# CS101C: Introduction to Programming (Using C)

Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

## Week7: Functions, Sorting

# This week

- Application of pointers
    a. Modular programming (functions)
- Call-by-value, Call-by-reference
- Global variables, static variables
- Sorting

# Functions

- You have seen functions `main`, `printf`, `scanf`, `pow` (anybody?)
  - `int main()`
  - `printf(“My name is %s”,name);`
  - `scanf(“%d”,&x);`

Return values, function name, function arguments and parameters

# Functions

- Let us define our own function to swap.

```
void swap(int a, int b){
    int tmp = a;
    a = b;
    b=tmp;
    return;
}
```

Function parameters, return statement, void type

# Functions

- Let us call the function swap from main.

```
int main(){
    int a=10;
    int b=20;
    swap(a, b);
    printf(“a=%d, b=%d”,a,b);
    return 0;
}
```

Function call / call site.

## Functions

```
int main(){
        int a=10;
        int b=20;
        swap(a, b);
        printf(“a=%d b=%d”,a,b); // prints a=10, b=20
}
```

**Call-by-value**

# Functions

- Let us define another version of the function to swap.

```
void swap2(int *a, int *b){
    int tmp = *a;
    *a = *b;
    *b=tmp;
    return;
}
```

Function parameters, return statement, void type

## Functions

```
int main(){
        int a=10;
        int b=20;
        swap2(&a, &b);
        printf("a=%d b=%d",a,b); // prints a=20, b=10
}
```

**Call-by-reference**

## Functions Declaration vs. Function Definition

- `void swap2(int *a, int *b); //declaration`
- `//definition follows`

```
void swap2(int *a, int *b){
                int tmp = *a;
                *a = *b;
                *b=tmp;
                return;
              }
```

# Functions Declaration vs. Function Definition

- Why you need a declaration?

So that you do not have to define the function before the function call site.

  - You can define a function after the call site in the same .c file
  - You can define a function in another .c file!

# Global and Static variables

- Global variables are visible to all functions in the program. Initialized to zero
- Static variables are visible inside a single function only but retain their previous value when the function is called again

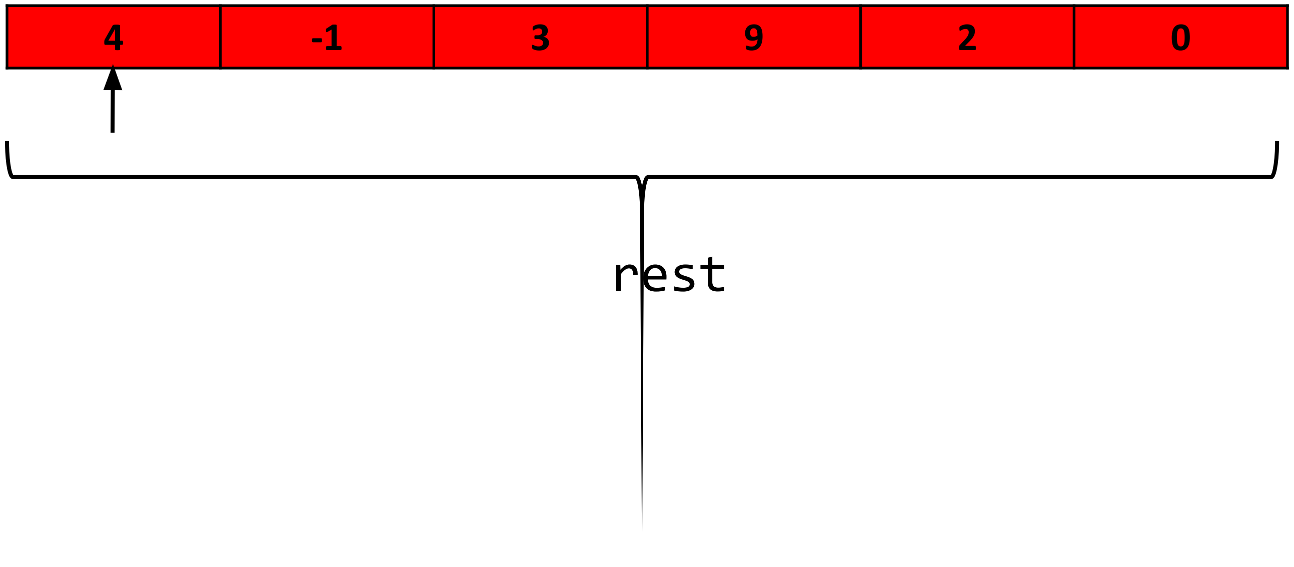
**see function.c shared in codeexample directory**

