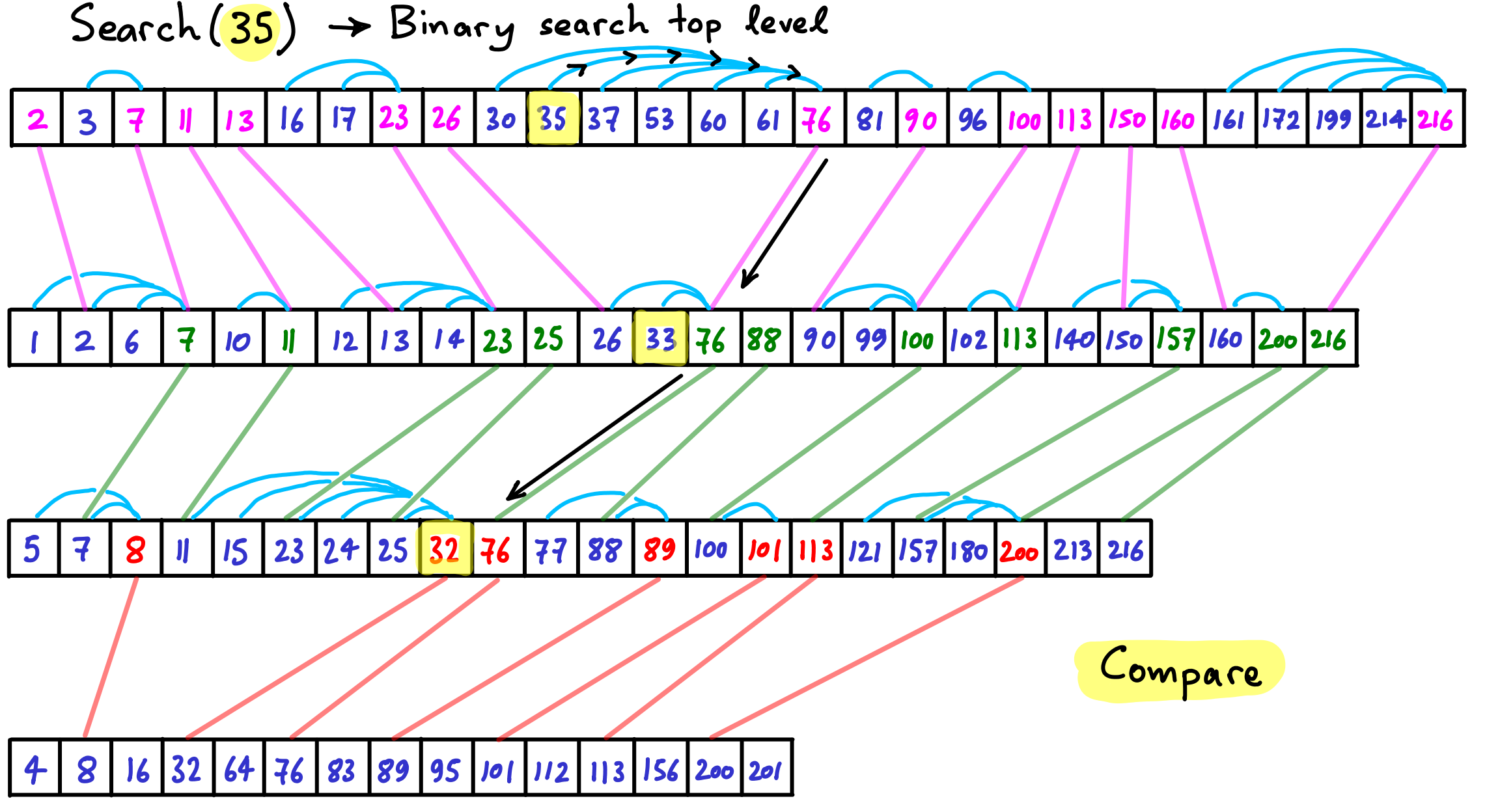

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201

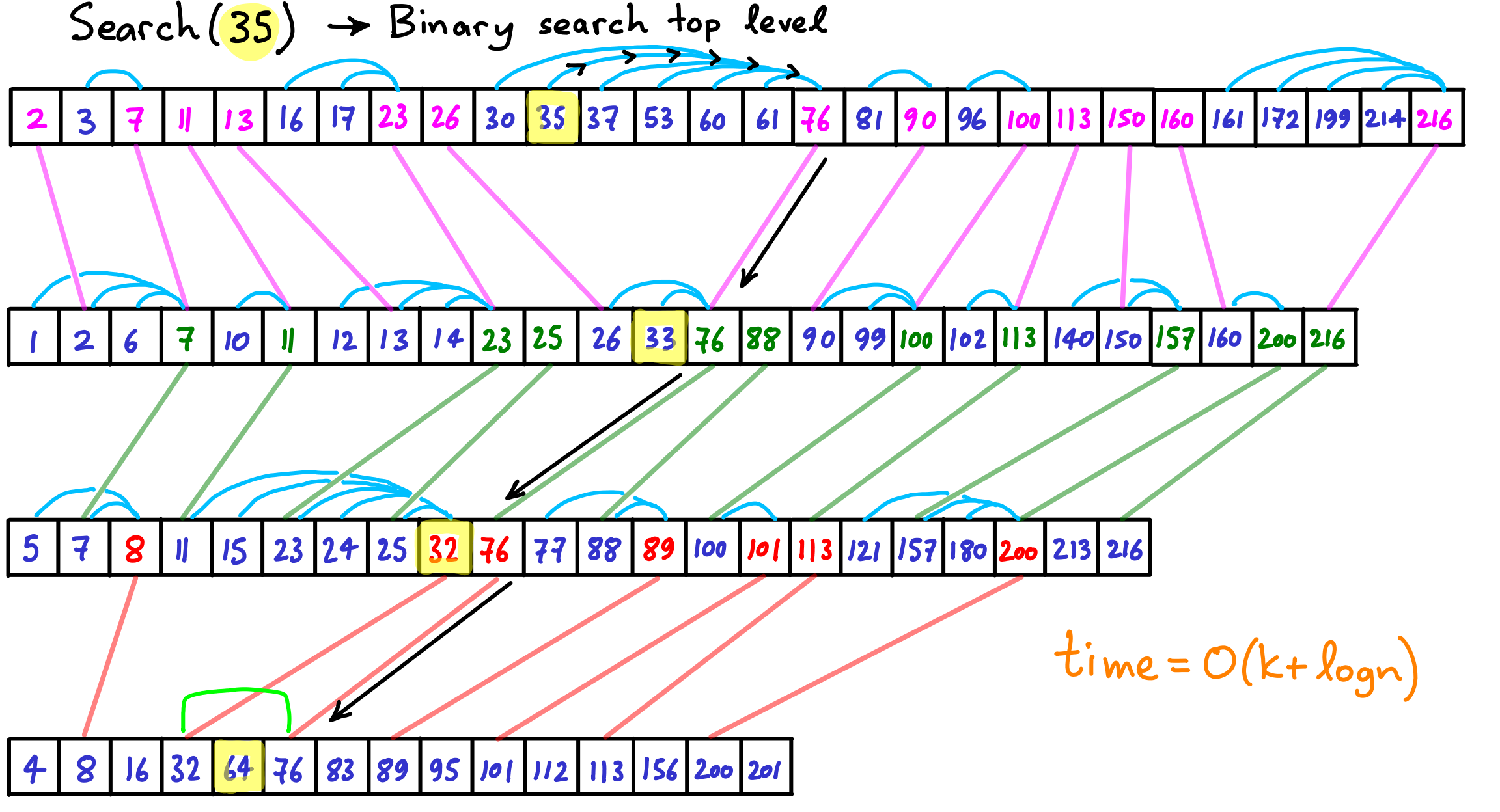

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
