# Cấu trúc dữ liệu và thuật toán sắp xếp

Fun and fast

Hoàng Thị Thảo, Lê Quốc Tuấn

Optimal seminar group

# Nội dung

1. Cấu trúc dữ liệu
(Data structure)

2. Các thuật toán sắp xếp

# Cấu trúc dữ liệu

## (Data structure)

# Định nghĩa

- Cấu trúc dữ liệu là tập hợp của các giá trị cùng mối quan hệ giữa chúng được mô tả thông qua các hàm số và phép toán.

# Định nghĩa

- Cấu trúc dữ liệu là tập hợp của các giá trị cùng mối quan hệ giữa chúng được mô tả thông qua các hàm số và phép toán.

- Cấu trúc dữ liệu cũng được hiểu là cách tổ chức và lưu trữ dữ liệu sao cho việc truy xuất và thi hành, cải tiến được hiệu quả.

# Định nghĩa

- Cấu trúc dữ liệu là tập hợp của các giá trị cùng mối quan hệ giữa chúng được mô tả thông qua các hàm số và phép toán.
- Cấu trúc dữ liệu cũng được hiểu là cách tổ chức và lưu trữ dữ liệu sao cho việc truy xuất và thi hành, cải tiến được hiệu quả.
- Với mỗi bài toán, việc lựa chọn cách thức tổ chức dữ liệu cần gắn chặt với các phép toán sẽ được sử dụng trên dữ liệu.

# Tại sao quan tâm?

- Tổ chức dữ liệu là cách thức con người thực hiện khi cần giải quyết một vấn đề trong cuộc sống.

# Tại sao quan tâm?

- Tổ chức dữ liệu là cách thức con người thực hiện khi cần giải quyết một vấn đề trong cuộc sống.
- Khi làm việc với lượng dữ liệu lớn, một cách tổ chức dữ liệu tốt sẽ tiết kiệm chi phí.

# Tại sao quan tâm?

- Tổ chức dữ liệu là cách thức con người thực hiện khi cần giải quyết một vấn đề trong cuộc sống.
- Khi làm việc với lượng dữ liệu lớn, một cách tổ chức dữ liệu tốt sẽ tiết kiệm chi phí.
  - ◇ Dung lượng (space).

# Tại sao quan tâm?

- Tổ chức dữ liệu là cách thức con người thực hiện khi cần giải quyết một vấn đề trong cuộc sống.

- Khi làm việc với lượng dữ liệu lớn, một cách tổ chức dữ liệu tốt sẽ tiết kiệm chi phí.
  - ◇ Dung lượng (space).
  - ◇ Thời gian.

# Tại sao quan tâm?

- Tổ chức dữ liệu là cách thức con người thực hiện khi cần giải quyết một vấn đề trong cuộc sống.

- Khi làm việc với lượng dữ liệu lớn, một cách tổ chức dữ liệu tốt sẽ tiết kiệm chi phí.
  - ◇ Dung lượng (space).
  - ◇ Thời gian.
  - ◇ Công sức lập trình và bảo trì.

# Tại sao quan tâm?

- Tổ chức dữ liệu là cách thức con người thực hiện khi cần giải quyết một vấn đề trong cuộc sống.
- Khi làm việc với lượng dữ liệu lớn, một cách tổ chức dữ liệu tốt sẽ tiết kiệm chi phí.
  - ◇ Dung lượng (space).
  - ◇ Thời gian.
  - ◇ Công sức lập trình và bảo trì.
- Là tiền đề, chìa khoá để áp dụng các giải thuật giúp giải quyết bài toán.

# Các kiểu cấu trúc dữ liệu cơ bản

- Mảng (array)

# Các kiểu cấu trúc dữ liệu cơ bản

- Mảng (array)
- Danh sách liên kết (links lists)

# Các kiểu cấu trúc dữ liệu cơ bản

- Mảng (array)
- Danh sách liên kết (links lists)
- Ngăn xếp (stack)

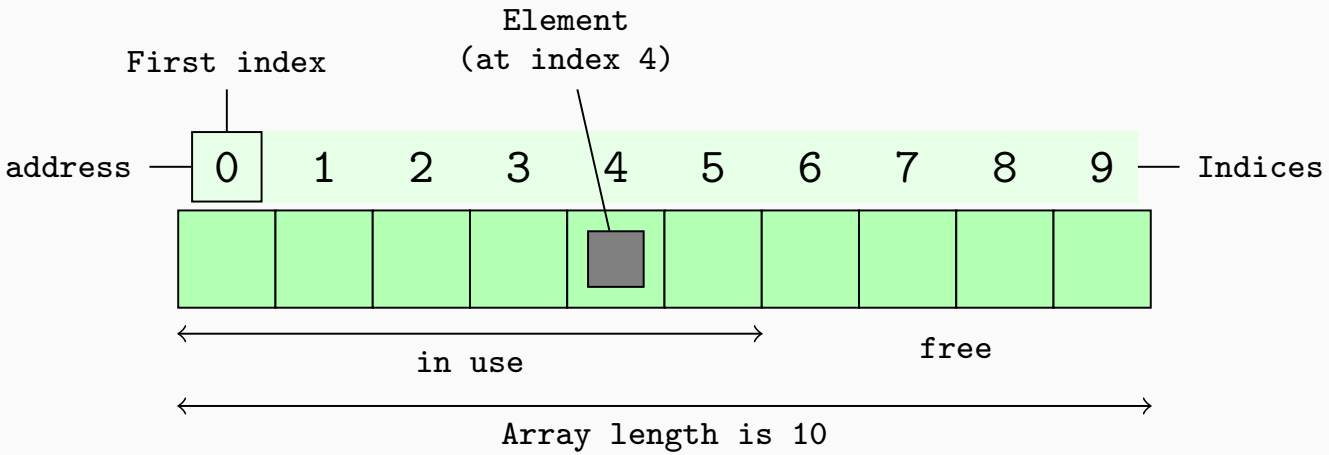
# Các kiểu cấu trúc dữ liệu cơ bản

- Mảng (array)
- Danh sách liên kết (links lists)
- Ngăn xếp (stack)
- Hàng đợi (queues)

# Các kiểu cấu trúc dữ liệu cơ bản

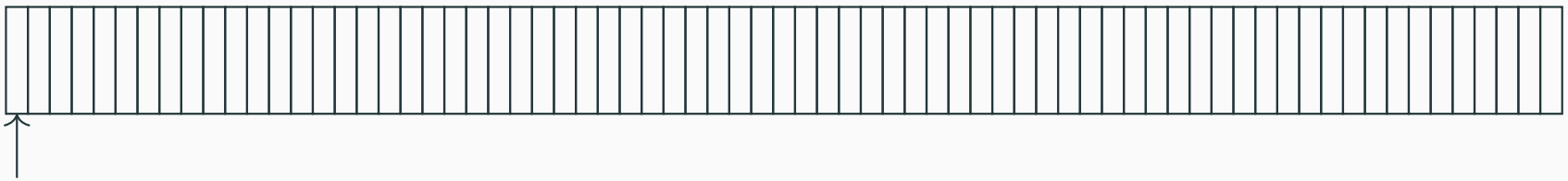
- Mảng (array)
- Danh sách liên kết (links lists)
- Ngăn xếp (stack)
- Hàng đợi (queues)
- Cây (trees)

# Mảng (array)


Element
(at index 4)
First index
address
0
1
2
3
4
5
6
7
8
9
Indices
in use
free
Array length is 10