## Sorting - Selection sort

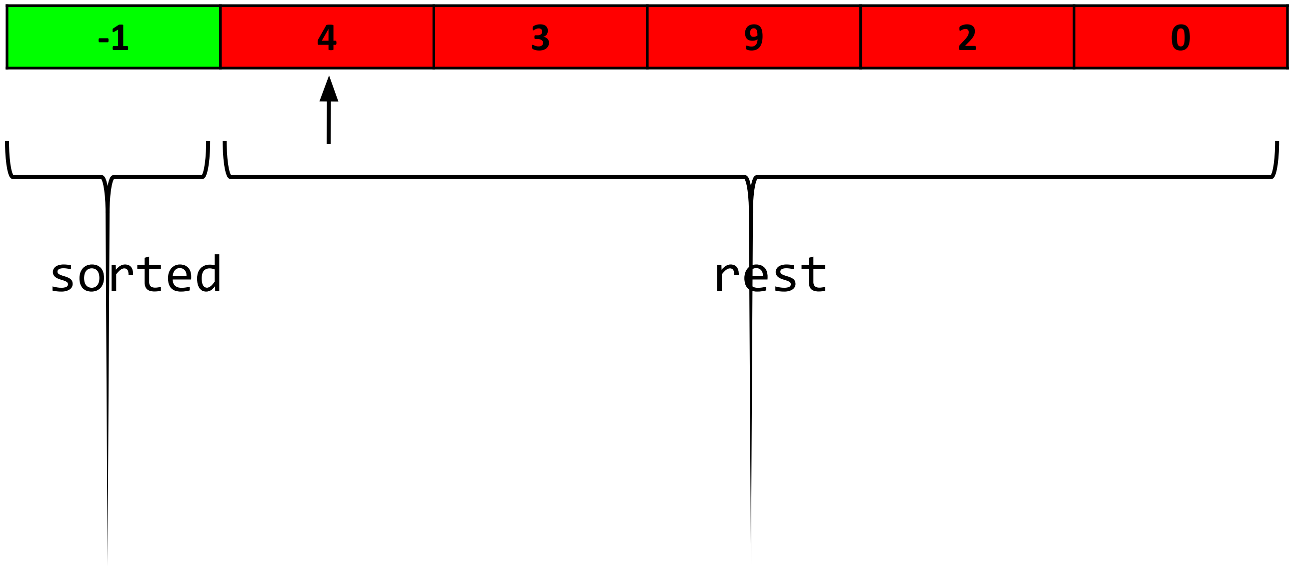
- Repeatedly find the minimum element in the unsorted array and put it at the beginning.
  - Divides the input array into 2 pieces - `sorted` and `rest`.
  - *All elements* in `sorted` are smaller than *any element* in the `rest` – *invariant*
  - Works by growing `sorted` and shrinking `rest`