35 < 76

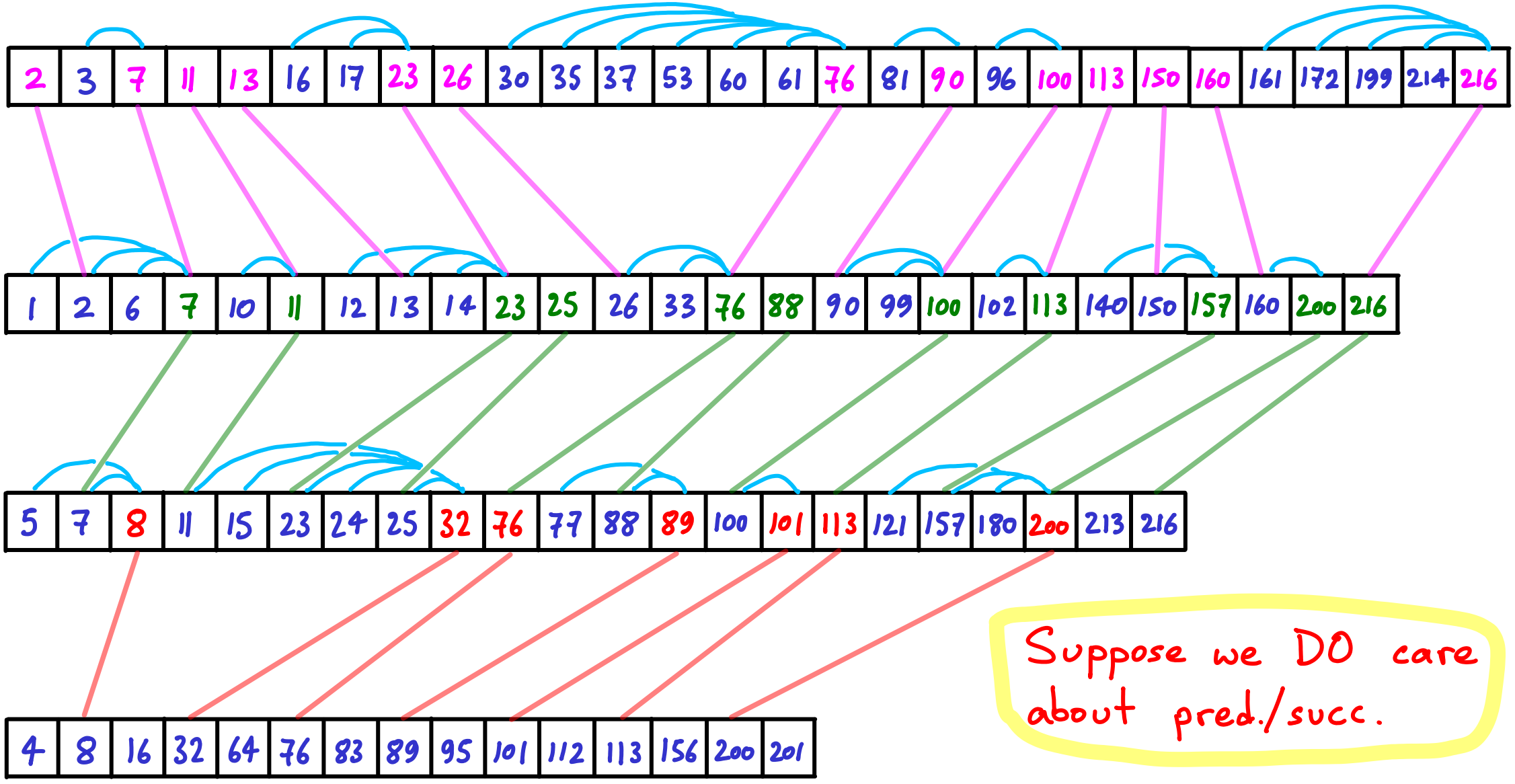

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
35 < 76
but also 26 > 35

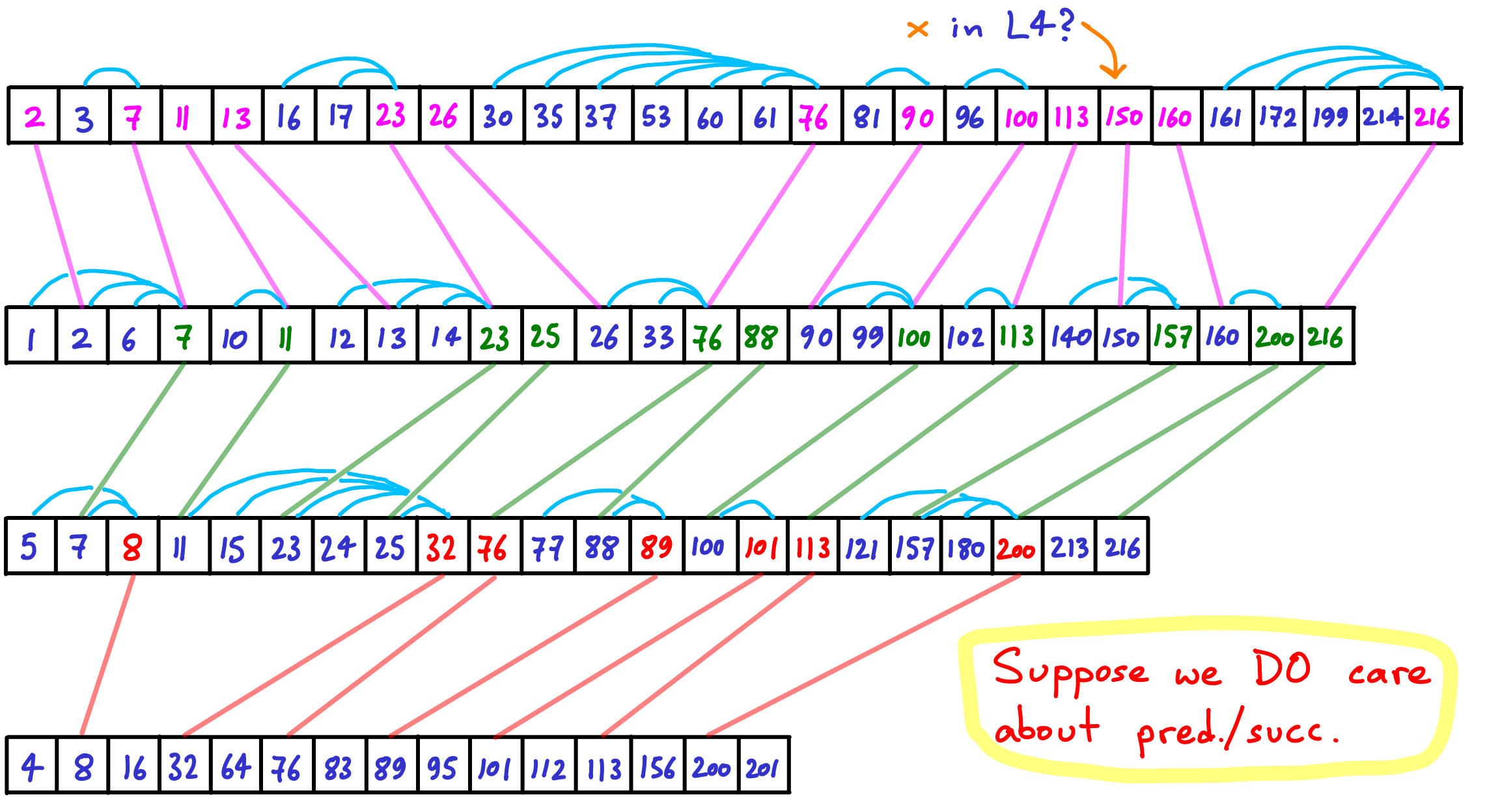

