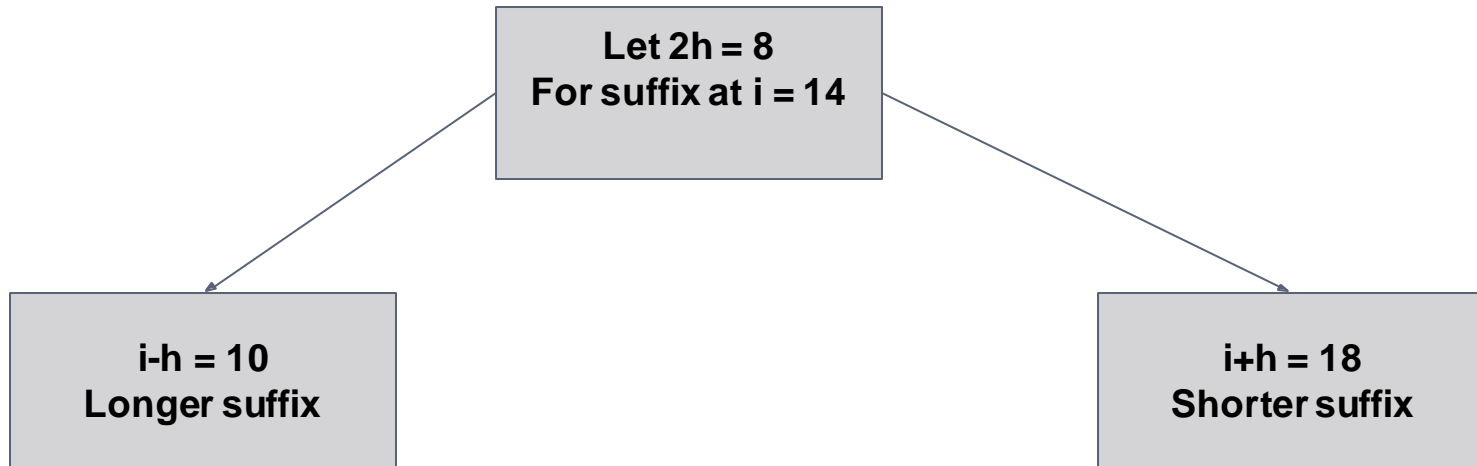


# $i+h$ vs $i-h$ observations



## Bottom up perspective

Suffix  $i$  (already sorted) is part of a longer suffix (to be sorted)

Observation on the **longer suffix** leads to  $O(n \log n)$

## Top down perspective

Suffix  $i$  (to be sorted) includes a shorter suffix (already sorted)

Observation on the **shorter suffix** leads to  $O(n \log n^2)$

# i-h

$i = 8$   
 $h = 2$   
 $i-h = 6$   
bra part of dabra

0	abracadabra
1	bracadabra
2	racadabra
3	acadabra
4	cadabra
5	adabra
6	dabra
7	abra
8	bra
9	ra
10	a
11	

The diagram illustrates the sliding window of indices  $i$  and  $h$  over the string "abracadabra". The window is currently at  $i=8$  and  $h=2$ , covering the substring "bra" at index 8. Arrows indicate the relationship between the current window and previous substrings: from index 8 to index 6 (dabra) and from index 8 to index 4 (cadabra).

$i = 8$   
 $h = 4$   
 $i-h = 4$   
bra part of cadabra

# Let's introduce: Group Start

The index of the **first time** for the group to appear

$i = 6$   
 $\text{suf}[6] = 8 = \text{bra}$   
 $\text{group}[8] = 2$   
 $\text{groupStart}[2] = 6$

$\text{groupStart}[\text{group}[\text{suf}[i]]] = 6$

Suffix (1)	Index	Group	Group Start
	11	0	0
a	10	1	1
abra	7	1	1
adabra	5	1	1
acadabra	3	1	1
abracadabra	0	1	1
bra	8	2	6
bracadabra	1	2	6
cadabra	4	3	8
dabra	6	4	9
ra	9	5	10
racadabra	2	5	10

# Let's introduce: Group Start

Suffix (1)	Index	Group	GStart		Suffix (2)	Index	Group	GStart		Suffix (4)	Index	Group	GStart
	11	0	0			11	0	0			11	0	0
<b>a</b>	10	1	1		<b>a</b>	10	1	1		<b>a</b>	10	1	1
<b>abra</b>	7	1	1		<b>abra</b>	7	2	2		<b>abra</b>	7	2	2
<b>adabra</b>	5	1	1		<b>abracadabra</b>	0	2	2		<b>abracadabra</b>	0	2	2
<b>acadabra</b>	3	1	1		<b>acadabra</b>	3	3	4		<b>acadabra</b>	3	3	4
<b>abracadabra</b>	0	1	1		<b>adabra</b>	5	4	5		<b>adabra</b>	5	4	5
<b>bra</b>	8	2	6		<b>bra</b>	8	5	6		<b>bra</b>	8	5	6
<b>bracadabra</b>	1	2	6		<b>bracadabra</b>	1	5	6		<b>bracadabra</b>	1	6	7
<b>cadabra</b>	4	3	8		<b>cadabra</b>	4	6	8		<b>cadabra</b>	4	7	8
<b>dabra</b>	6	4	9		<b>dabra</b>	6	7	9		<b>dabra</b>	6	8	9
<b>ra</b>	9	5	10		<b>ra</b>	9	8	10		<b>ra</b>	9	9	10
<b>racadabra</b>	2	5	10		<b>racadabra</b>	2	8	10		<b>racadabra</b>	2	10	11