## Selection sort - example

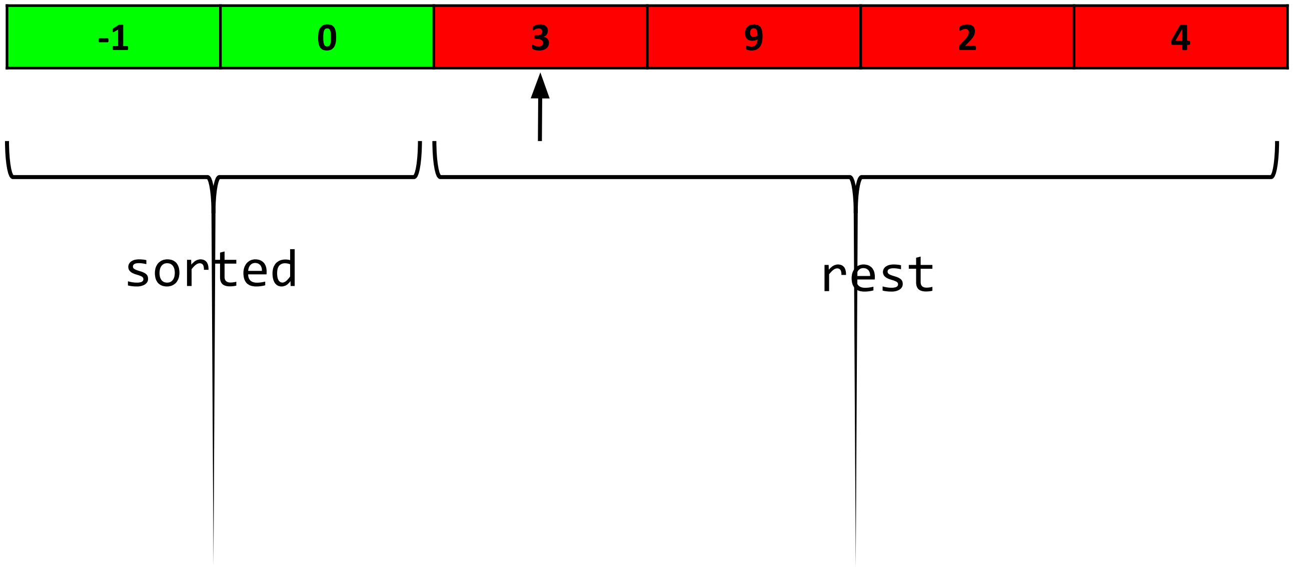
- A cursor dividing `sorted` and `rest`



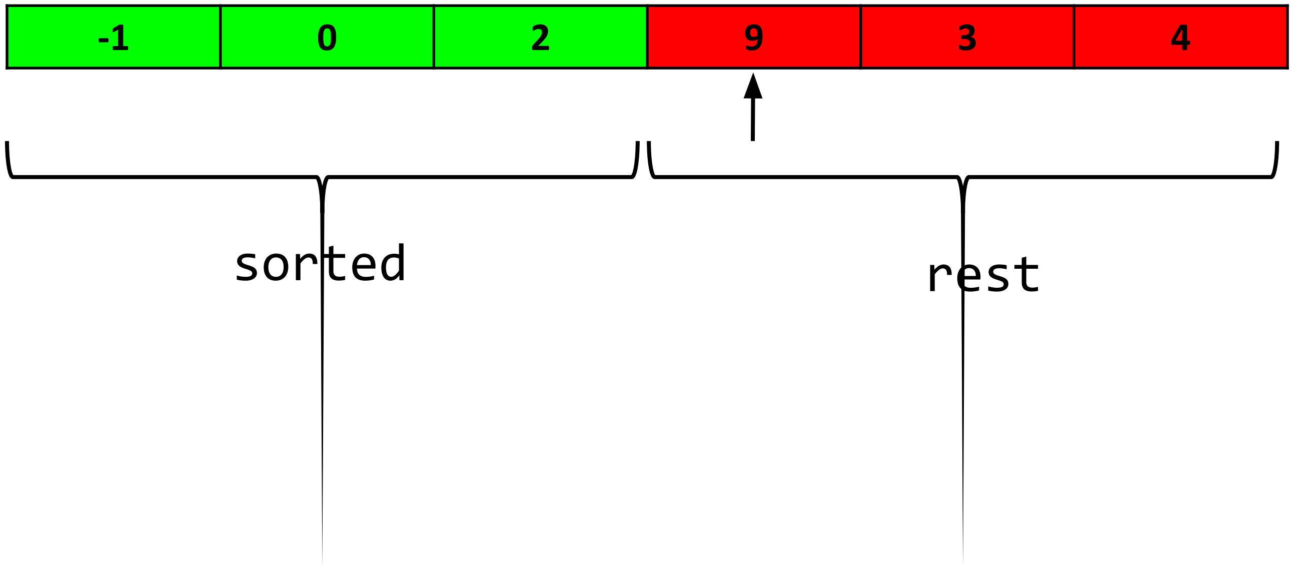
# Selection sort - example



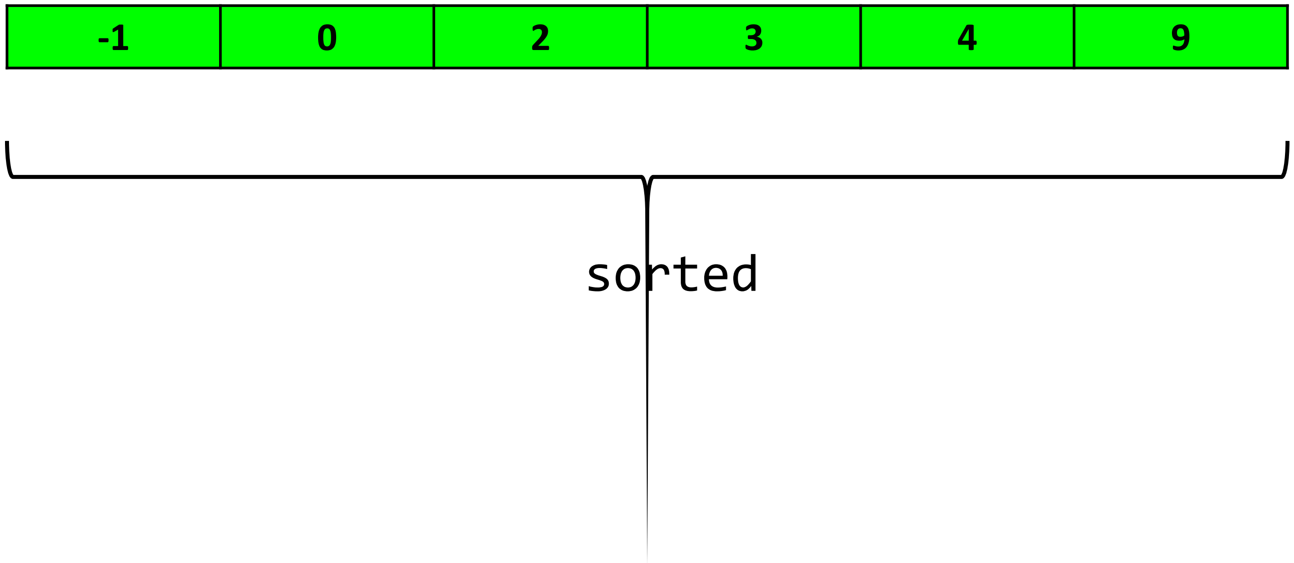
## Selection sort - example



# Selection sort - example



# Selection sort - example



## Sorting algorithms - Evaluation

- Many metrics used for evaluating sorting algorithms.
- Two most common metrics are:
    - How many comparisons are involved?
    - How much data movement is involved?

