

FÖRELÄSNING 5

Datastrukturer och algoritmer
KYH – 2022 HT

Andreas Nilsson Ström

The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the image, creating a modern, layered effect. The left side of the image is a solid, very light blue.

Repetition

Uppvärmning!

Algorithm: Summera faktet-siffror

- ▶ $n!$ betyder $n \times (n - 1) \times \dots \times 3 \times 2 \times 1$
- ▶ Till exempel, $10! = 10 \times 9 \times \dots \times 3 \times 2 \times 1 = 3628800$,
och summan av siffrorna i $10!$ Är $3 + 6 + 2 + 8 + 8 + 0 + 0 = 27$
- ▶ Uppgift: Hitta summan av siffrorna i numret 100!
- ▶ Källa: [ProjectEuler.net problem 20](https://projecteuler.net/problem/20)

Algorithm: Summera faktet-siffror

- ▶ $n!$ betyder $n \times (n - 1) \times \dots \times 3 \times 2 \times 1$
- ▶ Till exempel, $10! = 10 \times 9 \times \dots \times 3 \times 2 \times 1 = 3628800$,
och summan av siffrorna i $10!$ Är $3 + 6 + 2 + 8 + 8 + 0 + 0 = 27$
- ▶ Uppgift: Hitta summan av siffrorna i numret 100!
- ▶ Svar: 648
- ▶ Källa: [ProjectEuler.net problem 20](https://projecteuler.net/problem/20)

Agenda

- ▶ Sortering och sökning
- ▶ Algoritmer för sökning i data
- ▶ Algoritmer för sortering av data

Agenda

Sortering och sökning

- ▶ Algoritmer för sökning i data
- ▶ Algoritmer för sortering av data

Agenda

Sortering och sökning

- * Algoritmer för sökning i data
- ▶ Algoritmer för sortering av data

Sökning

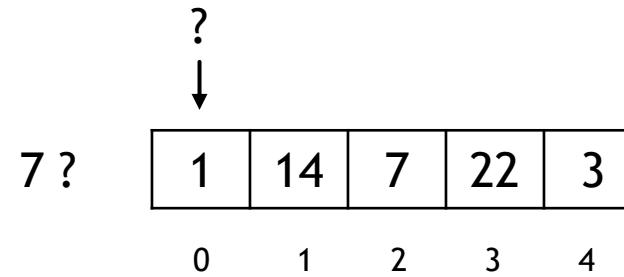
- ▶ Sökning är en viktig funktion i alla datastrukturer
- ▶ Det finns många olika metoder att hitta element i en struktur
 - ▶ Vi kommer framför allt utforska två olika varianter just nu
- ▶ Datan kan lagras i olika strukturer, som arrayer, länkade listor, träd, grafer...
- ▶ Vilken algoritm man föredrar beror dels på datan som lagras, men också vilken struktur den ligger lagrad i...

Sökning

- ▶ Gemensamt för sökningar:
 - ▶ Vi letar efter ett värde i en lista
 - ▶ Hittar vi rätt värde så ger vi tillbaka indexet

Linjär sökning

- ▶ A.k.a sekvensiell sökning
- ▶ Börja från ena änden, leta efter värdet.



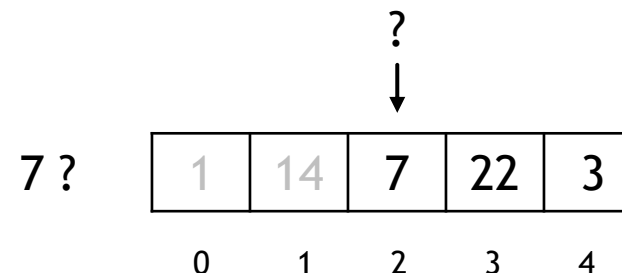
Linjär sökning

- ▶ A.k.a sekvensiell sökning
- ▶ Börja från ena änden, leta efter värdet.

7 ?

	?				
	↓				
	1	14	7	22	3
	0	1	2	3	4

Linjär sökning



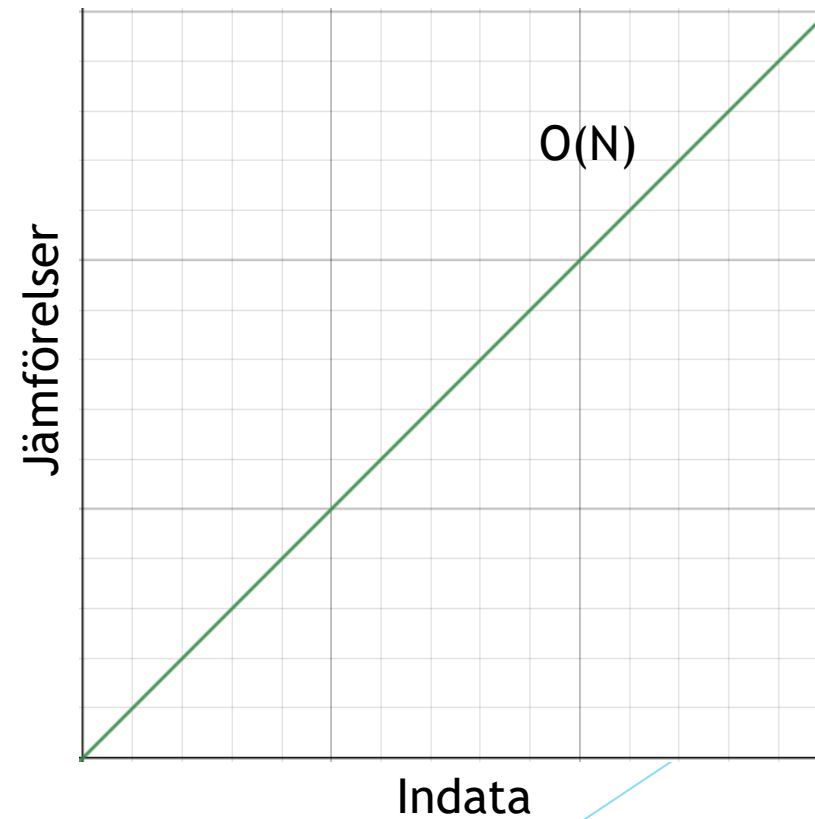
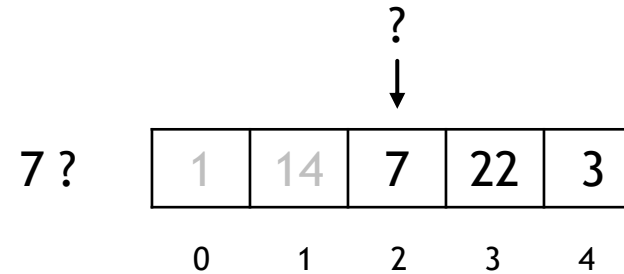
- ▶ A.k.a sekvensiell sökning
- ▶ Börja från ena änden, leta efter värdet.
- ▶ Best case? 1 operation: Värdet finns på index 0
- ▶ Worst case? N operationer: Värdet finns inte alls

```
def search(unordered_list, item):  
    for index, value in enumerate(unordered_list):  
        if item == value:  
            return index  
    return None
```

Linjär sökning

Vad är största problemet?