# i-h

aacdzz

aacdee

aacdxx

aacdww

group 3

ssmnehabxx

ssmnaacdzz

ssmnaacdxx

group 9

ehabhz

ehabtn

ehabxx

aacdab

group 5

$h = 4$

- Note: These are invalid suffixes
- Assume we sorted suffixes on  $h = 4$
- We have grouped the suffixes over  $h = 4$
- In group 9, suffixes of length 8+
- We know they are sorted on first 4
- What about the remaining letters?
- We know they are sorted on their suffixes on their first 4 letters

# i-h

aacdzz

aacdee

aacdxx

aacdww

group 3

ssmnehabxx

ssmnaacdzz

ssmnaacdxx

group 9

ehabhz

ehabtn

ehabxx

aacdab

group 5

h = 4

- aacdzz is sorted on 4 letters
- It belongs to the **smallest** group
- It is 2nd 4+ letters from ssmnaacdzz
- Then ssmnaacdzz must be in the top of its new group for h = 8
- As it has same first 4 letters in its group
- And its 2nd 4 letters has smaller group

ssmnaacdzz

h = 8

# i-h

aacdzz  
aacdee  
aacdxx  
aacdwz

group 3

ssmnehabxx  
ssmnaacdzz  
ssmnaacdxx

group 9

ehabhz  
ehabtn  
ehabxx  
aacdab

group 5

$h = 4$

- Similarly for **aacdxx**
- It belong to the **smallest** group
- It is 2nd 4+ letters from **ssmnaacdxx**
- Then **ssmnaacdxx** must be in the next available position its new group for  $h = 8$

ssmnaacdzz  
ssmnaacdxx

$h = 8$

# i-h

aacdzz  
aacdee  
aacdxx  
aacdwz

group 3

ehabhz  
ehabtn  
ehabxx  
aacdab

group 5

h = 4

ssmnehabxx  
ssmnaacdzz  
ssmnaacdxx

group 9

- Moving to the next smallest group = 5
- It has **ehabxx**
- It is part of **ssmnehabxx**
- So add it in next possible position in its group h = 8
- Now this new group has 3 suffixes sorted on first 8 letters

ssmnaacdzz  
ssmnaacdxx  
ssmnehabxx

h = 8


See notes **below** for a written text in case needed



# Assume sorted on h

- We don't sort in-place like sort function
- So create a new array (newSuf) for sorted suffixes

```
//sort using 2h in the array newSuf
for (int i = 0; i < n; i++) {
    int j = suf[i] - h;
    if (j < 0)
        continue;
    newSuf[groupStart[group[j]]++] = j;
}
```



- j is a 4 letters sorted suffix in its group
- suf[i] is part of suffix at j
- suf[i] is the smallest in current order
- then put suf[j] in the next available position in its **group**

- groupStart[group[j]] is the next available position in its group
- Add suffix at j in this position
- Increment for next possible one

# Once sorted over 2h

```
for (int i = 1; i < n; i++) { //compute the 2h group data given h group data
    bool newgroup = group[newSuf[i - 1]] < group[newSuf[i]] || // Smaller in current group
        (group[newSuf[i - 1]] == group[newSuf[i]] && // Or my 2nd h letters smaller
         group[newSuf[i - 1] + h] < group[newSuf[i] + h]);

    sorGroup[i] = sorGroup[i - 1] + newgroup;

    if (newgroup)
        groupStart[sorGroup[i]] = i;
}
```

- Build new sorted groups and group start
- Very similar to old one
- Either we already in different groups, so we are still different
- Are we were in same group, and out **shorter suffix** is different

# Initialization

- One can do that in several easy ways
- Quick sort
  - Sort in  $O(n \log n)$  based on the first letter
  - Iterate over strings and build group and groupStart arrays
- Bucket sort (linear)
  - Linked list (vector) of positions per letter
  - E.g. `vector<vector<int>>` lists;
  - Then iterate and build group and groupStart arrays
  - *Or one might use any 2 1D arrays to build these lists*
  - Use 1 array has head of list and the other as next (->)
  - Iterating on them = iterating on the linked lists