Programmer

Memory

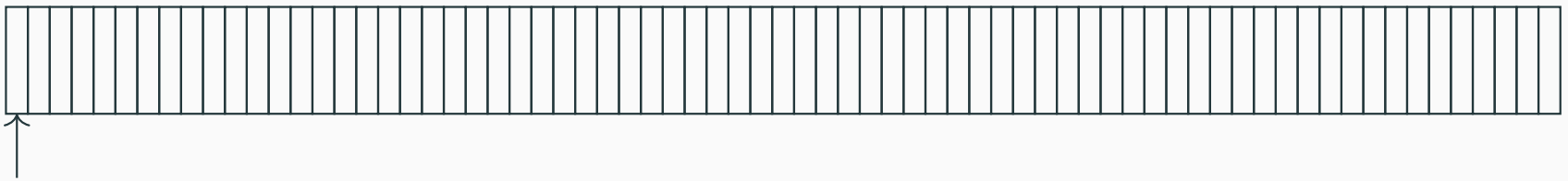


102

Programmer

```
int x;
```

Memory



102

Programmer

```
int x;
```

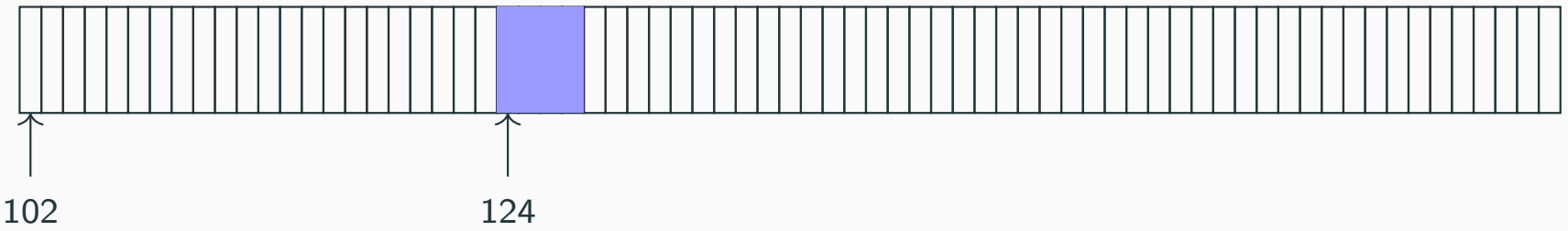
Memory

102


Programmer

```
int x;
```

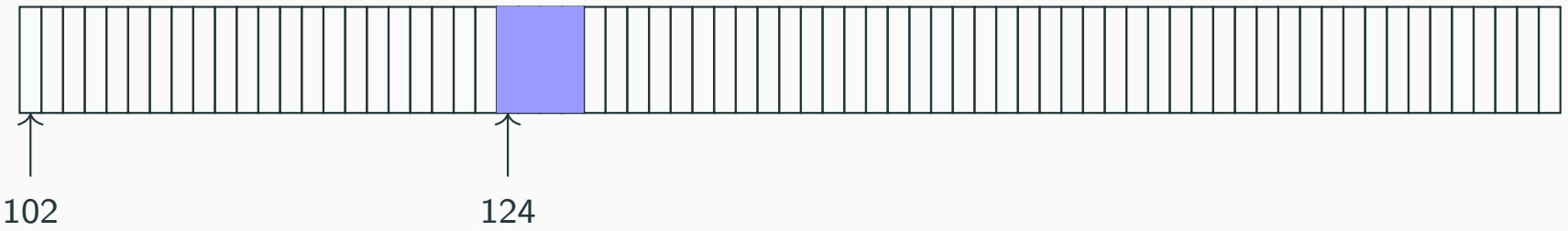
Memory




Programmer

```
int x;
x=65;
```

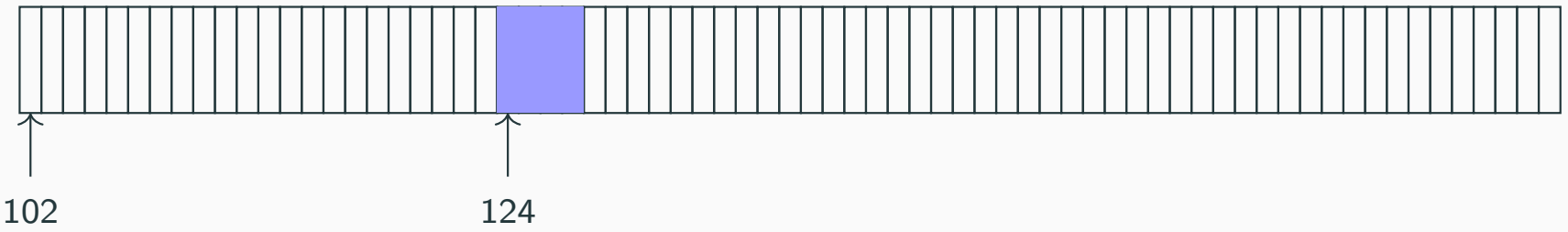
Memory




Programmer

```
int x;
x=65;
```

Memory

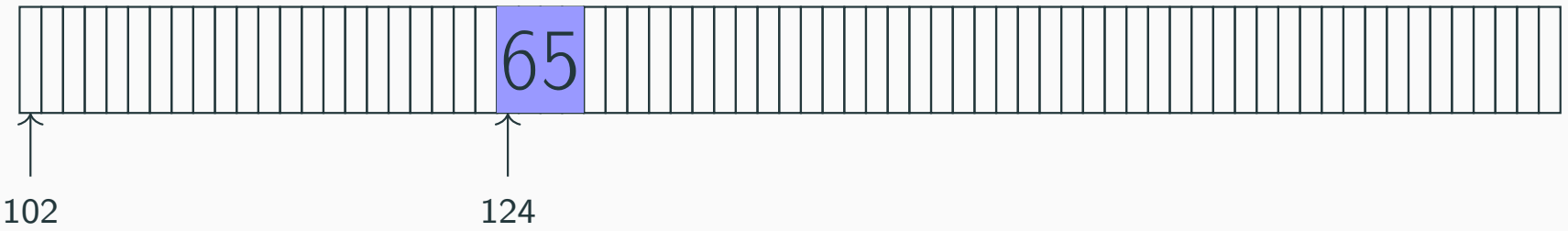



so easy

Programmer

```
int x;
x=65;
```

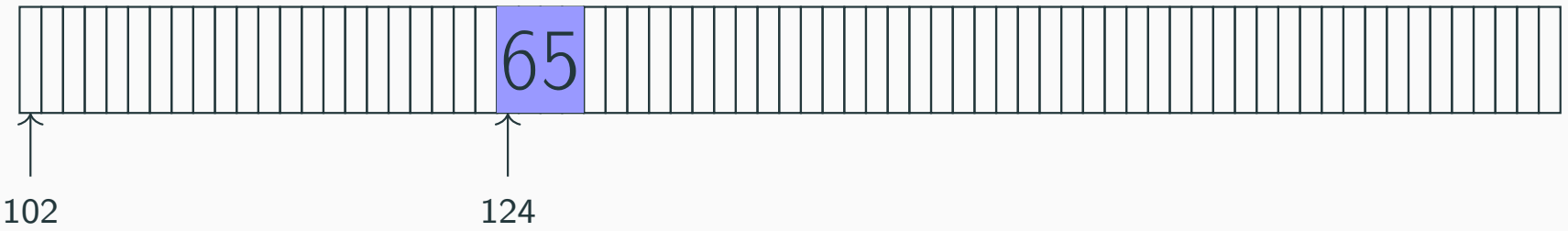
Memory




Programmer