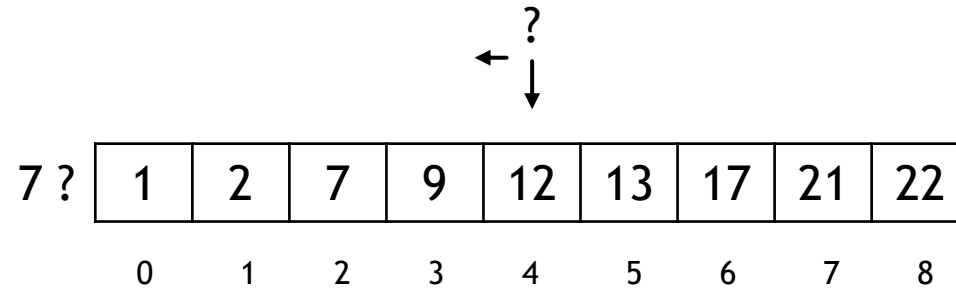
- ▶ Listan är osorterad
- ▶ Om vi istället söker i sorterad data?



Sorterad data

1	2	7	9	12	13	17	21	22
0	1	2	3	4	5	6	7	8

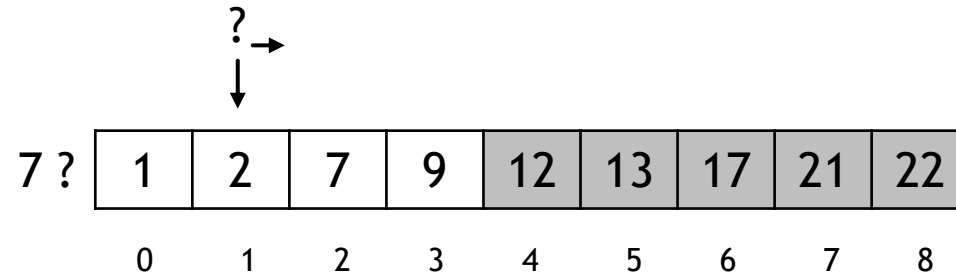
Binär sökning



Algoritm:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

Binär sökning



7 ?	1	2	7	9	12	13	17	21	22
	0	1	2	3	4	5	6	7	8

Algoritm:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

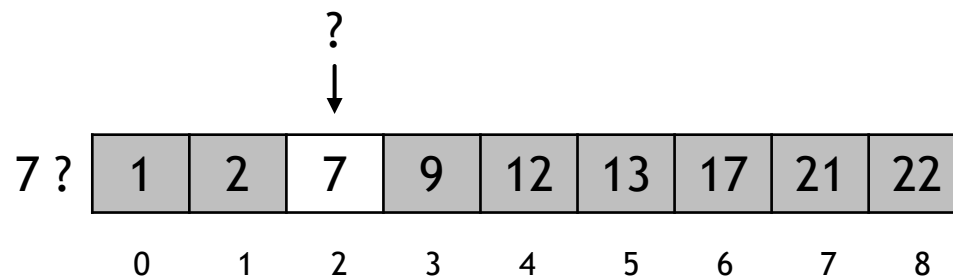
Binär sökning

		?							
		↓							
7 ?	1	2	7	9	12	13	17	21	22
	0	1	2	3	4	5	6	7	8

Algoritm:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

Binär sökning



Algoritm:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

```
def binary_search(a_list, item):  
    first = 0  
    last = len(a_list) - 1  
  
    while first <= last:  
        midpoint = (first + last) // 2  
        if a_list[midpoint] == item:  
            return midpoint  
        elif item < a_list[midpoint]:  
            last = midpoint - 1  
        else:  
            first = midpoint + 1  
  
    return None
```

Binär sökning

	first			mid				last	
	↓			↓				↓	
7 ?	1	2	7	9	12	13	17	21	22
	0	1	2	3	4	5	6	7	8

Algoritm:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

```
def binary_search(a_list, item):  
    first = 0  
    last = len(a_list) - 1  
  
    while first <= last:  
        midpoint = (first + last) // 2  
        if a_list[midpoint] == item:  
            return midpoint  
        elif item < a_list[midpoint]:  
            last = midpoint - 1  
        else:  
            first = midpoint + 1  
  
    return None
```

Binär sökning

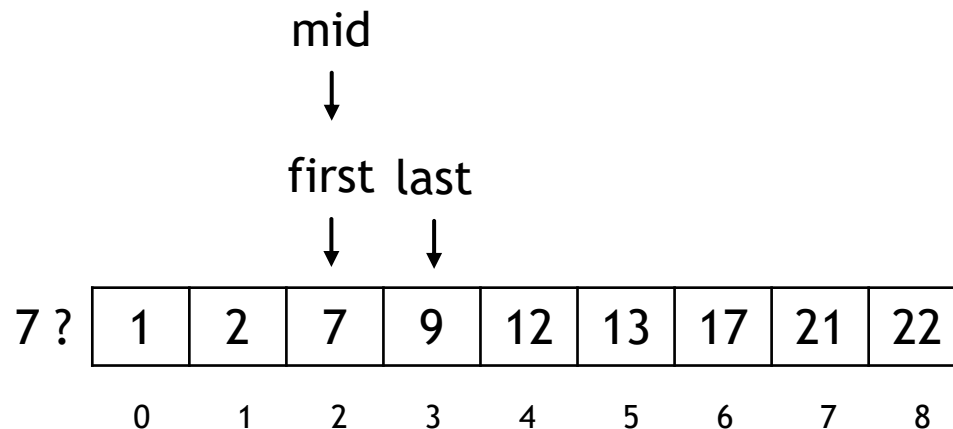
	first	mid		last					
	↓	↓		↓					
7 ?	1	2	7	9	12	13	17	21	22
	0	1	2	3	4	5	6	7	8