# Selection sort - pseudocode

```
int input[N] = //input
int cursor = 0 //initial position of the cursor
for(cursor = 0; cursor < N; cursor++)
  //sorted list from [0,cursor)
  //rest of the list from [cursor, N)
  for(i = cursor; i < N; i++)
    //search the rest of the list to find the smallest value
  //swap the smallest value with the value at input[cursor]
```

## Selection sort - Analysis

```
int input[N] = //input
int cursor = 0 //initial position of the cursor
for(cursor = 0; cursor < N; cursor++)
 //sorted list from [0,cursor)
 //rest of the list from [cursor, N)
 for(i = cursor; i < N; i++)
   //search the rest of the list to find the smallest value
 //swap the smallest value with the value at input[cursor]
```

## Selection sort - Analysis

```
int input[N] = //input
int cursor = 0 //initial position of the cursor
for(cursor = 0; cursor < N; cursor++)
 //sorted list from [0,cursor)
 //rest of the list from [cursor, N)
 for(i = cursor; i < N; i++)
   //search the rest of the list to find the smallest value
 //swap the smallest value with the value at input[cursor]
```

- inner loop runs N times, (N - cursor) iterations every time.

$$= \sum_{i=0}^{N-1} N - i$$

$$= \sum_{i=1}^{N} i \quad = \frac{N\,(N+1)}{2}$$

## Selection sort - Analysis

- outer loop runs for N iterations
- inner loop runs for ~ N(N+1)/2 iterations
    - inner loop dominates

*1. Approximately how many array write operations occur?*

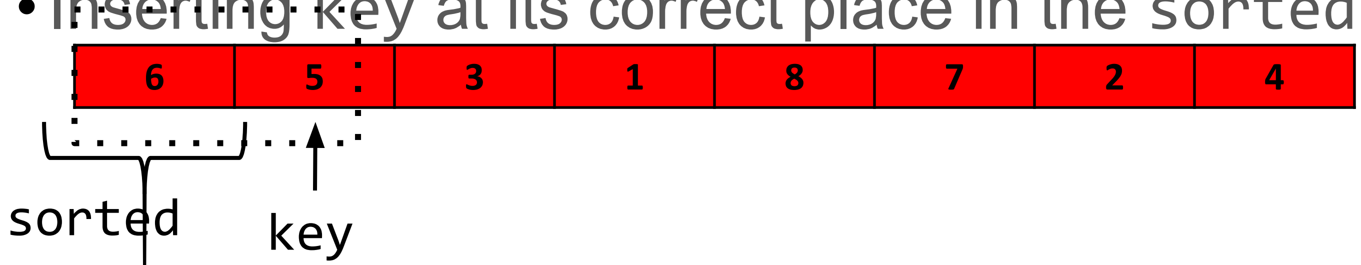
*2. Double the input, how long does Selection sort take?*

## Sorting - Insertion sort

- Iterate through the array one element at a time, build a sorted partial list.
  - Divides the input array into 2 pieces - `sorted`, `key`.
  - Inserts `key` into its right place in `sorted`.
  - Works by growing `sorted`

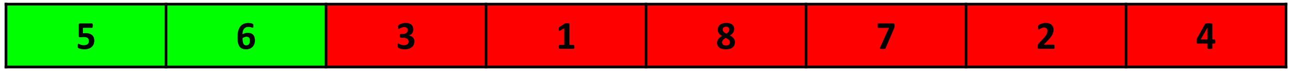
# Insertion sort - example

- Inserting key at its correct place in the sorted list

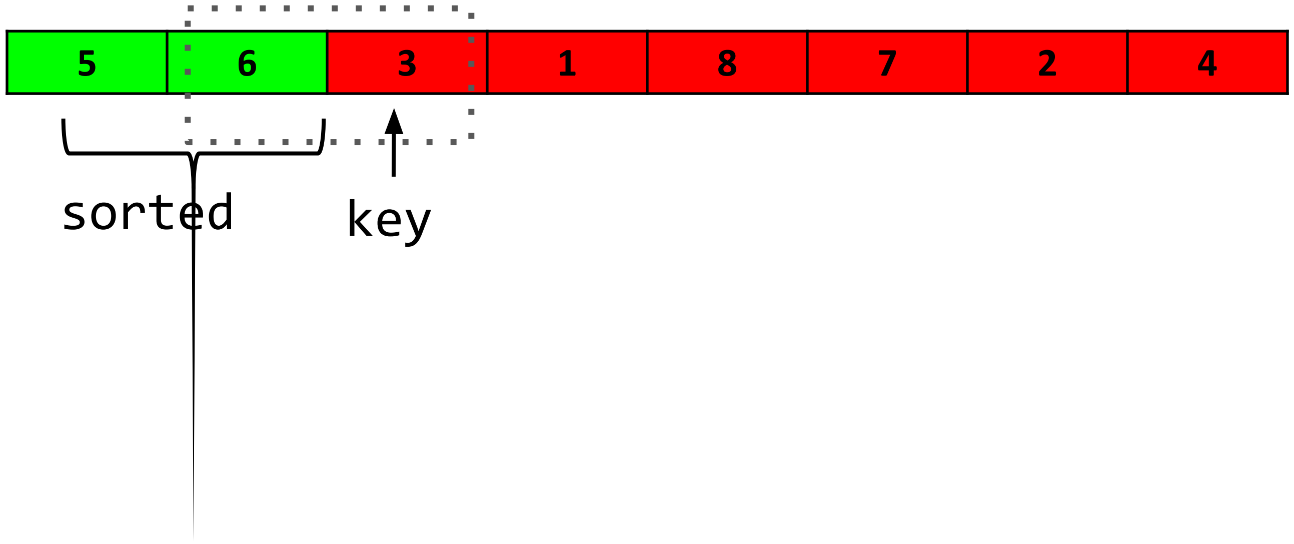


- Start from index=1 because a list of 1 item is trivially sorted.
- Is a[key] < a[key-1]
  - YES: swap(a[key], a[key-1])
  - NO: done with current element. a[key] is the largest seen so far.