```
int x;
x=65;
int A[4];
```

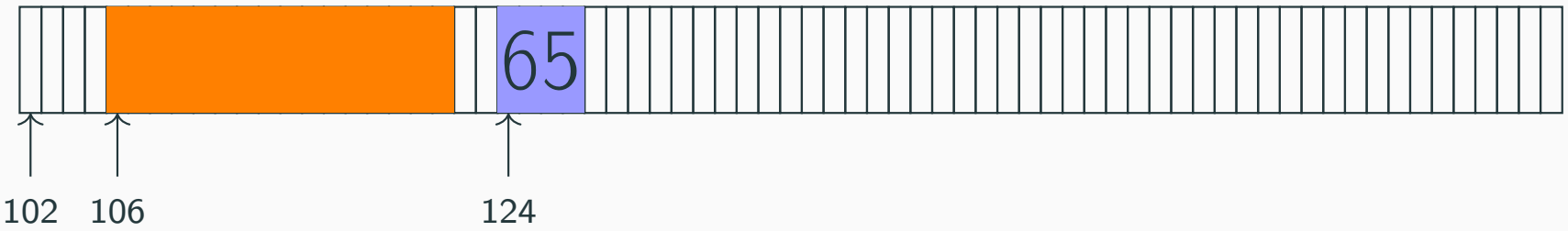
Memory




Programmer

```
int x;
x=65;
int A[4];
```

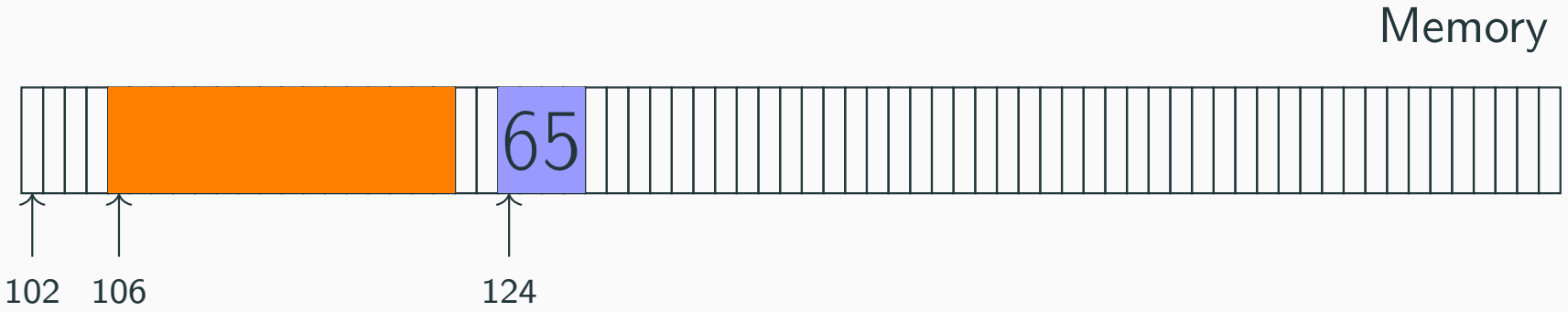
Memory





Programmer

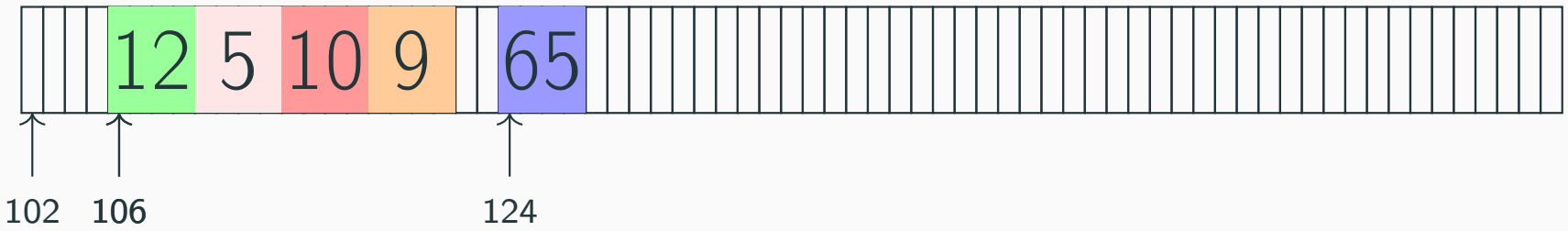
```
int x;
x=65;
int A[4];
A[0]=12; A[1]=5;
A[2]=10; A[3]=9;
```

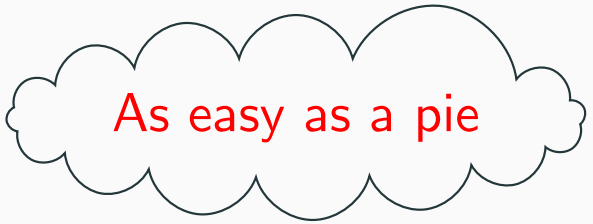



Programmer

```
int x;
x=65;
int A[4];
A[0]=12; A[1]=5;
A[2]=10; A[3]=9;
```

Memory

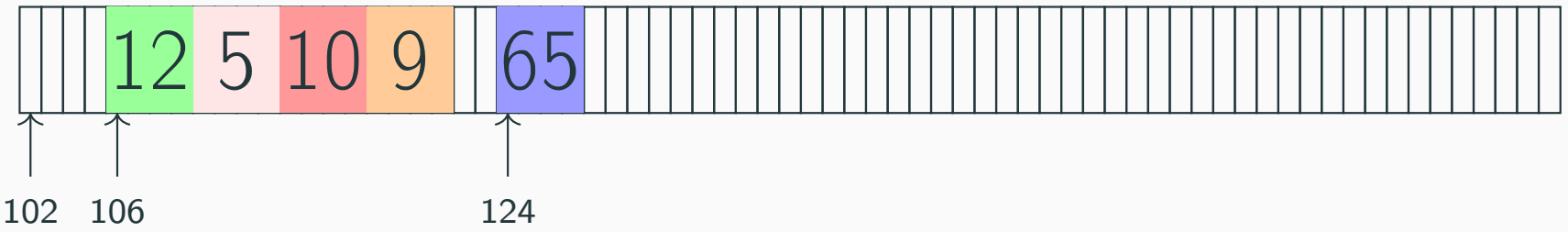



Programmer

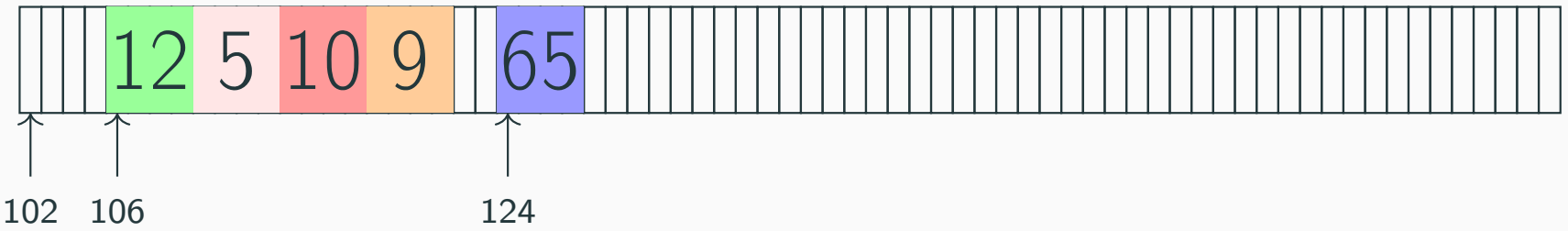
Can I extend
the array A?