Algoritm:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

```
def binary_search(a_list, item):  
    first = 0  
    last = len(a_list) - 1  
  
    while first <= last:  
        midpoint = (first + last) // 2  
        if a_list[midpoint] == item:  
            return midpoint  
        elif item < a_list[midpoint]:  
            last = midpoint - 1  
        else:  
            first = midpoint + 1  
  
    return None
```

Binär sökning

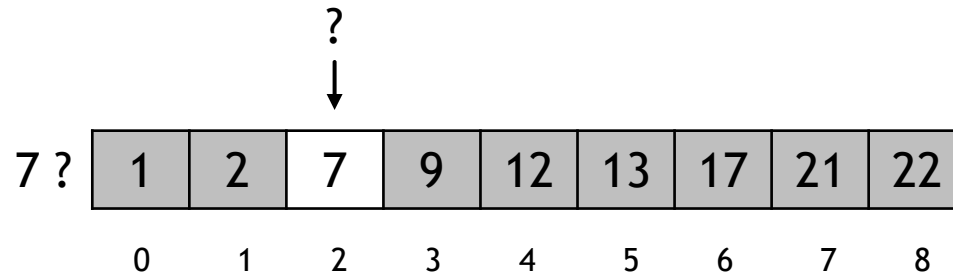


Algoritmen:

- ▶ Börja i mitten
- ▶ Värdet mindre eller större?
- ▶ Dela upp listan i en höger och vänster-del. Välj riktning.
- ▶ Repetera!

```
def binary_search(a_list, item):  
    first = 0  
    last = len(a_list) - 1  
  
    while first <= last:  
        midpoint = (first + last) // 2  
        if a_list[midpoint] == item:  
            return midpoint  
        elif item < a_list[midpoint]:  
            last = midpoint - 1  
        else:  
            first = midpoint + 1  
  
    return None
```

Binär sökning



Analys?

Vad är worst case?

För varje jämförelse slänger vi bort halva listan.

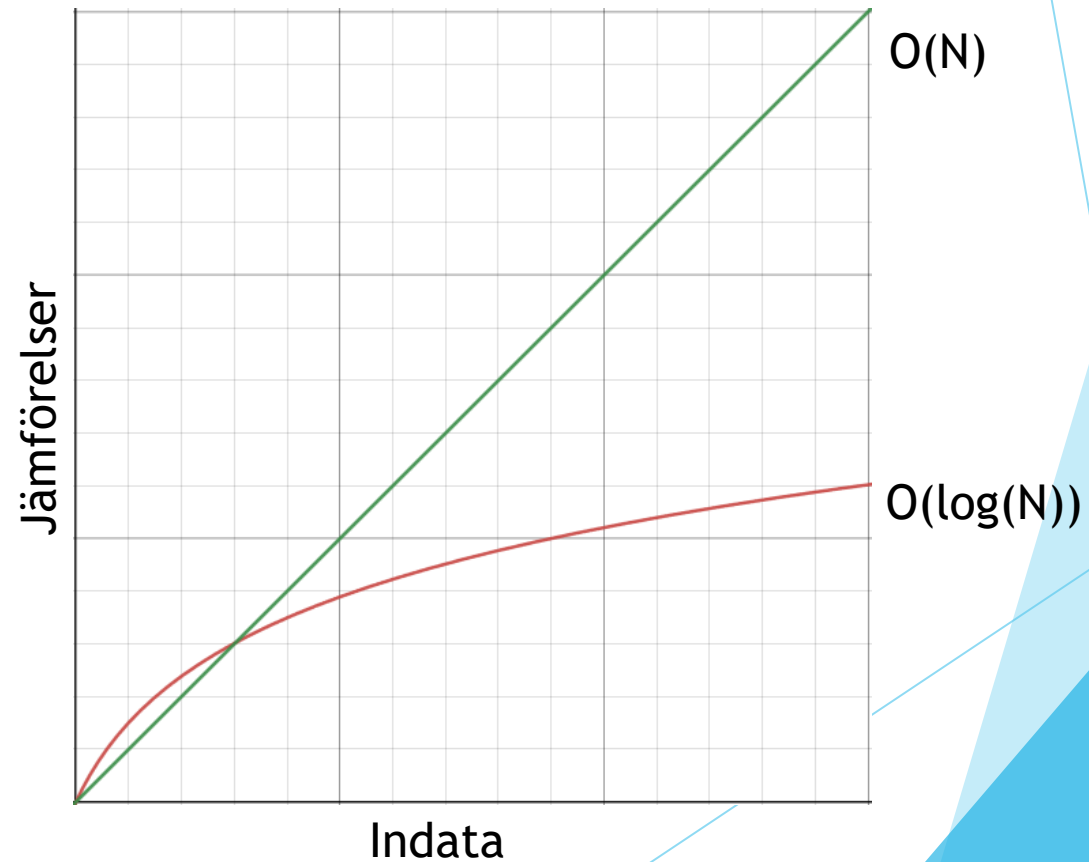
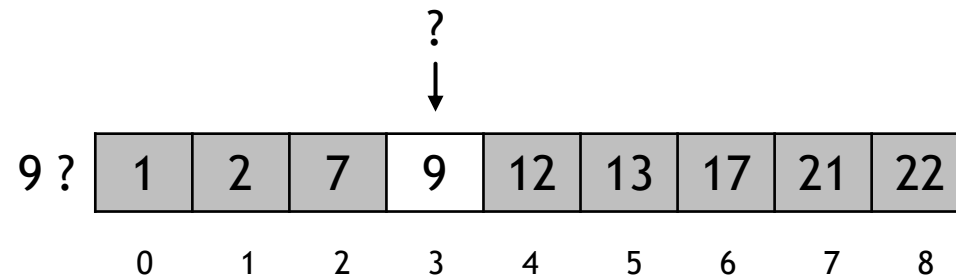
Efter 3 jämförelser har vi slängt bort halva listan tre gånger

Jämförelser	~ Antal värden kvar
1	$n/2$
2	$n/4$
3	$n/8$
...	...
i	$n / (2^i)$

Binär sökning

Analys?

Vad är worst case?