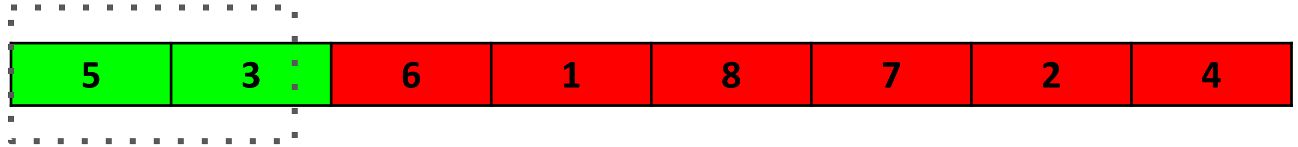
# Insertion sort - example



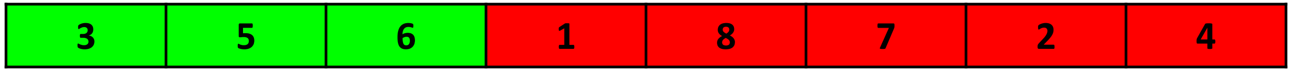
# Insertion sort - example



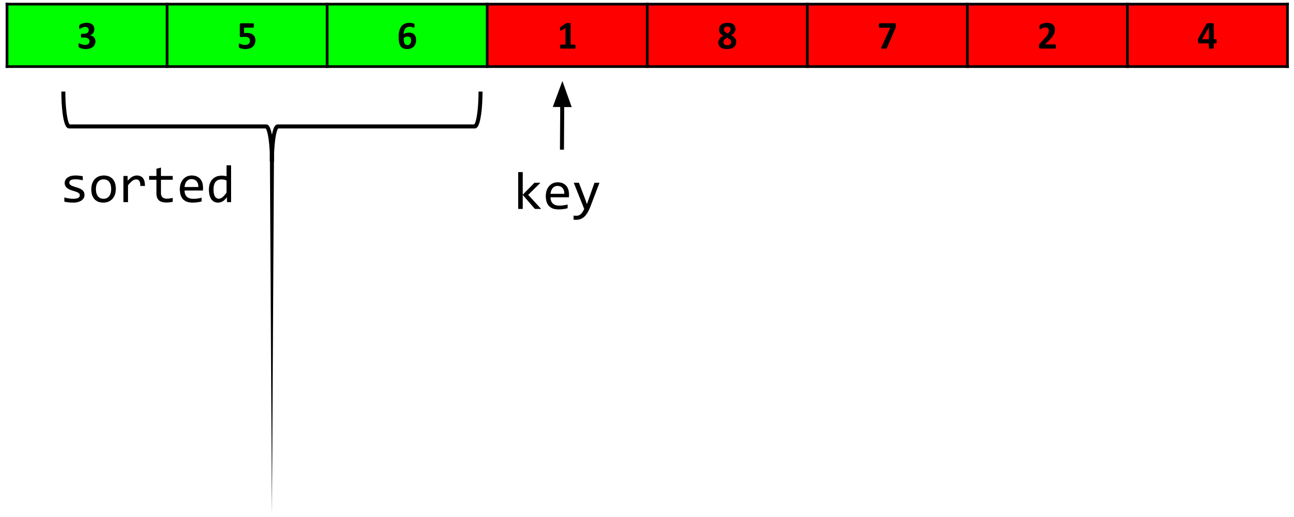