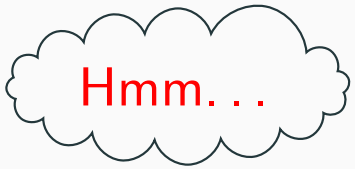
Memory




Programmer

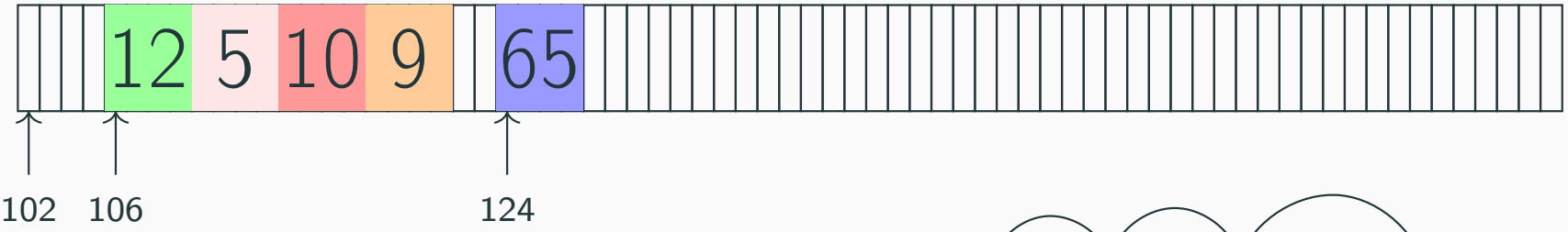
Memory




Programmer

Memory

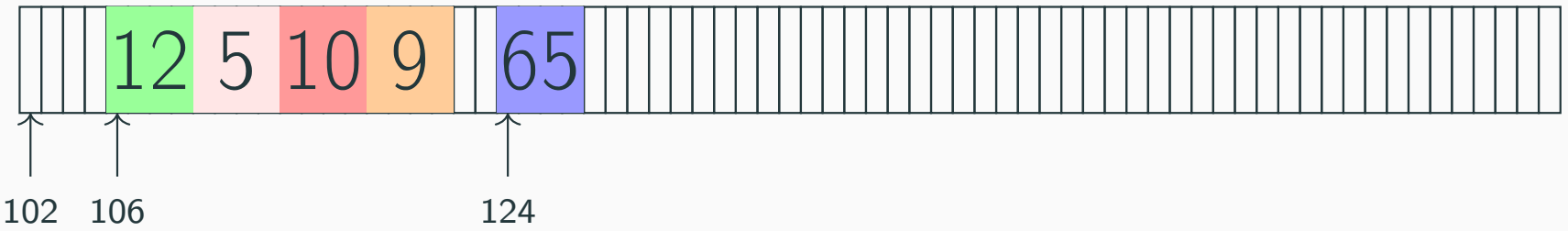



Nope!!! It's fixed size.

What all option do I have?

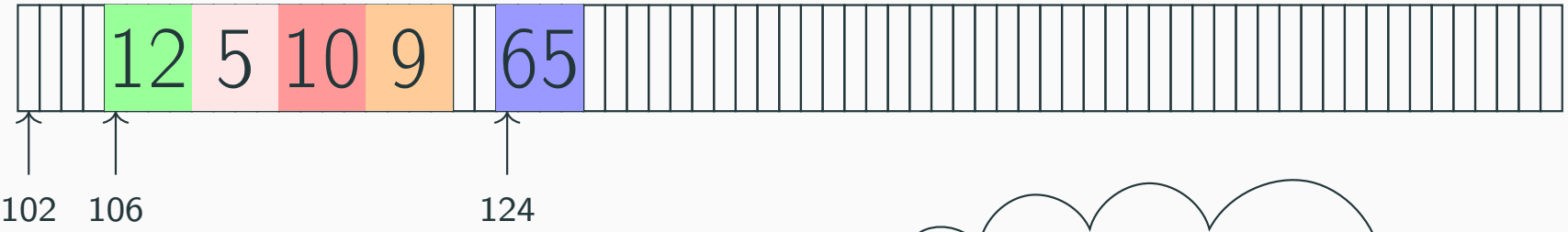
Programmer

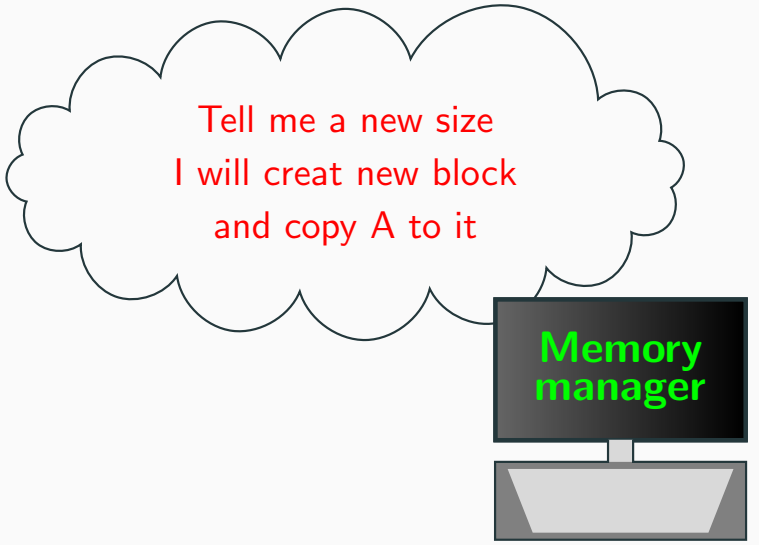
Memory




Programmer

Memory



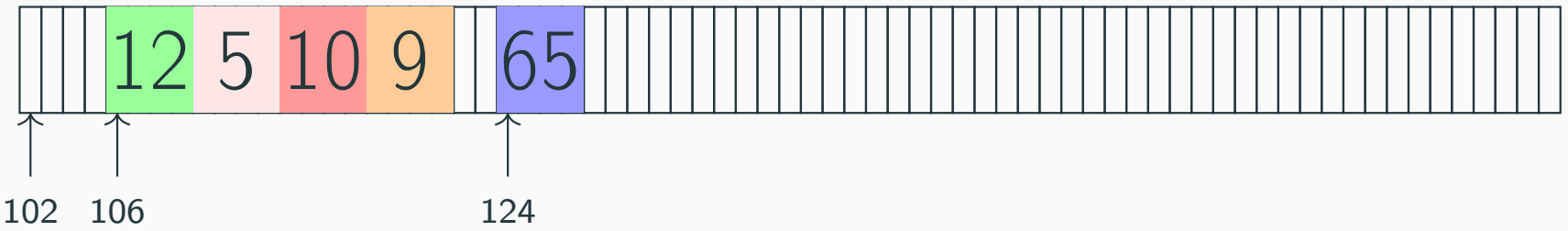



Programmer

Yes! Let do it.