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Compare

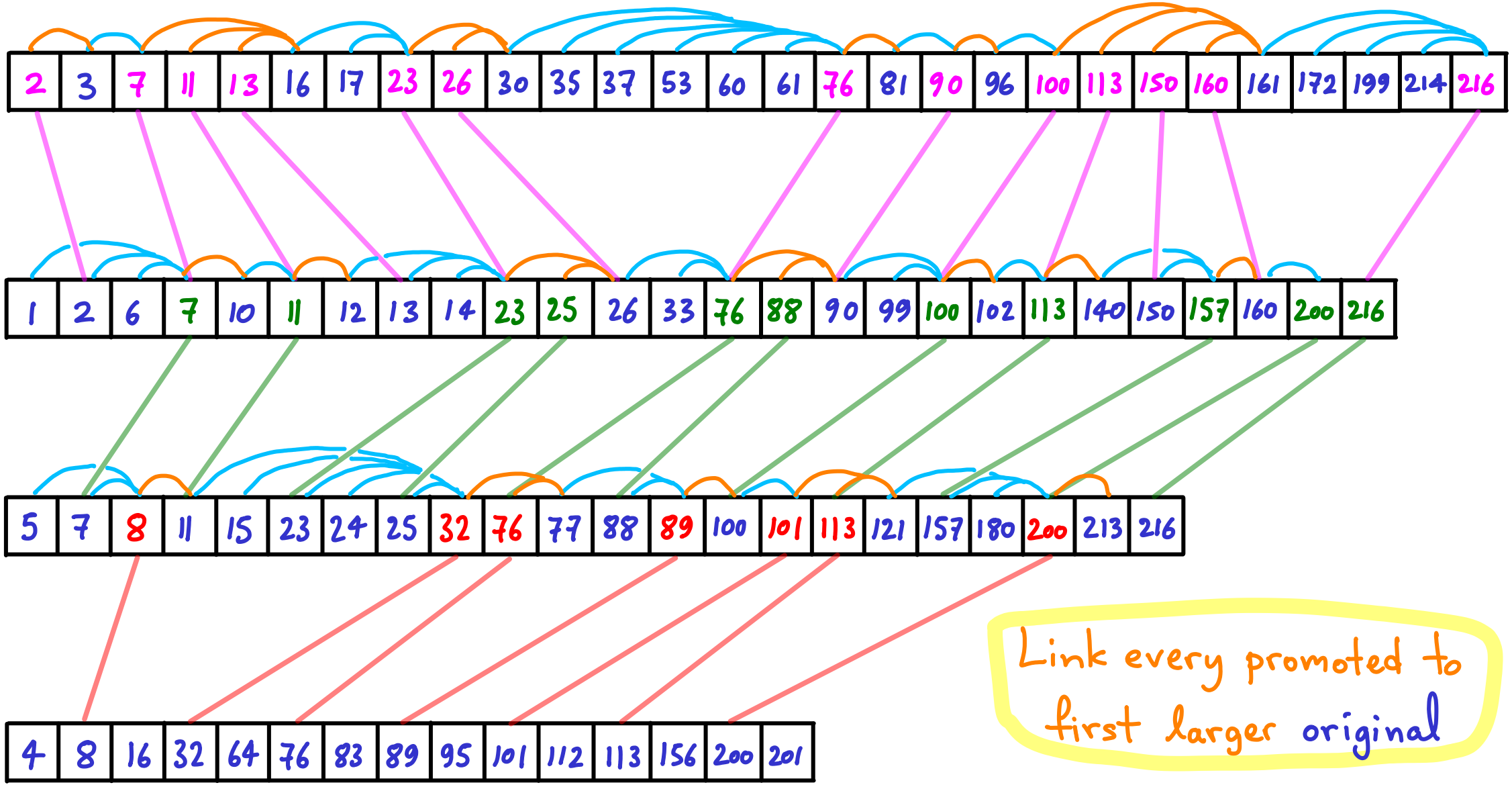

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Continue