## Insertion sort - example



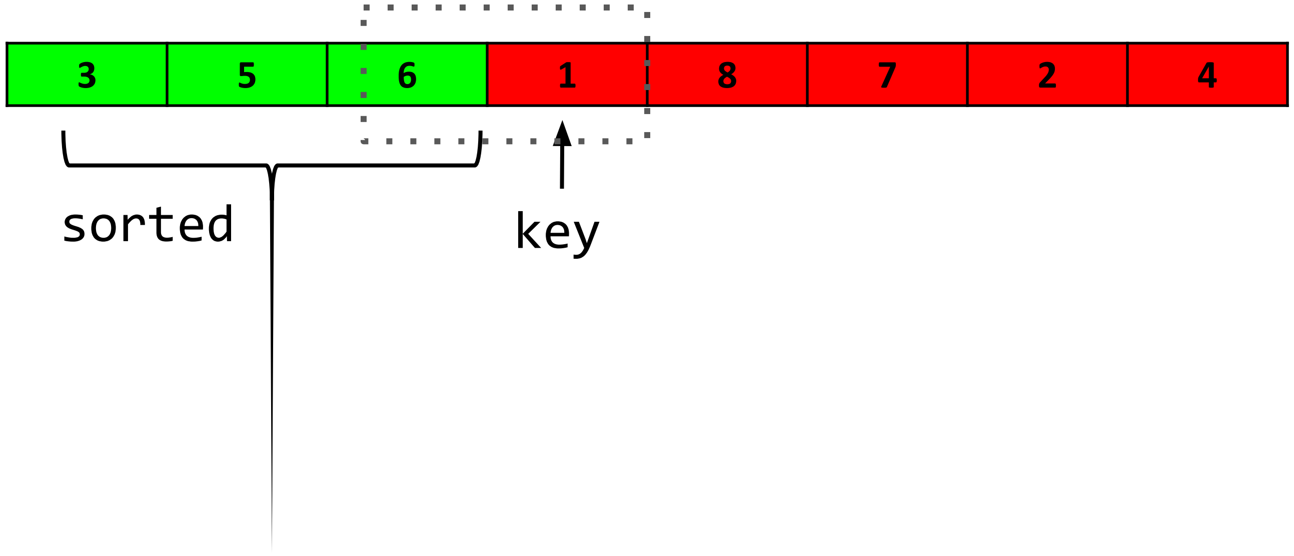
## Insertion sort - example



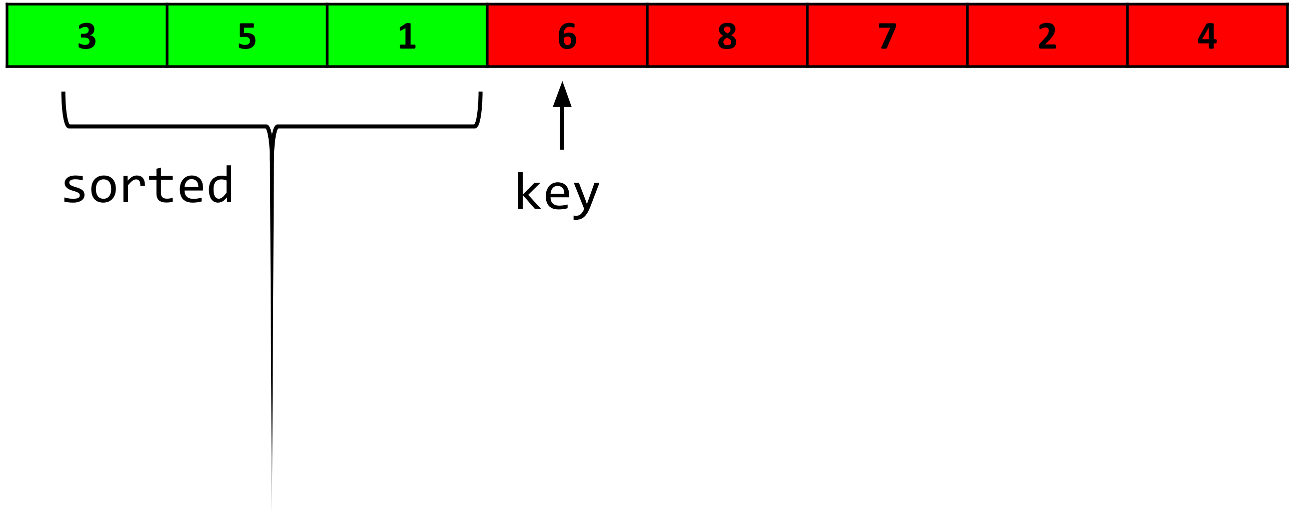
# Insertion sort - example