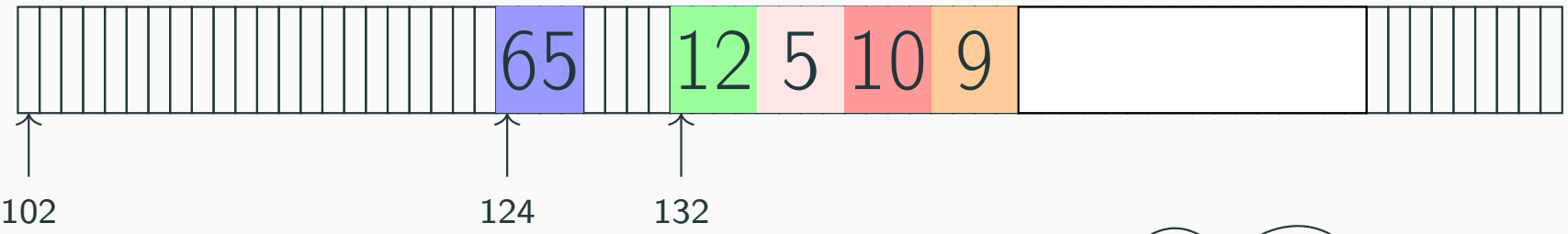
Double it's size

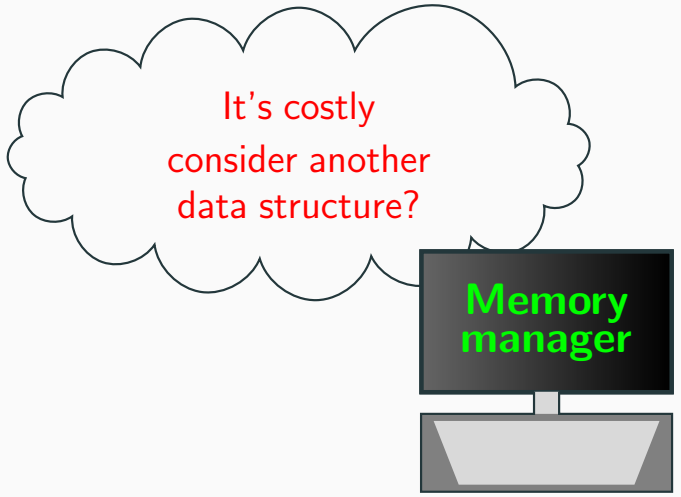
Memory




Programmer

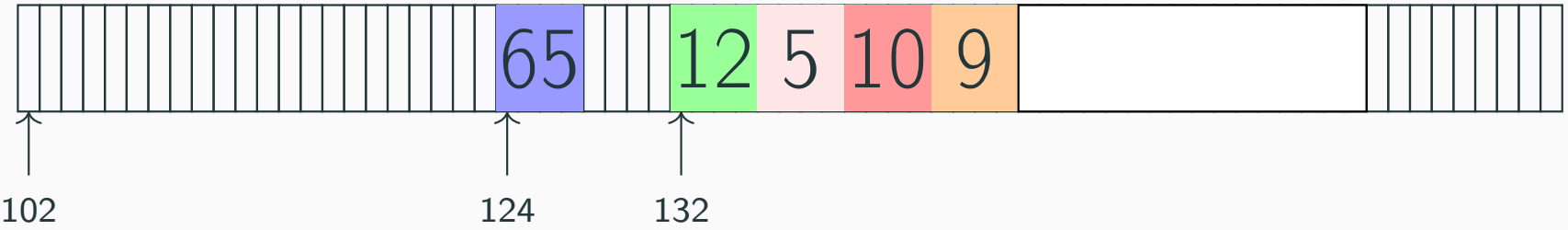
Memory



Programmer

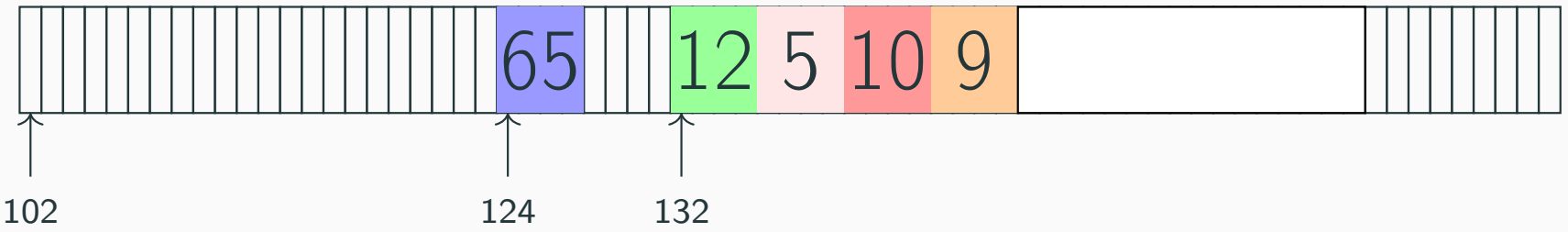
Memory



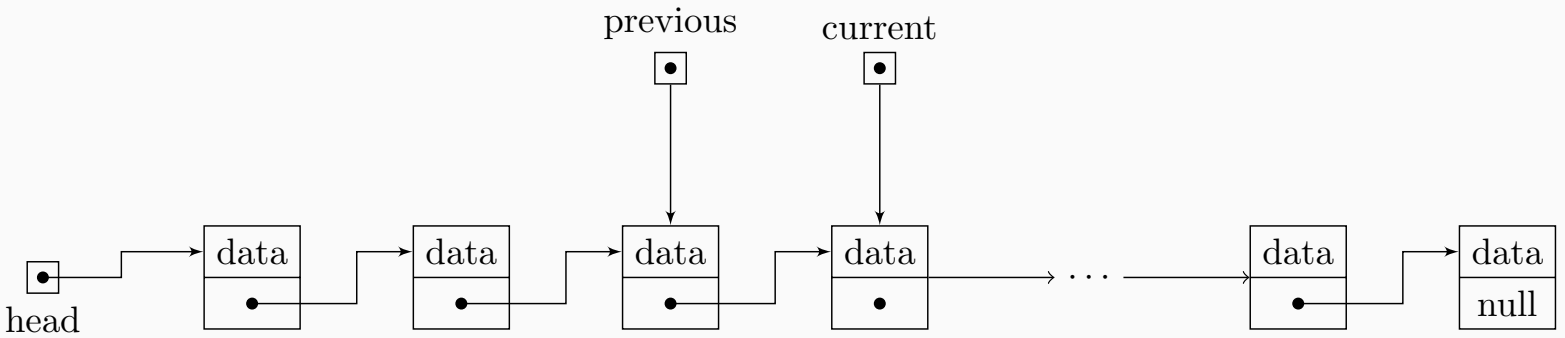

Yes, sir!!!

Programmer

Memory




# Danh sách liên kết



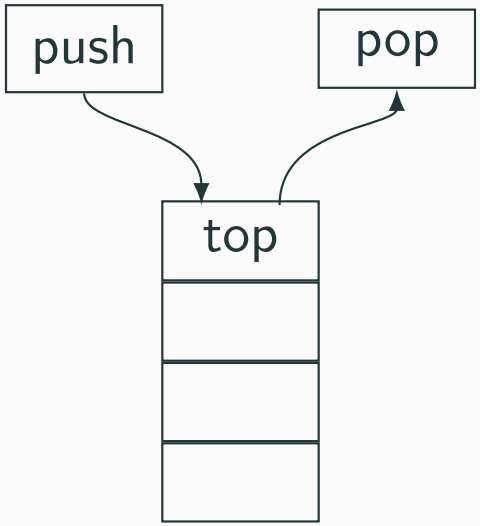

**Listing 1:** Simple node

```
1    struct node{
2        int x;
3        node *next;
4    };
```

# So sánh các kiểu dữ liệu danh sách

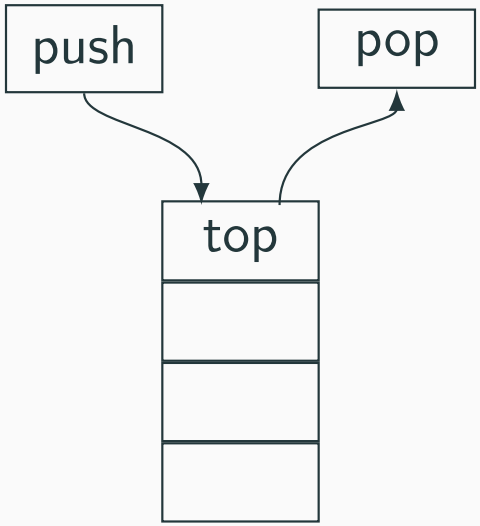
| | Linked list | **Array** | **Dynamic array** |
|---|---|---|---|
| Indexing | O(n) | O(1) | O(1) |
| Insert/delete at beginning | O(1) | N/A | O(n) |
| Insert/delete at end | O(1) when last element is known;<br>O(n) when last element is unknown | N/A | O(1) amortized |
| Insert/delete in middle | search time + (1) | N/A | O(n) |
| Wasted space (average) | O(n) | 0 | O(n) |

# Ngăn xếp (stack)



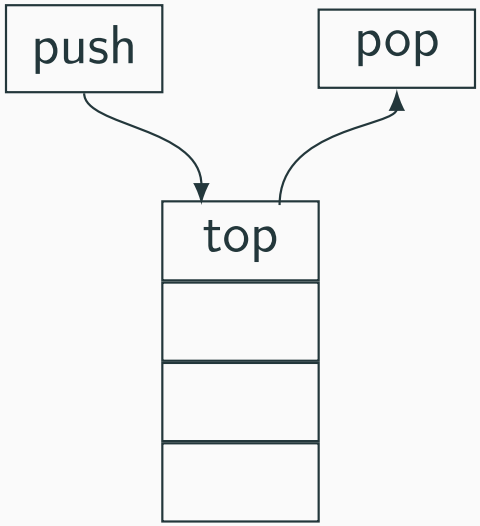

- Con người dùng cấu trúc dữ liệu stack khi giải quyết một vấn đề.