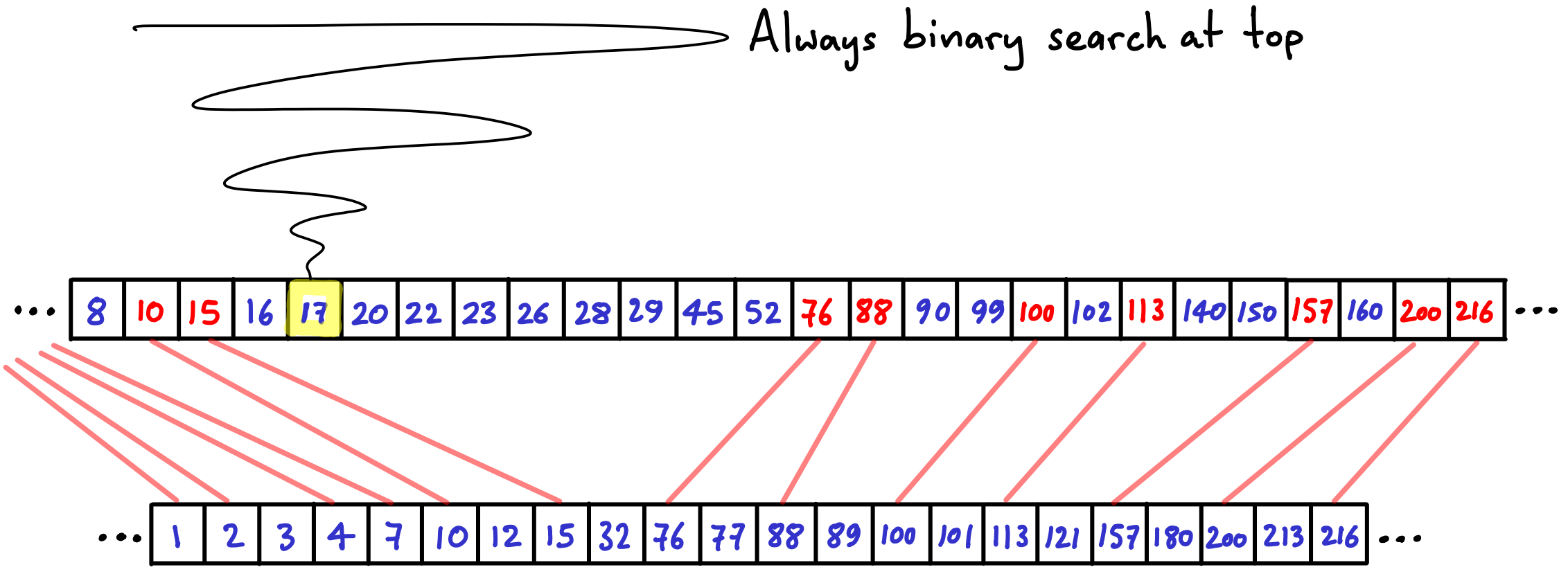

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Compare

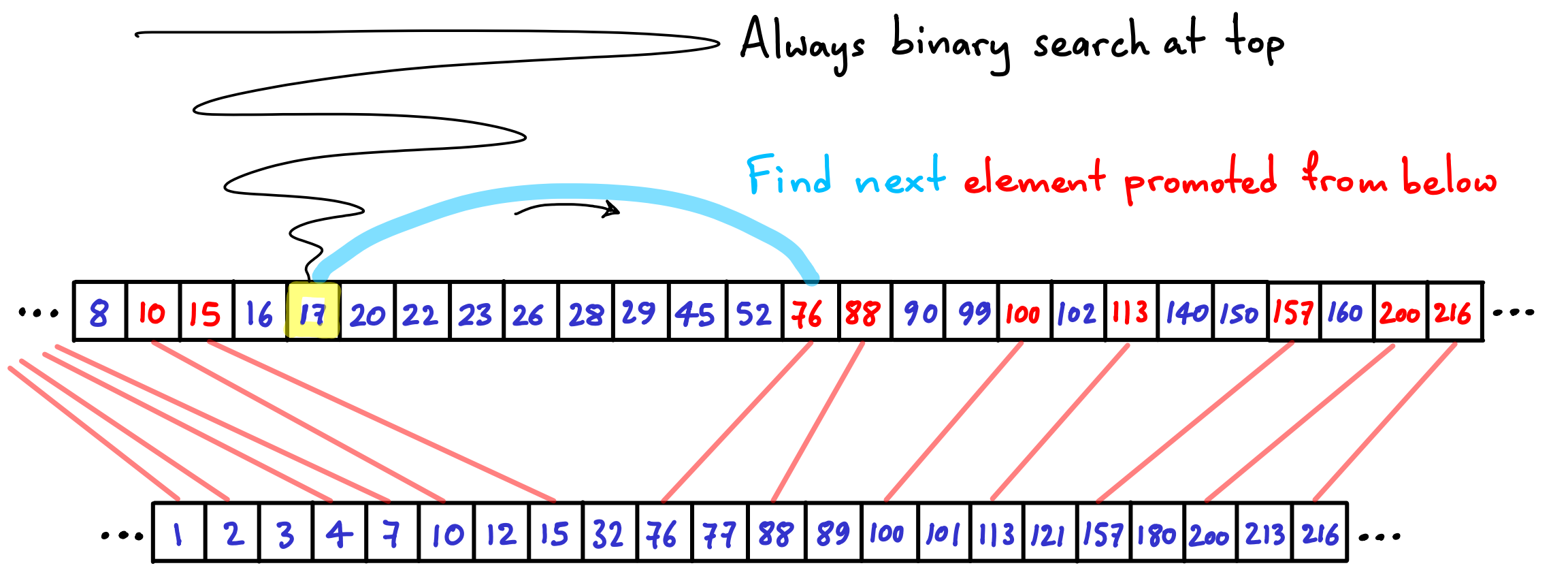

Search(35) → Binary search top level