Binär sökning

- ▶ Den här algoritmen använder en stil vi inte har sett förut, där man delar in problemet i mindre delar, sen löser det mindre problemet i sig.
- ▶ Detta kallas "Divide and Conquer"-principen
- ▶ Väldigt kraftfullt sätt att lösa stora problem, och ofta väldigt snabbt

Andra sökalgoritmer

- ▶ Jump search. Börja längst till vänster. Hoppa några platser åt höger varje gång.
- ▶ Interpolation search
- ▶ Exponential search
- ▶ ... Många fler
- ▶ "Bogo search" - Slumpa fram ett index. Repetera tills vi hittar rätt.

Agenda

Sortering och sökning

- ▶ Algoritmer för sökning i data
- * Algoritmer för sortering av data

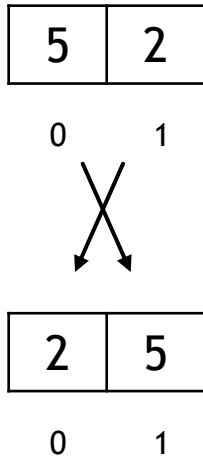
Sortering

- ▶ Sortering betyder att omorganisera datan på ett sådant sätt att den till exempel är i fallande eller stigande ordning
- ▶ Det är ett av de viktigaste problemen att lösa inom datavetenskap
- ▶ När datan är sorterad så kan den mer effektivt hanteras
 - ▶ Sökas i
 - ▶ Hämtas
 - ▶ ...
- ▶ Nästan oavsett ändamål. En samling namn. Telefonnummer. Saker på en att-göra-lista.

Bubble sort

- ▶ Idén är väldigt enkel.
 - ▶ Givet en osorterad lista
 - ▶ Gå igenom varje par som är grannar och byt plats om de är i fel ordning
 - ▶ Repetera tills listan är sorterad

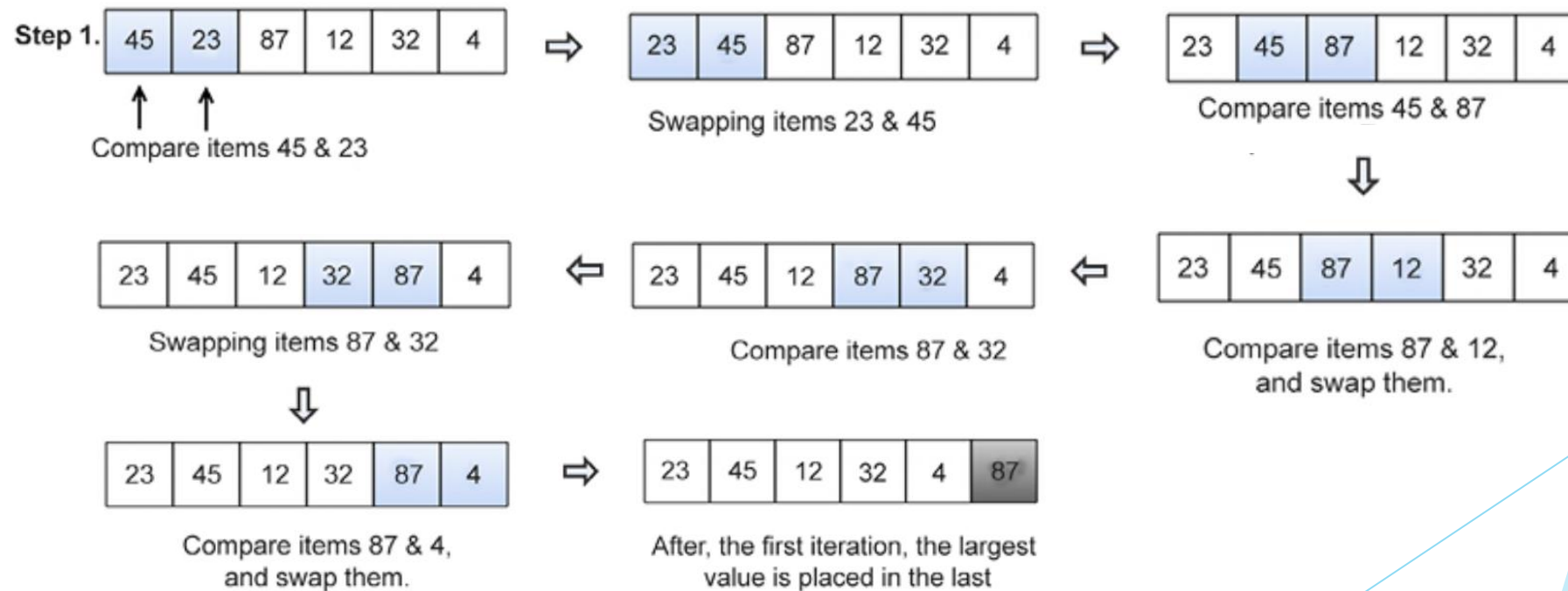
Bubble sort



```
ul = [5, 2]                                # "unsorted list"  
ul[0], ul[1] = ul[1], ul[0]               # Byt plats  
  
print(ul)    # [2, 5]
```