# Ngăn xếp (stack)



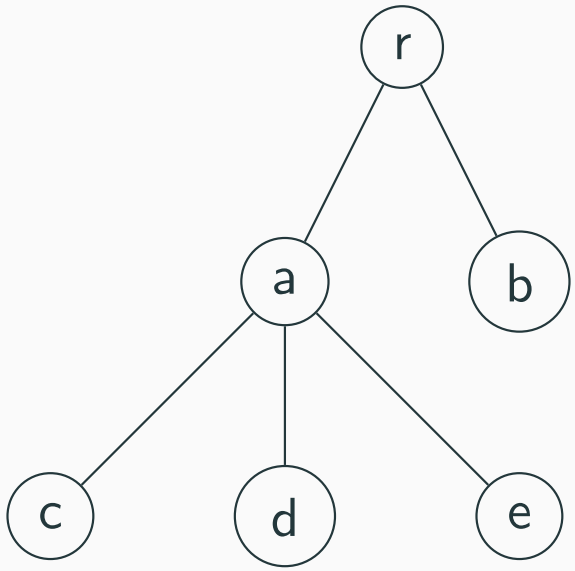

- Con người dùng cấu trúc dữ liệu stack khi giải quyết một vấn đề.
- Một vùng nhớ đặc biệt của máy tính, nơi lưu trữ các biến tạm thời được tạo ra trong hàm (bao gồm cả hàm main).

# Ngăn xếp (stack)




- Con người dùng cấu trúc dữ liệu stack khi giải quyết một vấn đề.
- Một vùng nhớ đặc biệt của máy tính, nơi lưu trữ các biến tạm thời được tạo ra trong hàm (bao gồm cả hàm main).
- Dùng nhiều trong thuật toán đệ quy.

# Cây



# Các thuật toán sắp xếp

# Selection sort

- Input: Mảng gồm $n$ phần tử $A[1 \ldots n]$.
  Output: Mảng $A[1...n]$ đã được sắp thứ tự.
- Ý tưởng: Chọn phần tử nhỏ nhất (hoặc lớn nhất) trong dãy xếp vào vị trí đầu tiên (hoặc cuối cùng), tương tự với phần tử thứ 2,3... lặp lại quá trình này với các phần tử còn lại.

# Selection sort's pseudo-code

Selection sort

```
1: for i ← 1 to N do
2:     min ← i
3:     for j ← i + 1 to N do
4:         if A[j] < A[min] then
5:             min ← j
6:     swap(A[i], A[min])
```

# Selection sort

Độ phức tạp tính toán:

- Thuật toán ít phải đổi chỗ các phần tử ($n$ lần đổi chỗ), số phép so sánh luôn là $O(n^2)$.
- Thuật toán tốn thời gian gần như bằng nhau đối với mảng đã được sắp xếp cũng như mảng chưa được sắp xếp.

# Insection sort

- Input: Mảng gồm n phần tử $A[1...n]$.
  Output: Mảng $A[1...n]$ đã được sắp thứ tự.
- Ý tưởng: Chèn một phần tử đang xét của dãy vào một dãy con đã được sắp xếp.

# Insertion sort's pseudo-code

Insertion sort

```
1: for j ← 2 to N do
2:     key ← A[j]
3:     i ← j − 1
4:     while i > 0 and A[i] > key do
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     A[i + 1] ← key
```

# Insection sort

Độ phức tạp tính toán:

- Trường hợp tốt nhất: $n - 1$ phép so sánh, 0 lần đổi chỗ.
- TH trung bình: $n^2/2$ phép so sánh, $n^2/4$ lần đổi chỗ.
- TH xấu nhất: $n^2/2$ phép so sánh, $n^2/2$ lần đổi chỗ.

# Mergesort

Ý tưởng:

Chia: Chia dãy $n$-phần tử cần sắp xếp thành hai dãy con, mỗi dãy $n/2$ phần tử.

Trị: Sắp xếp hai dãy con bằng đệ quy sử dụng merge sort.

Hợp: Dính hay dãy con lại với nhau để thu được dãy sắp xếp hoàn chỉnh.

# Merge sort's pseudo-code

Merge sort

```
1:  n_1 ← q − p + 1
2:  n_2 ← r − q
3:  for i ← 1 to n_1 do
4:      L[i] ← A[p + i − 1]
5:  for j ← 1 to n_2 do
6:      R[j] ← A[q + j]
7:  L[n_1 + 1] ← ∞
8:  R[n_2 + 1] ← ∞
9:  i ← 1
10: j ← 1
11: for k ← p to r do
12:     if L[i] ≤ R[j] then
13:         A[k] ← L[i]
14:         i ← i + 1
15:     else
16:         A[k] ← R[j]
17:         j ← j + 1
```

# Mergesort's java-code

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
   for (int k = lo; k <= hi; k++)
      aux[k] = a[k];