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
time = O(k+logn)

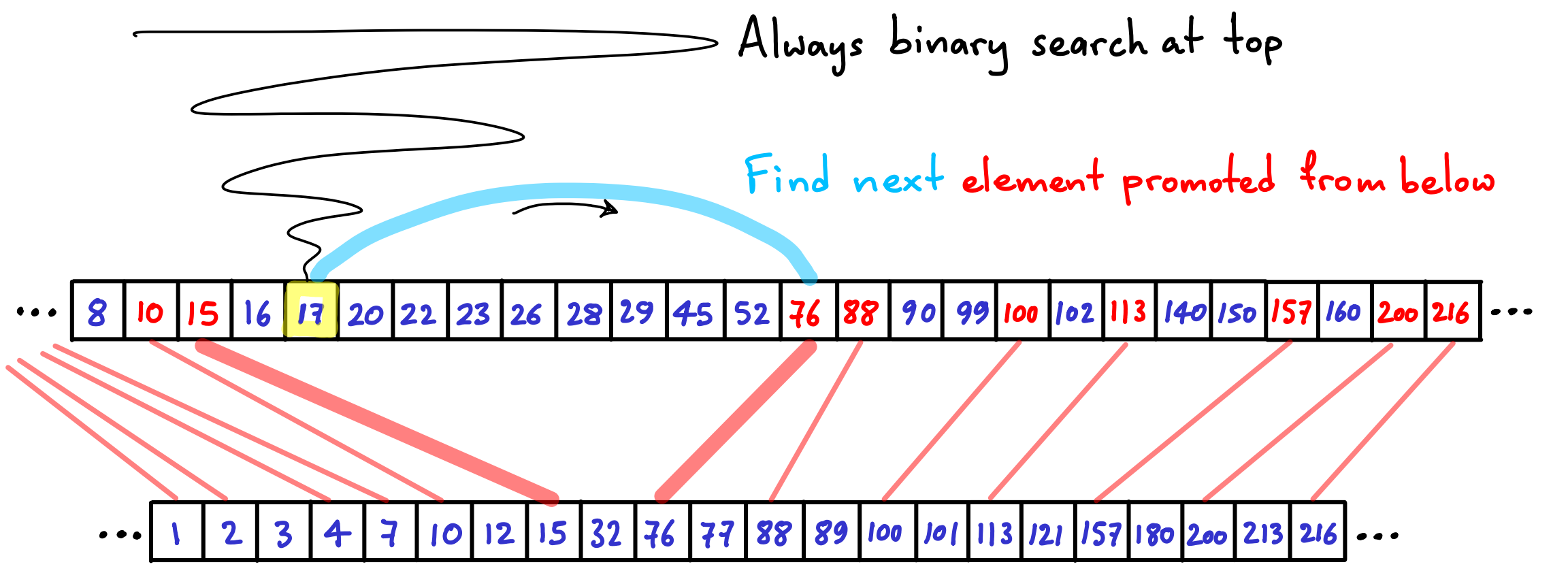

2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Suppose we DO care about pred./succ.