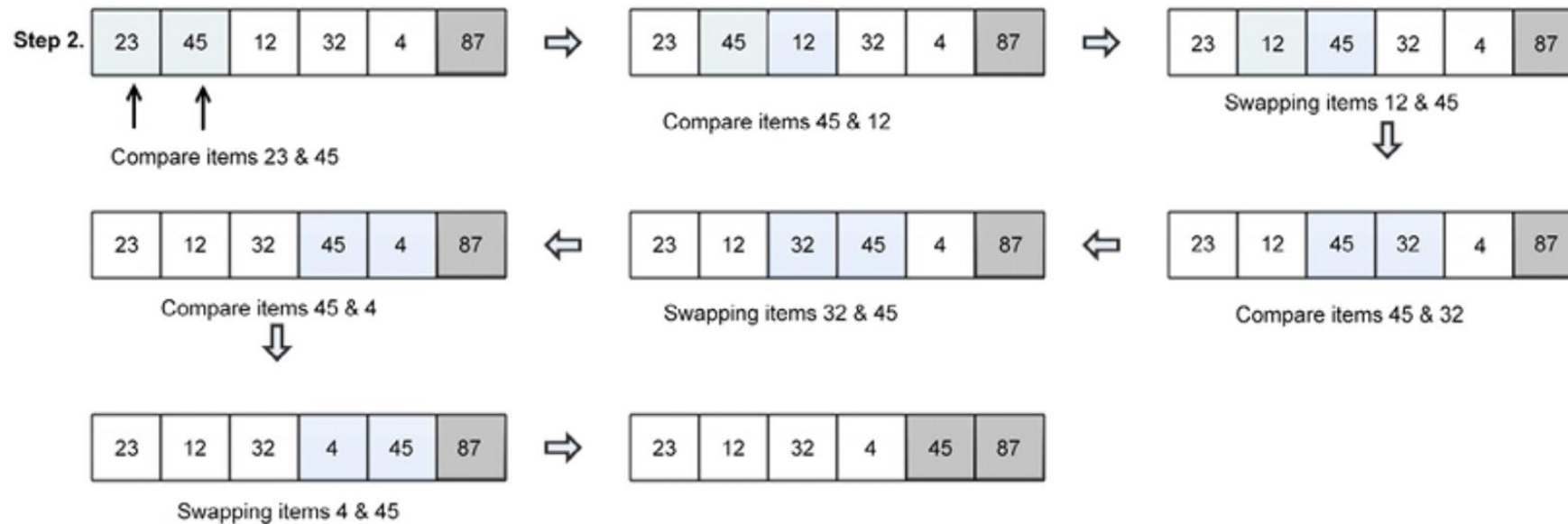
Bubble sort

- Om vi nu tar samma idé och använder på en större lista

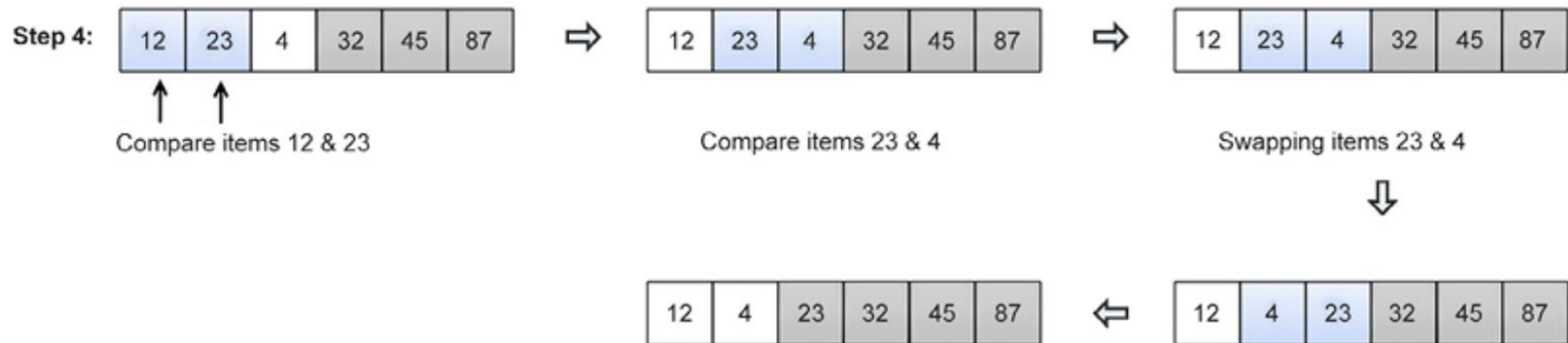


Bubble sort

- ▶ Repetera från början igen, men skippa att jämföra med sista platsen



Bubble sort



Bubble sort



Övning

Implementera bubble sort!

▶ `bubble_sort([5, 3, 4, 7, 2]) -> [2, 3, 4, 5, 7]`

Pseudo-kod för en grundläggande implementering:

- ▶ Function `bubble_sort(unsorted_list)`:
 - ▶ Loop: Set `i` = number from 0 to length of `unsorted_list` - 1
 - ▶ Loop: Set `j` = number from 0 to length of `unsorted_list` - 1
 - ▶ If values of place `j` and `j+1` are not in order:
 - ▶ Swap values