   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)
   {
      if      (i > mid)              a[k] = aux[j++];
      else if (j > hi)               a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                           a[k] = aux[i++];
   }
}
```

copy

merge

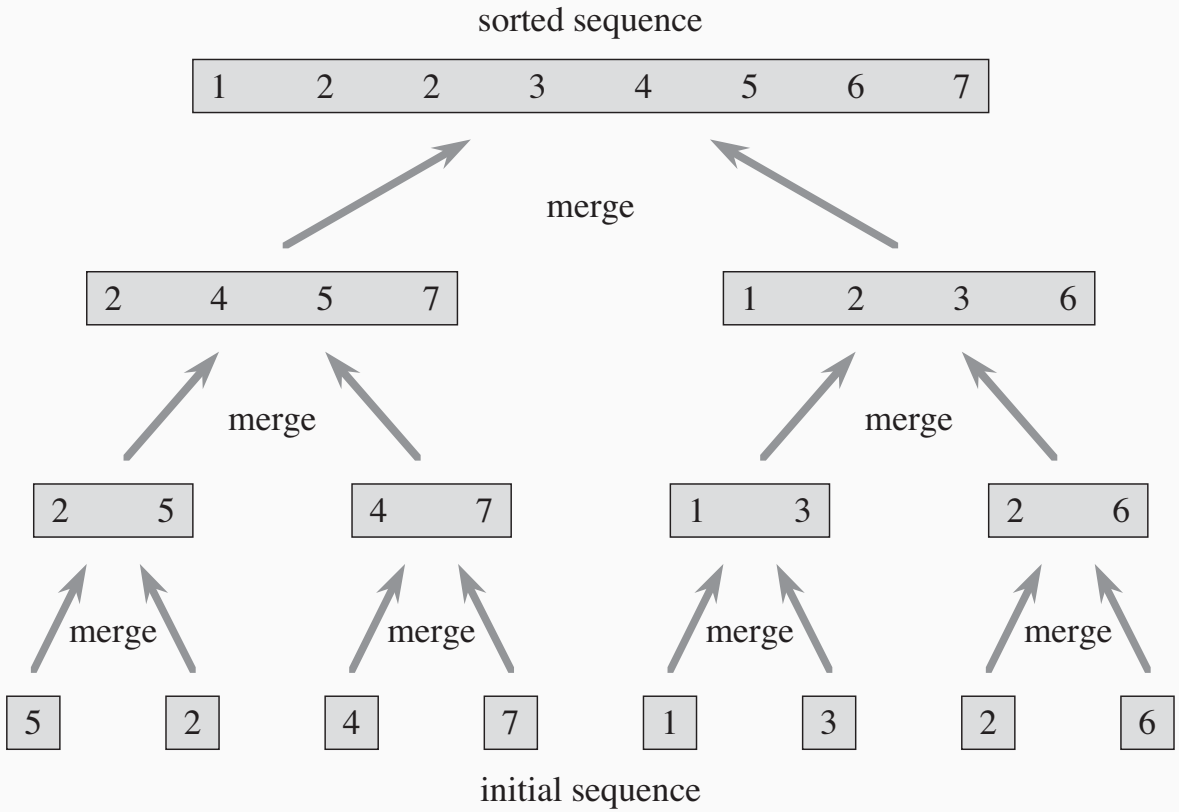
# Merge sort's main pseudo-code

Merge sort$(A, p, r)$

1: **if** $p<r$ **then**
2: $\quad q = \lfloor (p + r)/2 \rfloor$
3: $\quad$ MERGE-SORT$(A, p, q)$
4: $\quad$ MERGE-SORT $(A, q + 1, r)$
5: $\quad$ MERGE-SORT $(A, p, q, r)$

# Mergesort trace

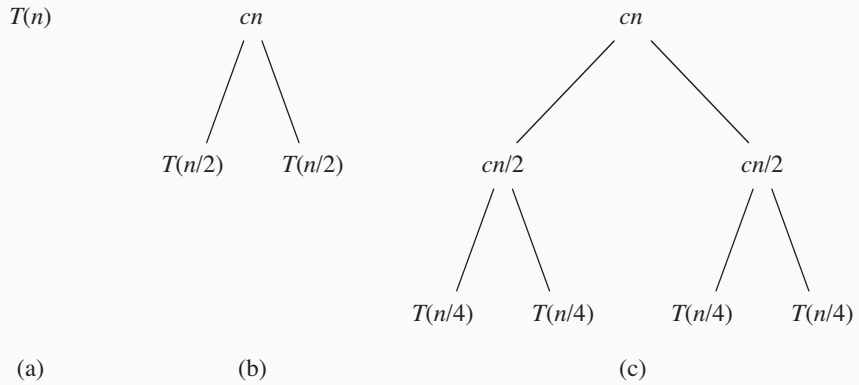
# Mergesort's pseudo-code

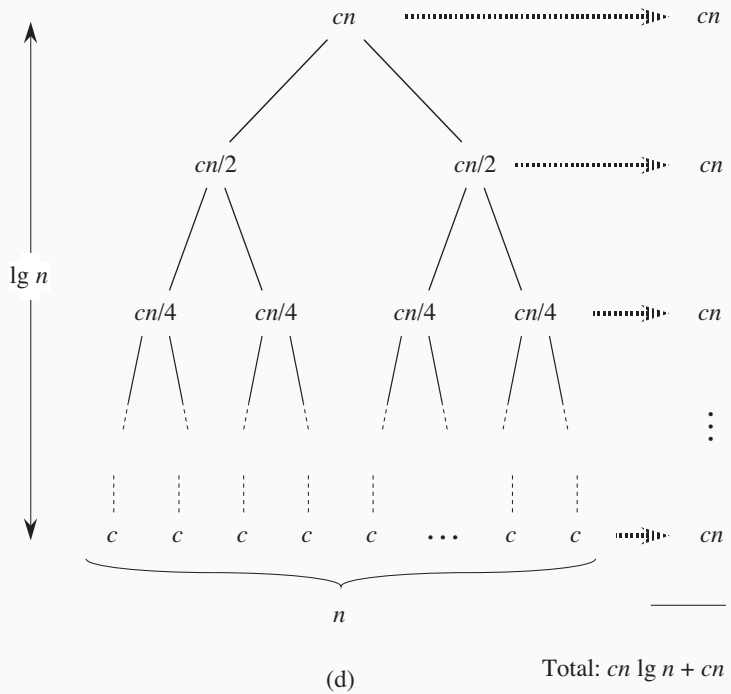



Cách tính thời gian với giải thuật sử dụng đệ quy:

$$T(n) = \begin{cases} \Theta(n) & \text{nếu } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{còn lại.} \end{cases}$$

# Mergesort's running time



# Mergesort: real running time

Thời gian chạy ước lượng:

Laptop thực hiện $10^8$ so sánh/giây.

Siêu máy tính thực hiện $10^{12}$ so sánh/giây.

| | insertion sort ($N^2$) | | | mergesort (N log N) | | |
|---|---|---|---|---|---|---|
| computer | thousand | million | billion | thousand | million | billion |
| home | instant | 2.8 hours | 317 years | instant | 1 second | 18 min |
| super | instant | 1 second | 1 week | instant | instant | instant |

# Quicksort: Ý tưởng

Ý tưởng:

Chia: Phân bổ (đổi chỗ các phần tử) mảng $A[p \dots r]$ thành hai mảng $A[p \dots q-1]$ và $A[q+1 \dots r]$, sao cho các phần tử của mảng $A[p \dots q-1]$ đều nhỏ hơn hoặc bằng $A[q]$ và các phần tử của mảng $A[q+1 \dots r]$ đều lớn hơn hoặc bằng $A[q]$.

Trị: Sắp xếp hai dãy con bằng đệ quy sử dụng quick sort.

Hợp: Vì các mảng con đã được sắp xếp và $A[q]$ nằm đúng vị trí của nó trong mảng nên mảng $A[p \dots r]$ được sắp xếp.

# Quicksort's pseudo-code

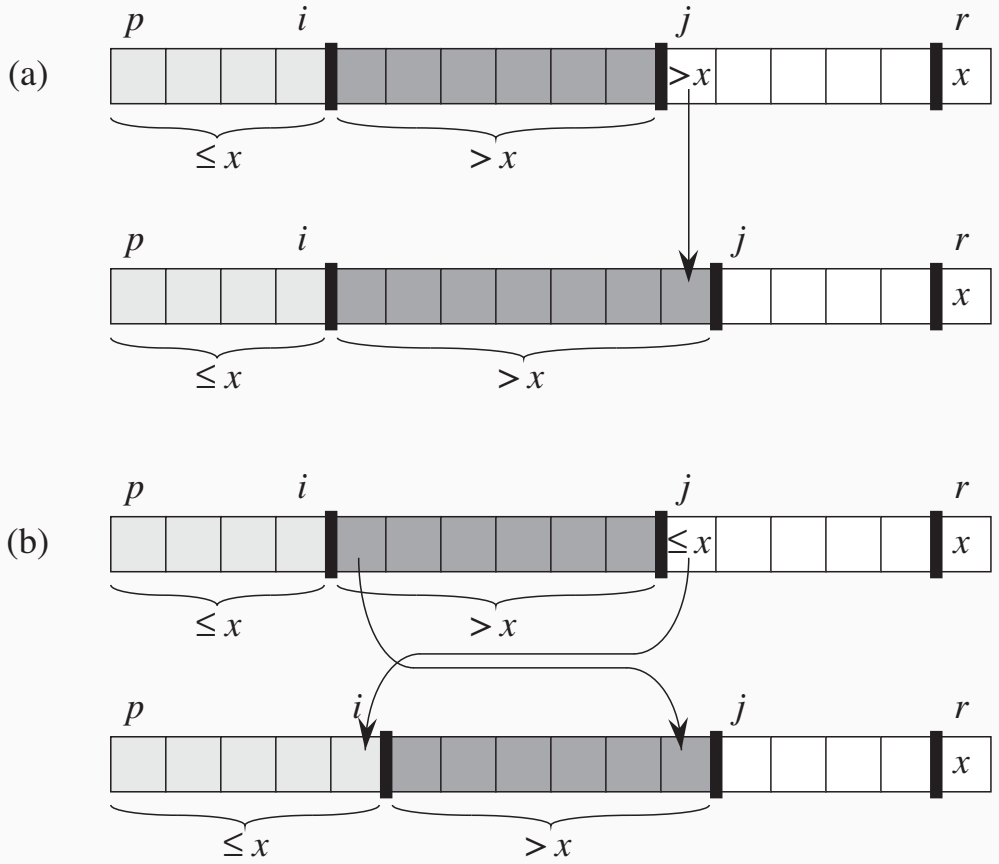
PARTITION($A, p, r$)

1: $x = A[r]$
2: $i = p - 1$
3: **for** $j = p$ to $r - 1$ **do**
4: **if** $A[j] \leq x$ **then**
5: $i = i + 1$
6: exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
7: **return** $i + 1$

# Quicksort's main pseudo-code

Quick sort$(A, p, r)$

1: **if** $p<r$ **then**
2: $q =$PARTITION$(A, p, r)$
3: QUICKSORT$(A, p, q-1)$
4: QUICKSORT $(A, q+1, r)$


(a)
p
i
j
r
> x
x
≤ x
> x
p
i
j
r
x
≤ x
> x
(b)
p
i
j
r
≤ x
x
≤ x