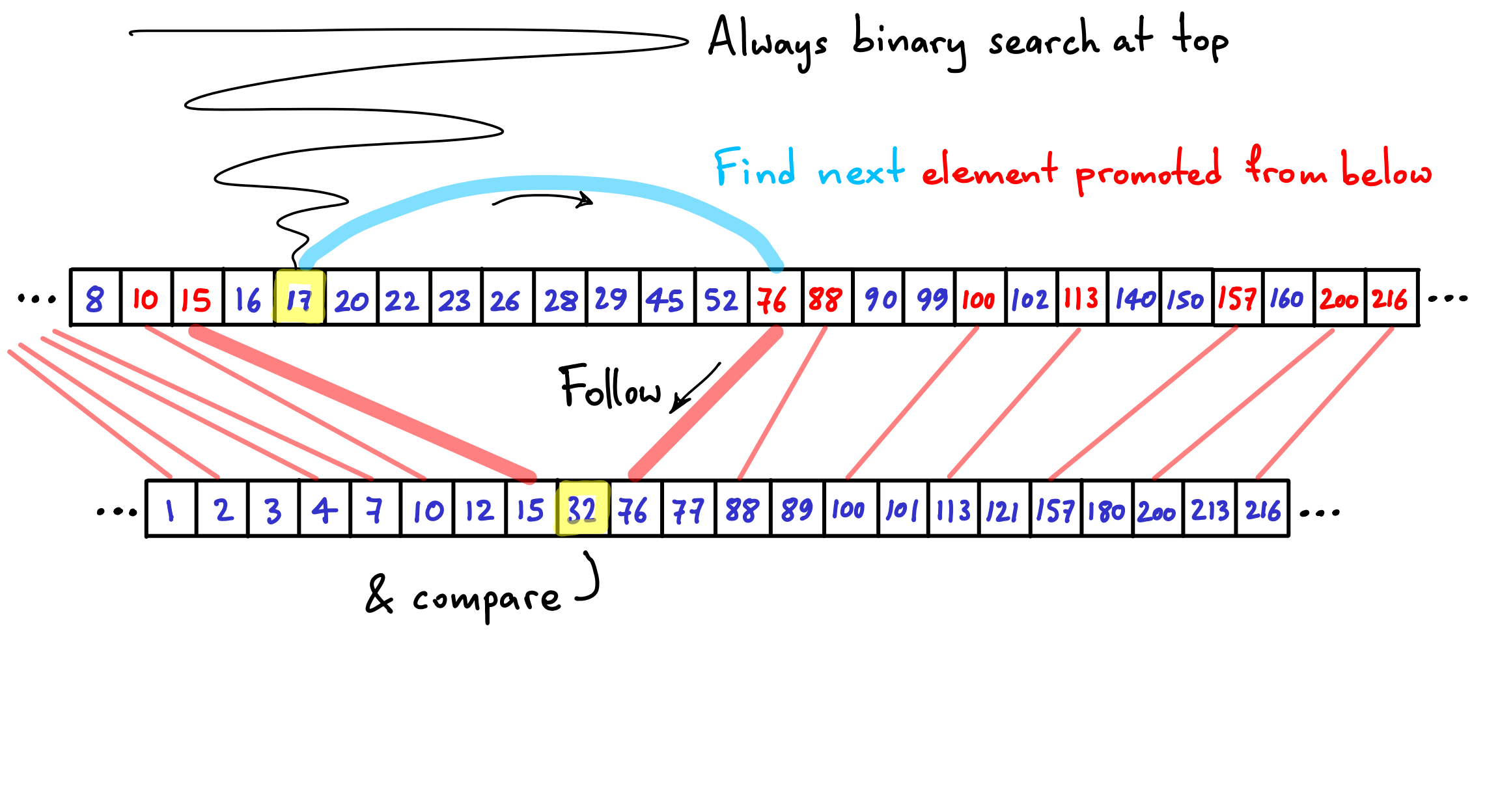

x in L4?
2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Suppose we DO care about pred./succ.