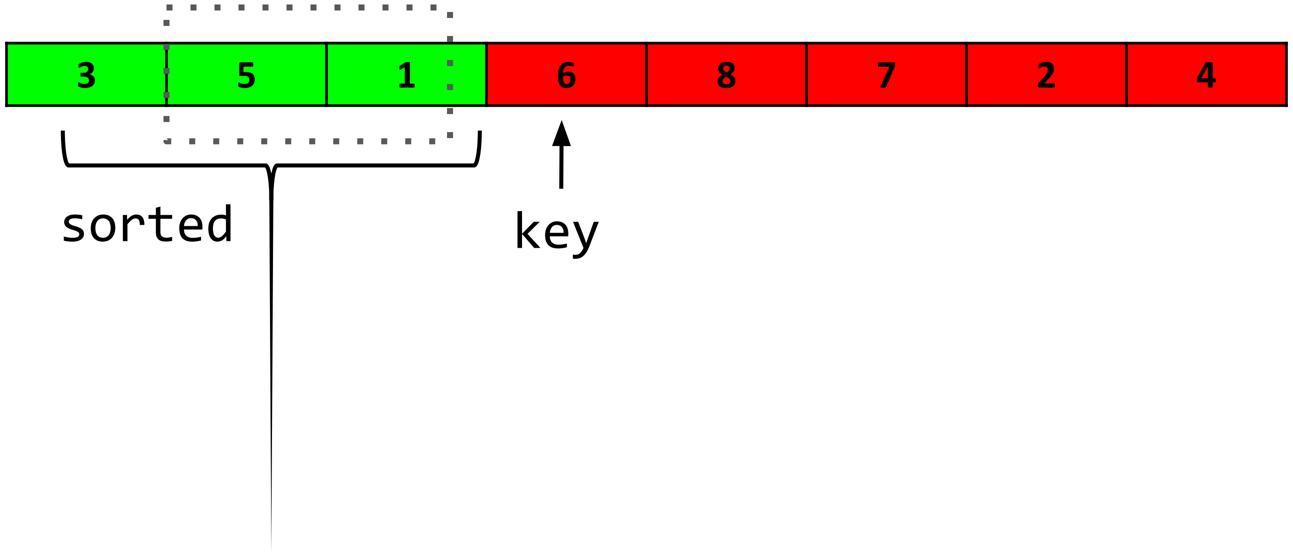
# Insertion sort - example



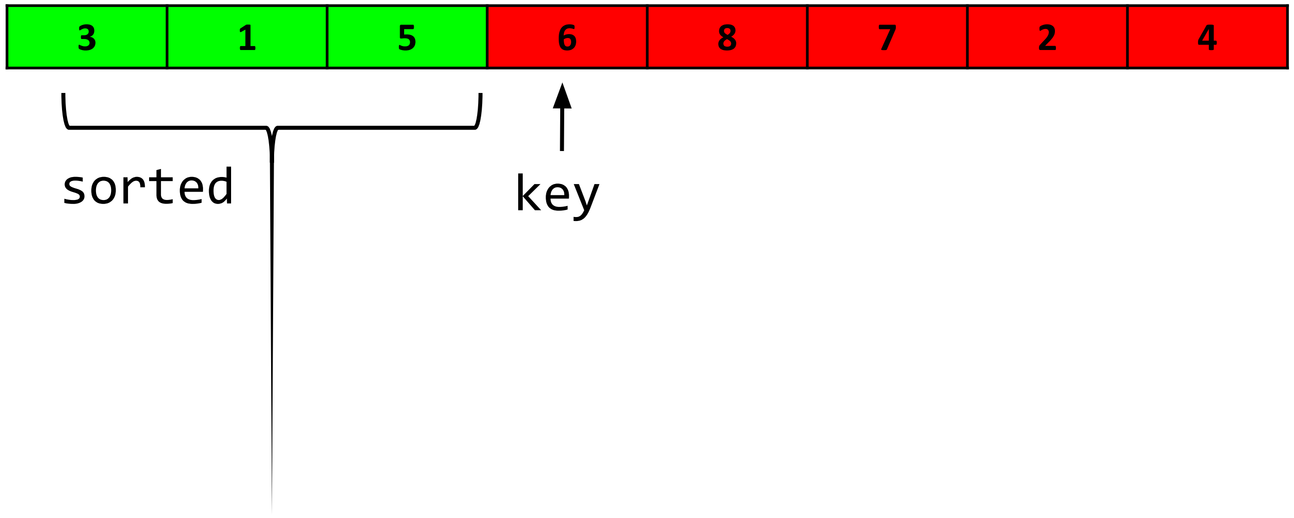
# Insertion sort - example



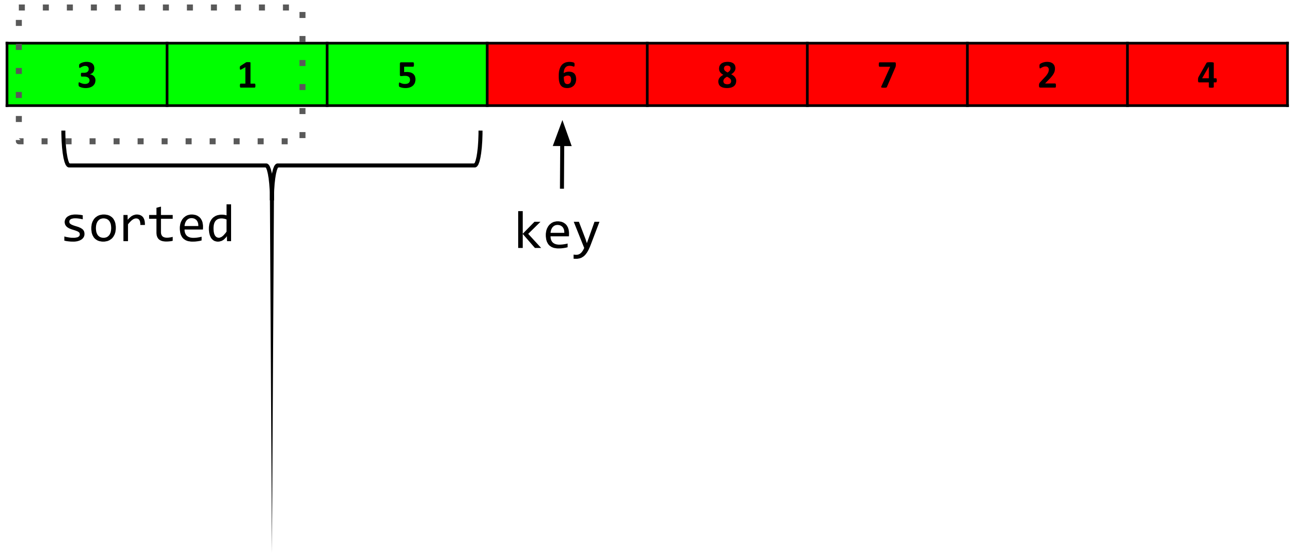