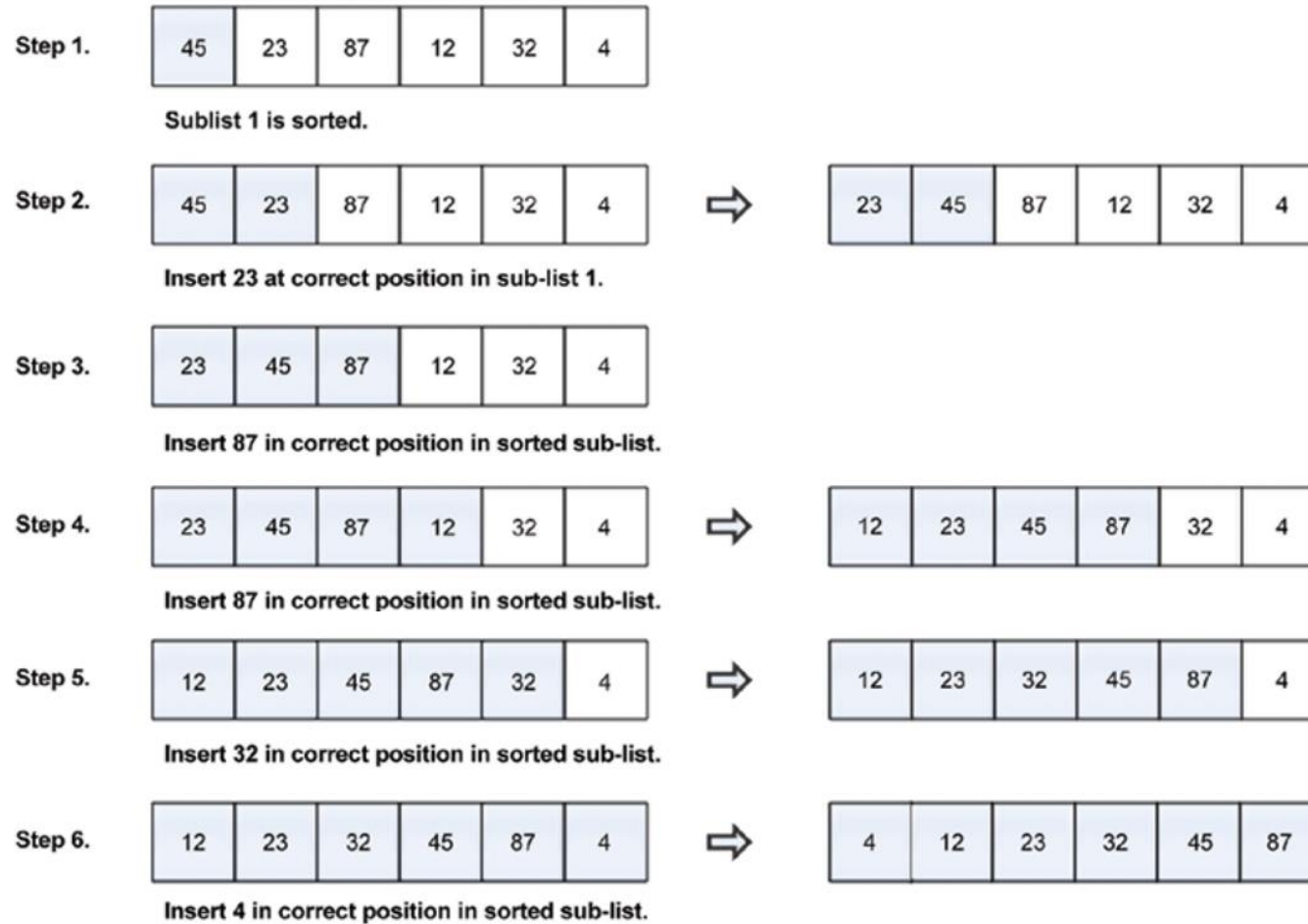
Bubble sort

- ▶ Hur växer antalet jämförelser med storleken på input?
- ▶ Dubbla loopar som går igenom hela listan = Antalet jämförelser blir $N * N$
- ▶ Alltså: $O(N^2)$

Insertion sort

- ▶ Algoritm där man börjar med en osorterad lista. Sen delar man in den i sorterade element och osorterade element.
- ▶ Sorterade på vänster, osorterade på höger.
- ▶ Väx sorterade med ett element i taget, som man lägger på rätt plats
 - ▶ Repetera till hela listan är sorterad

Insertion sort



Insertion sort

► Kod för sorteringen

```
def insertion_sort(unsorted_list):  
    for index in range(1, len(unsorted_list)):  
        search_index = index  
        insert_value = unsorted_list[index]  
  
        while search_index > 0 and unsorted_list[search_index-1] > insert_value:  
            unsorted_list[search_index] = unsorted_list[search_index-1]  
            search_index -= 1  
  
        unsorted_list[search_index] = insert_value
```

Insertion sort

- ▶ Låt oss ta ett till exempel
- ▶ Demo: Listan [5, 1, 100, 2, 10]

Insertion sort

- ▶ Bättre än bubble sort?
- ▶ Fortfarande loop i en annan loop som går igenom hela sakerna... Så nä.
- ▶ $O(N^2)$

Quick Sort

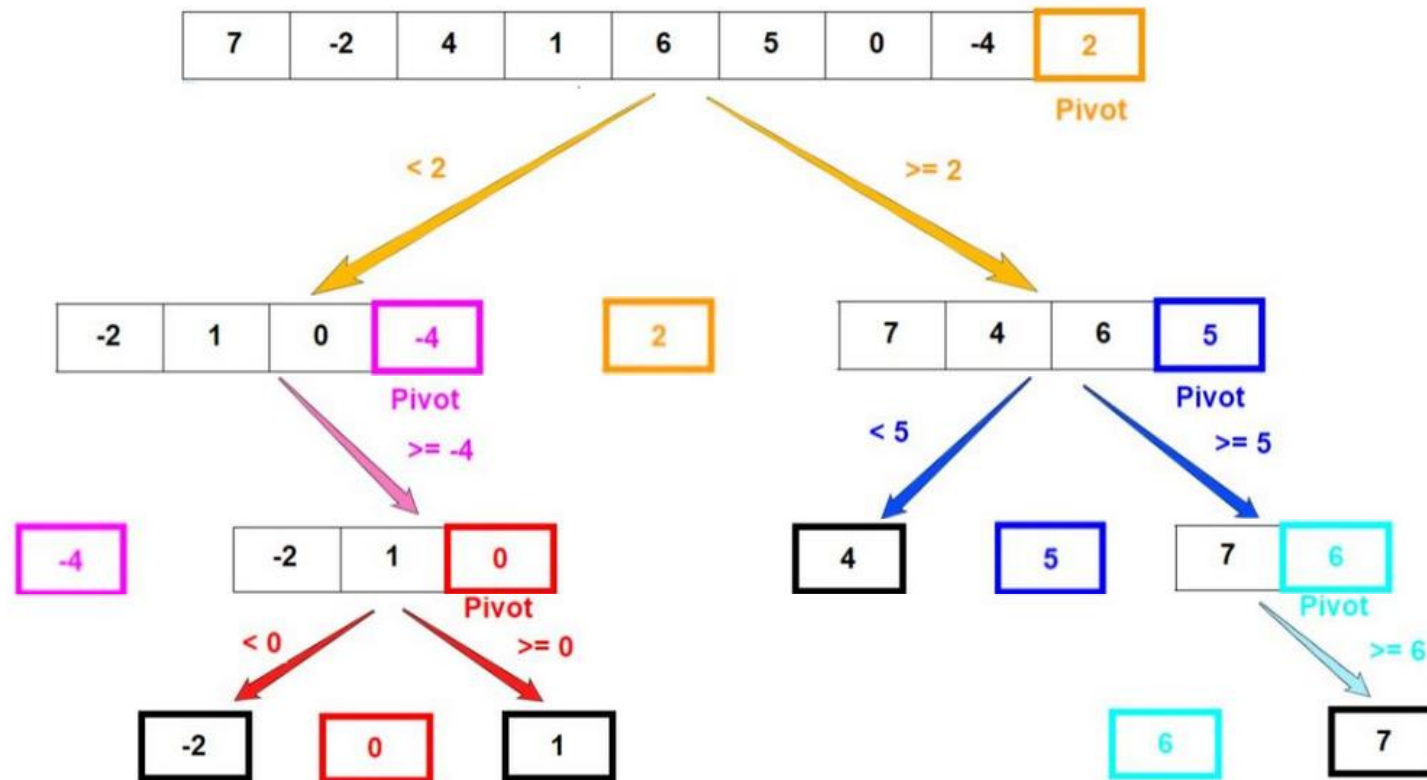
- ▶ Quick Sort använder Divide and Conquer-metoden för att dela upp problemet i mindre delar.
 - ▶ Dela upp
 - ▶ Lös den mindre delen
 - ▶ Samla ihop resultatet