> x
p
i
j
r
x
≤ x
> x

# Quicksort: real running time

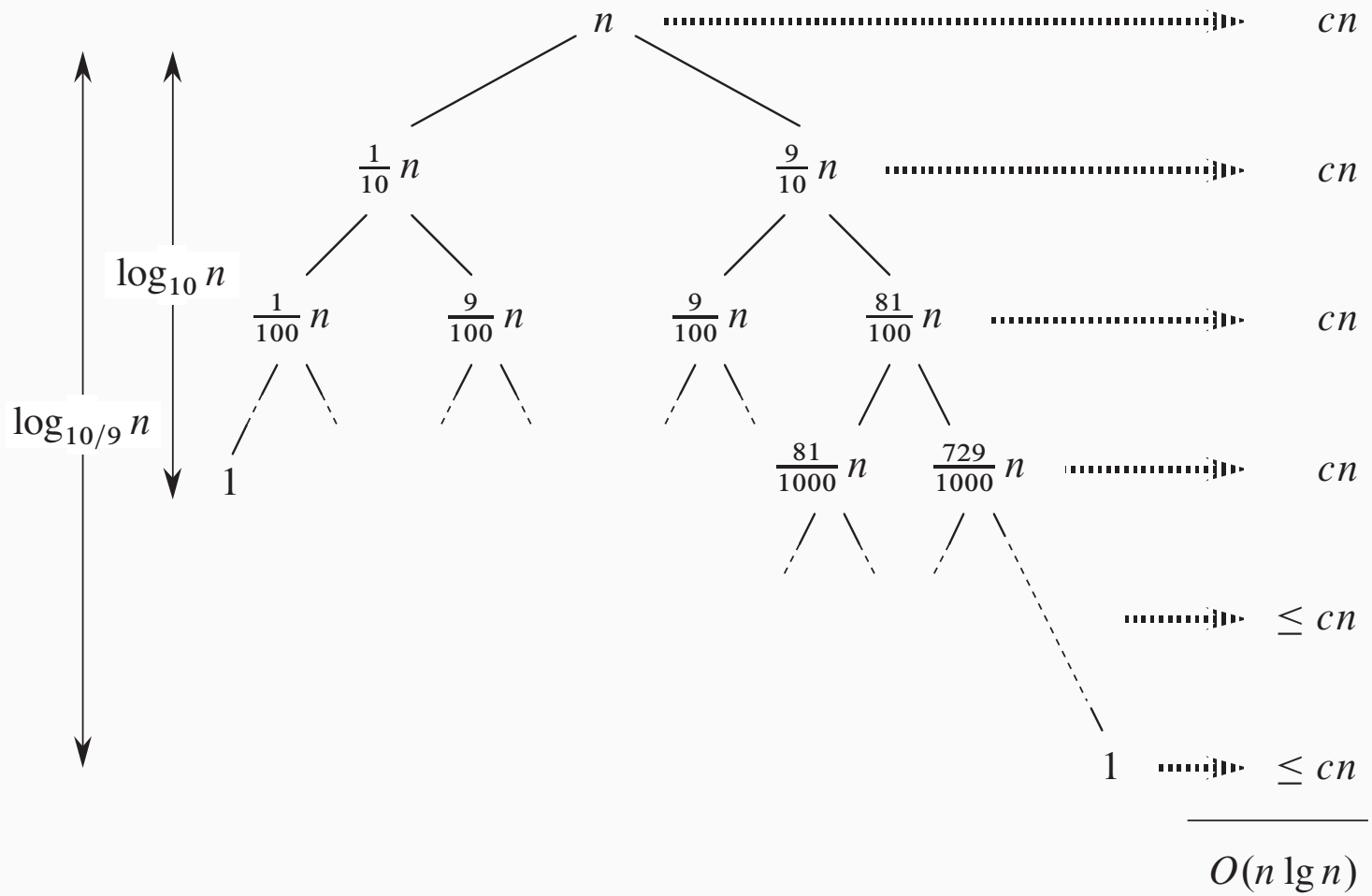
Thời gian chạy ước lượng:

Laptop thực hiện $10^8$ so sánh/giây.

Siêu máy tính thực hiện $10^{12}$ so sánh/giây.

| | insertion sort ($n^2$) | | | mergesort (n log n) | | | quicksort (n log n) | | |
|---|---|---|---|---|---|---|---|---|---|
| computer | thousand | million | billion | thousand | million | billion | thousand | million | billion |
| home | instant | 2.8 hours | 317 years | instant | 1 second | 18 min | instant | 0.6 sec | 12 min |
| super | instant | 1 second | 1 week | instant | instant | instant | instant | instant | instant |

# Quicksort's running time complexity




Cách tính thời gian với giải thuật sử dụng chia để trị:

$$T(n) = T(9n/10) + T(n/10) + cn.$$

# Quicksort: average-case analysis

Proposition. The average number of compares $C_n$ to quicksort an array of $n$ distinct keys is $\sim 2n \ln n$ (and the number of exchanges is $\sim \frac{1}{3} n \ln n$ ).

Pf. $C_n$ satisfies the recurrence $C_0 = C_1 = 0$ and for $n \geq 2$:

partitioning; left; right

$$C_N = (N+1) + \left(\frac{C_0 + C_{N-1}}{N}\right) + \left(\frac{C_1 + C_{N-2}}{N}\right) + \ldots + \left(\frac{C_{N-1} + C_0}{N}\right)$$

partitioning probability

- Multiply both sides by $n$ and collect terms:

$$NC_N = N(N+1) + 2(C_0 + C_1 + \ldots + C_{N-1})$$

- Subtract from this equation the same equation for $n - 1$:

$$NC_N - (N-1)C_{N-1} = 2N + 2C_{N-1}$$

- Rearrange terms and divide by $n\,(n+1)$:

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}$$

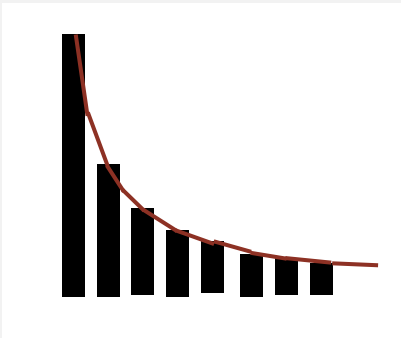
# Quicksort: average-case analysis

- Repeatedly apply previous equation:

$$\begin{aligned}
\frac{C_N}{N+1} &= \frac{C_{N-1}}{N} + \frac{2}{N+1} \\
&= \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \qquad \longleftarrow \text{substitute previous equation} \\
&= \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\
&= \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \ldots + \frac{2}{N+1}
\end{aligned}$$

- Approximate sum by an integral:

$$\begin{aligned}
C_N &= 2(N+1)\left(\frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots \frac{1}{N+1}\right) \\
&\sim 2(N+1)\int_3^{N+1} \frac{1}{x}\, dx
\end{aligned}$$



- Finally, the desired result:

$$C_N \sim 2(N+1)\ln N \approx 1.39 N \lg N$$