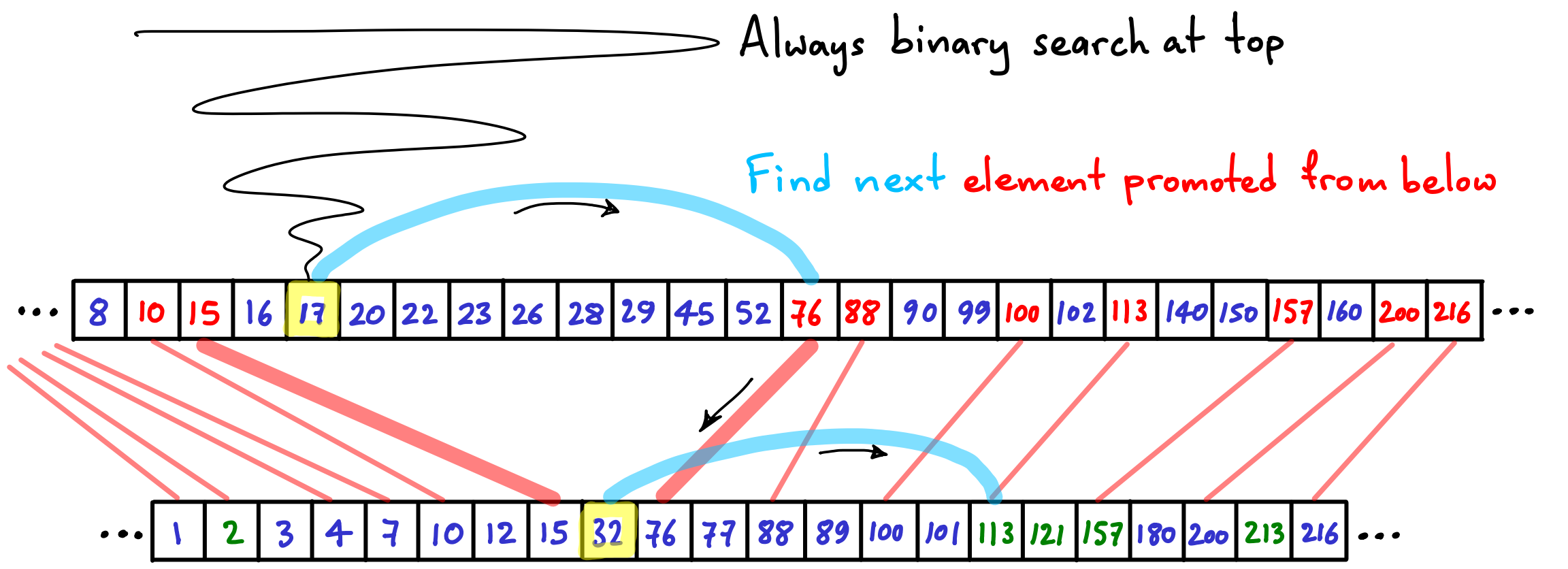

2 3 7 11 13 16 17 23 26 30 35 37 53 60 61 76 81 90 96 100 113 150 160 161 172 199 214 216
1 2 6 7 10 11 12 13 14 23 25 26 33 76 88 90 99 100 102 113 140 150 157 160 200 216
5 7 8 11 15 23 24 25 32 76 77 88 89 100 101 113 121 157 180 200 213 216
4 8 16 32 64 76 83 89 95 101 112 113 156 200 201
Link every promoted to first larger original