Quick Sort

1. Välj ett tal ur listan (Kallas "pivot"-talet)
2. Dela upp listan i två delar
 1. En del med tal $<$ pivot
 2. En del med tal \geq pivot
3. Repetera steg 1-2 tills vi inte kan dela upp listan mer
4. Samla ihop och lägg i ordning

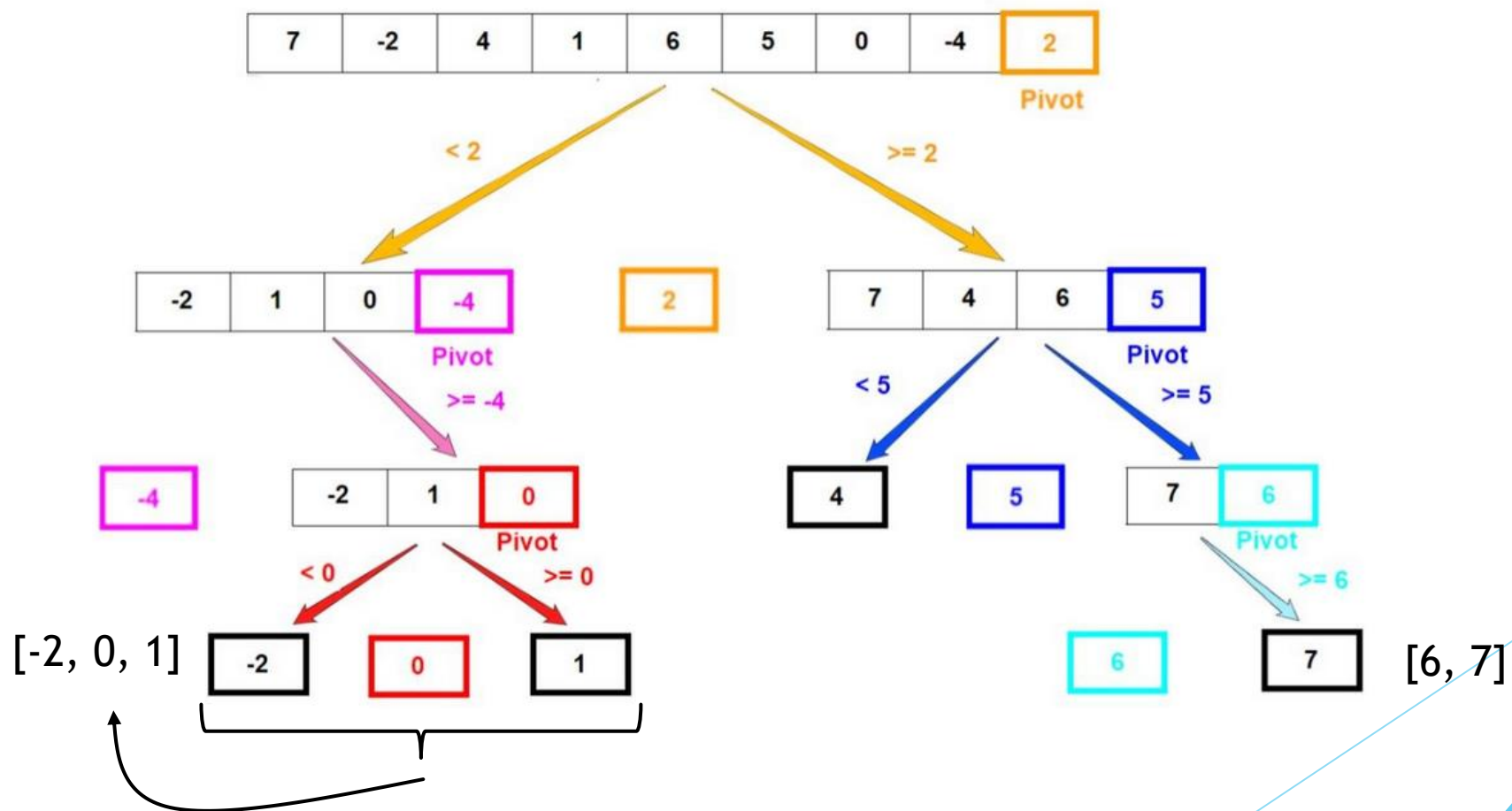
Quick Sort

Välj en pivot. Dela upp resten av listan i två delar. Repetera tills vi har hinkar med individuella element



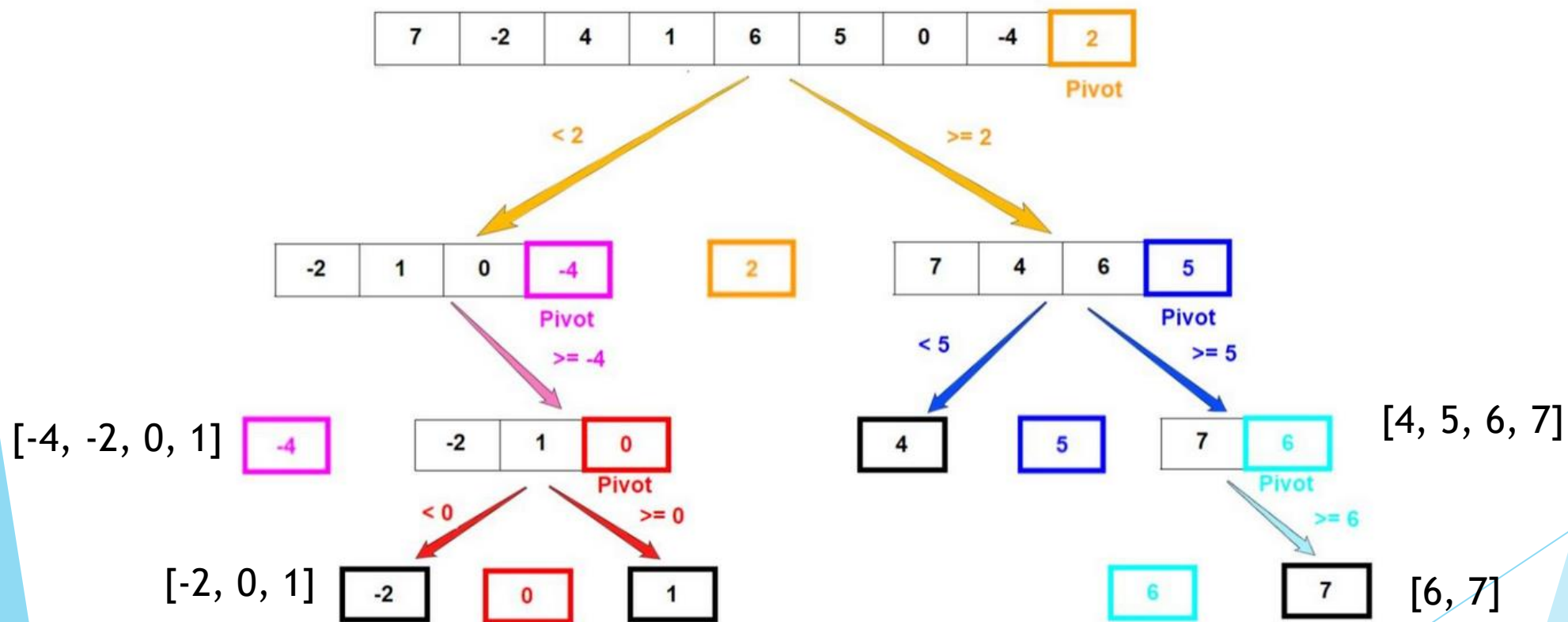
Quick Sort

Samla ihop från botten till toppen



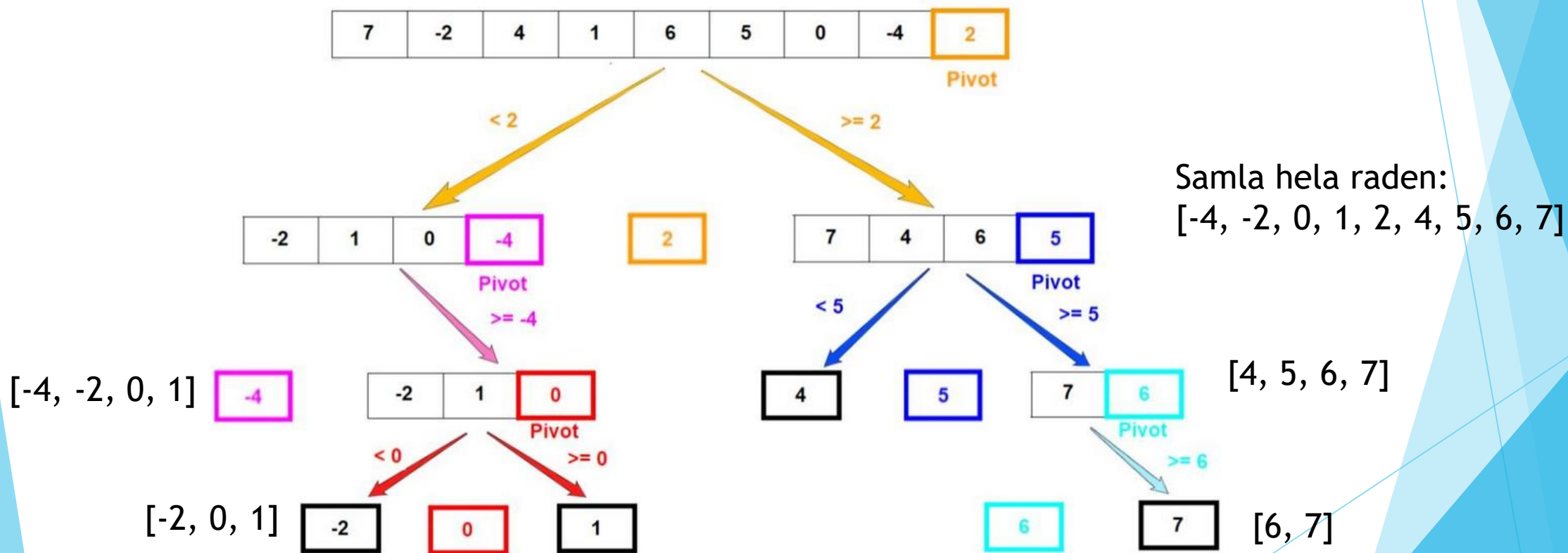
Quick Sort

Samla ihop från botten till toppen



Quick Sort

Samla ihop från botten till toppen



Andra sorteringsalgoritmer

- ▶ Selection Sort
- ▶ Shell Sort
- ▶ Merge Sort
- ▶ Timsort (Används av Python)
- ▶ ... Många många fler
- ▶ Bogosort - Randomisera lista. Är den inte sorterad? Repetera från start!

Diskussion

- ▶ Vad finns det för sökalgoritmer vi kan använda på osorterad data?
- ▶ Kan vi använda binär sökning i enkel-länkade listor?
 - ▶ Vilka algoritmer finns det som kan användas?
- ▶ Beskriv "Divide and Conquer"-principen
 - ▶ Ge några exempel på algoritmer som använder den: Sökning, Sortering
- ▶ Hur kan vi (algoritmiskt) avgöra om två kvadrater vi ritar överlappar?
 - ▶ (Hint: De har två egenskaper: Storlek och position)

Översikt

- ▶ F4: Abstrakta datastrukturer, Deque, Intro till sortering
- ▶ F5: Sortering och sökning
- ▶ F6: Rekursion
- ▶ F7: Träd och Grafer
- ▶ F8-F9: Repetition

	M	T	O	T	F	L	S
44	31	NOVEMBER 1	2	3	4	5	6
45	7	F 1	F 2		F 3 L 1	12	13
46	14 F 4	15	16 F 5	17	18 L 2	19	20
47	21 F 6	22	23 F 7	24	25 L 3	26	27
48	28 F 8	29 F 9	30	DECEMBER 1 T	2	3	4

Extramaterial

- ▶ Tom Scott
 - ▶ Video: [Sorting algorithms and Big O notation](#)
- ▶ Youtube: Computerphile
 - ▶ Video: [Quick Sort](#)
 - ▶ Video: [Getting Sorted & Big O Notation](#)
- ▶ Wikipedia: [Jämförelse av sorterings-algoritmer](#)

Mer läsning

Bok

- ▶ **Problem Solving with Algorithms and Data Structures Using Python**
- ▶ [Länk](#)

Dagens material:

- ▶ Sökning: Kapitel 5.2-5.4
- ▶ Sortering: Första delarna av: Kapitel 5.6, 5.7, 5.9, 5.12-5.16

Nästa gång: Rekursion

- ▶ Om man vill läsa i förväg: Kapitel 4 - Recursion