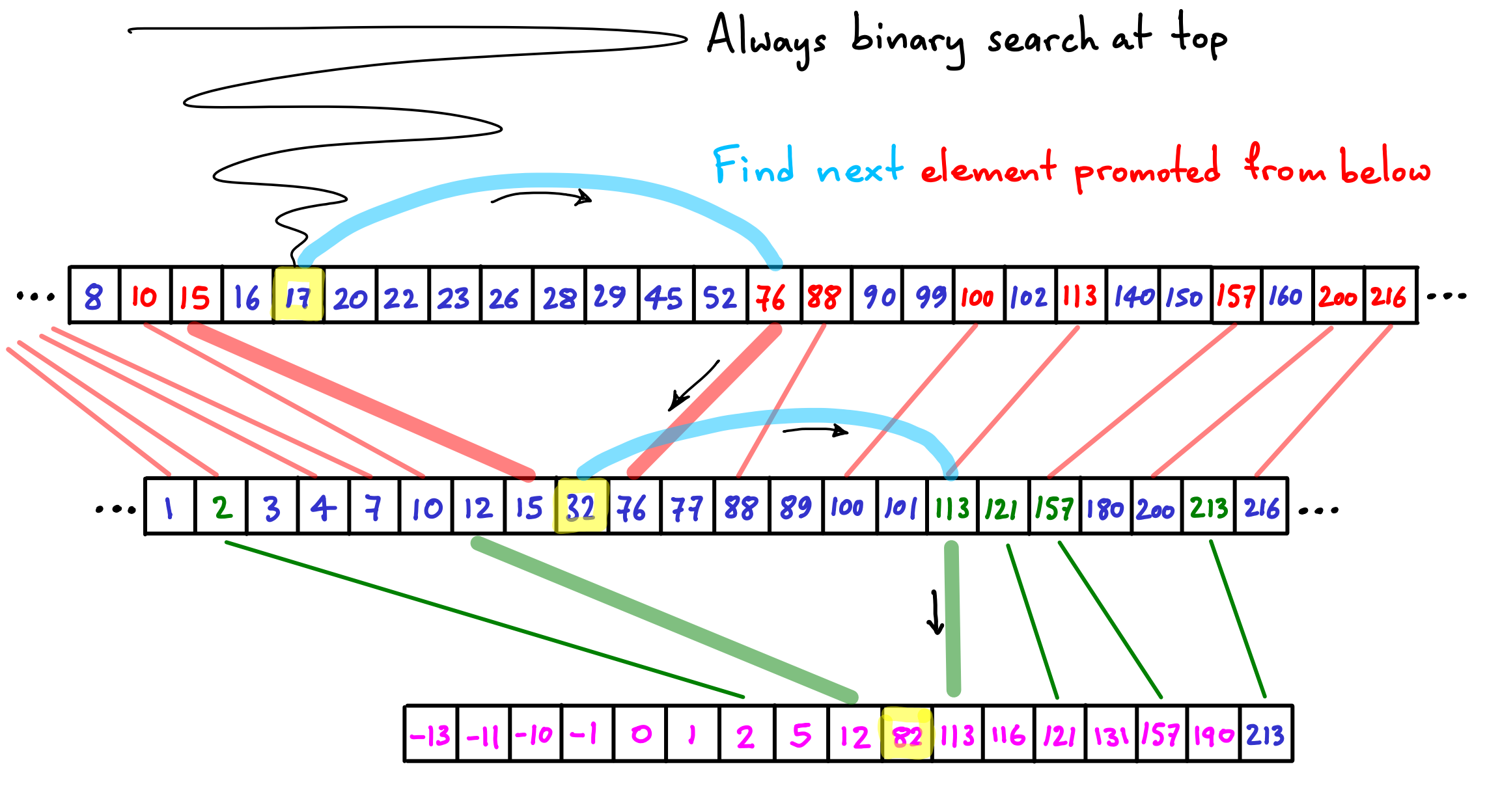

Always binary search at top
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216

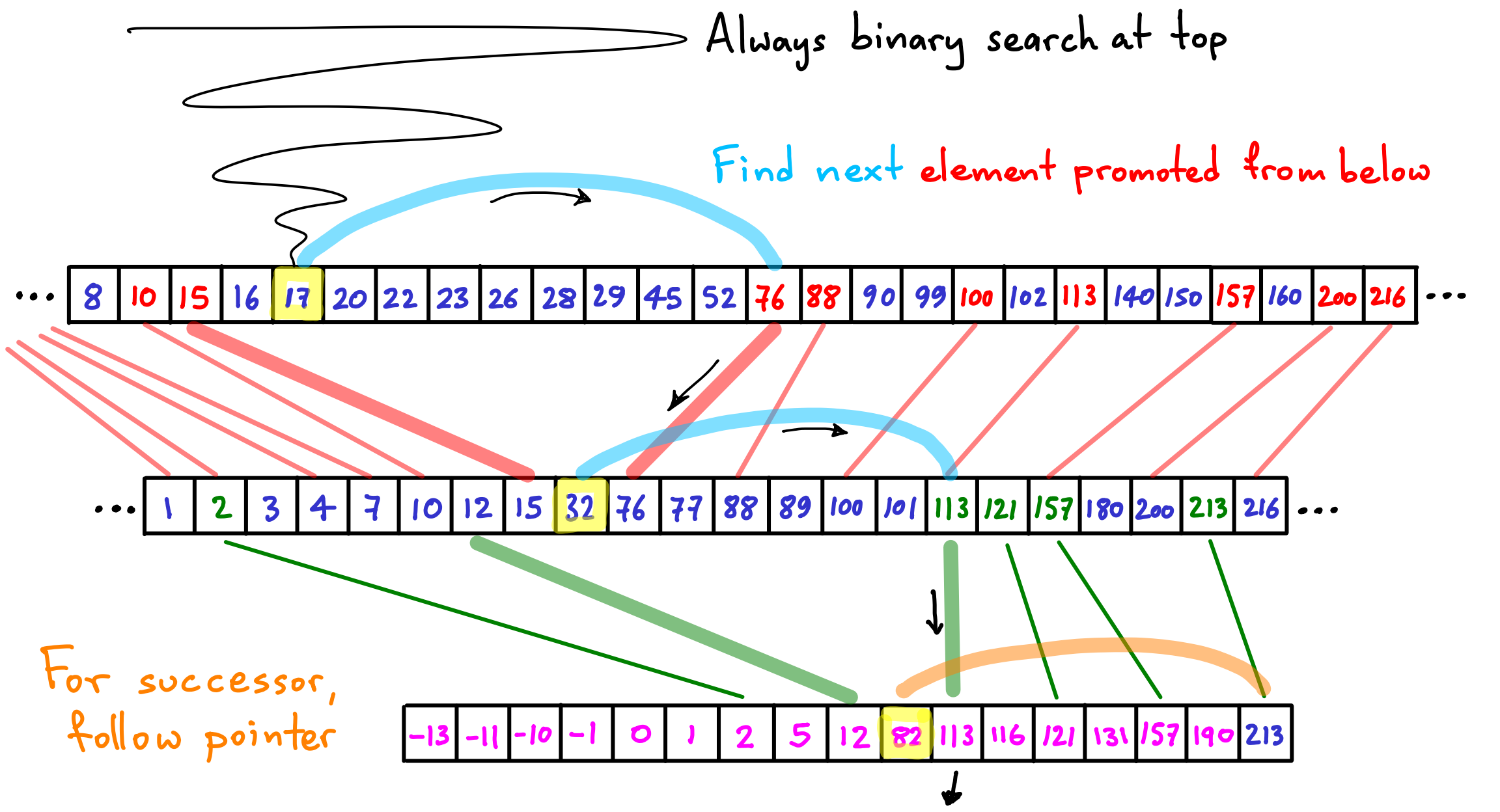

Always binary search at top
Find next element promoted from below
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216


Always binary search at top
Find next element promoted from below
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216
Now we know that 17 is between two elements in list below


Always binary search at top
Find next element promoted from below
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
Follow
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216
& compare


Always binary search at top
Find next element promoted from below
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216


Always binary search at top
Find next element promoted from below
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216
-13 -11 -10 -1 0 1 2 5 12 82 113 116 121 131 157 190 213


Always binary search at top
Find next element promoted from below
8 10 15 16 17 20 22 23 26 28 29 45 52 76 88 90 99 100 102 113 140 150 157 160 200 216
1 2 3 4 7 10 12 15 32 76 77 88 89 100 101 113 121 157 180 200 213 216
For successor, follow pointer
-13 -11 -10 -1 0 1 2 5 12 82 113 116 121 131 157 190 213