# Insertion sort - example



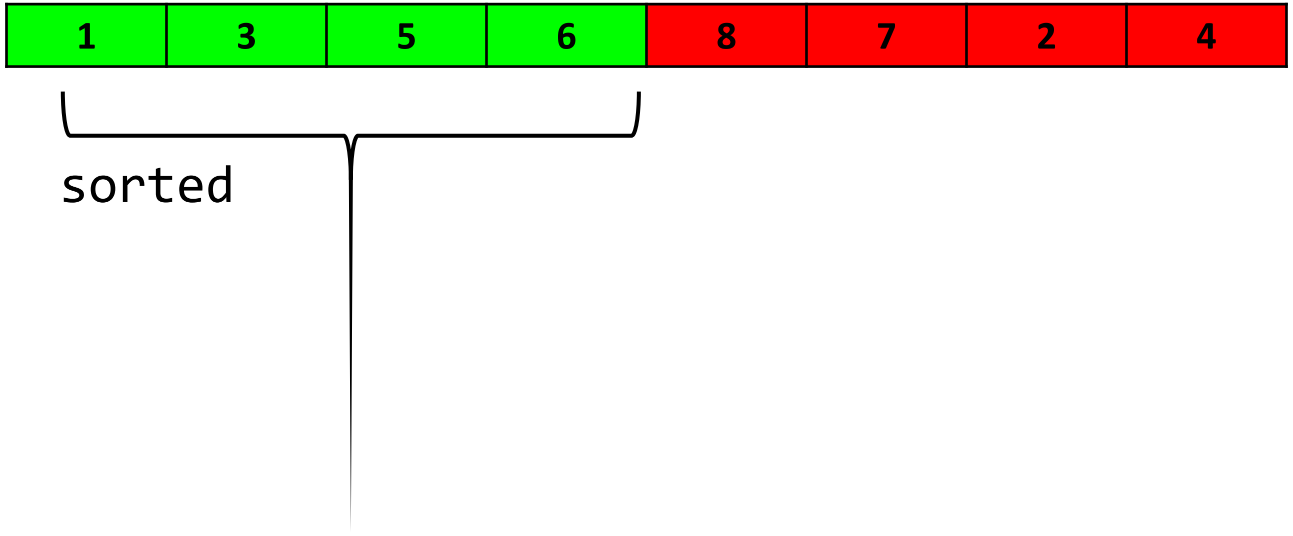
## Insertion sort - example



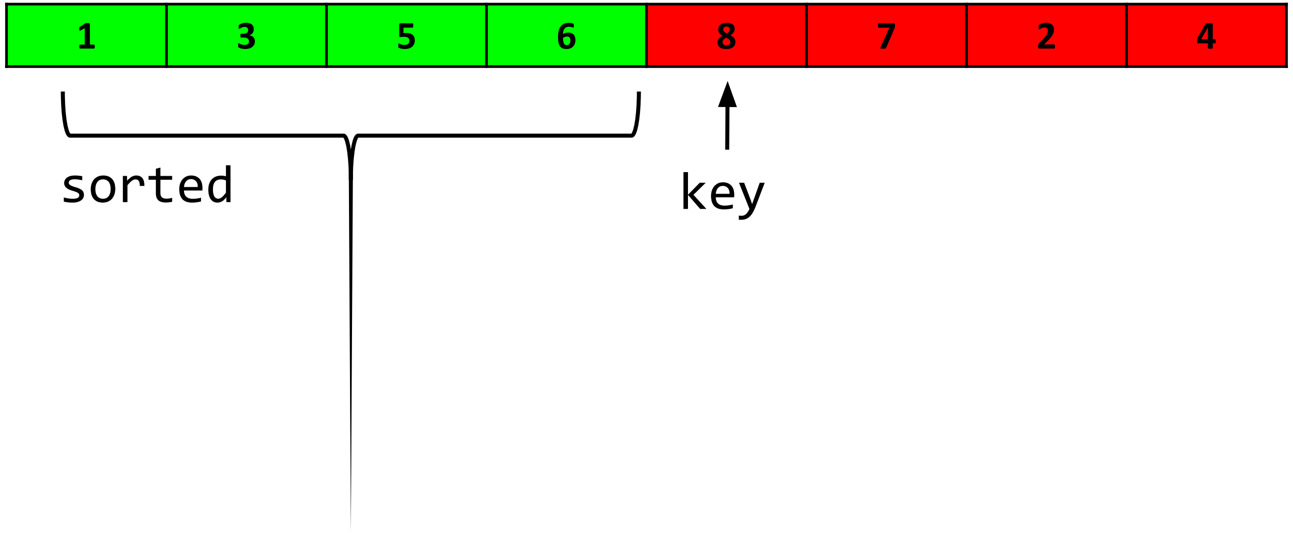
## Insertion sort - example



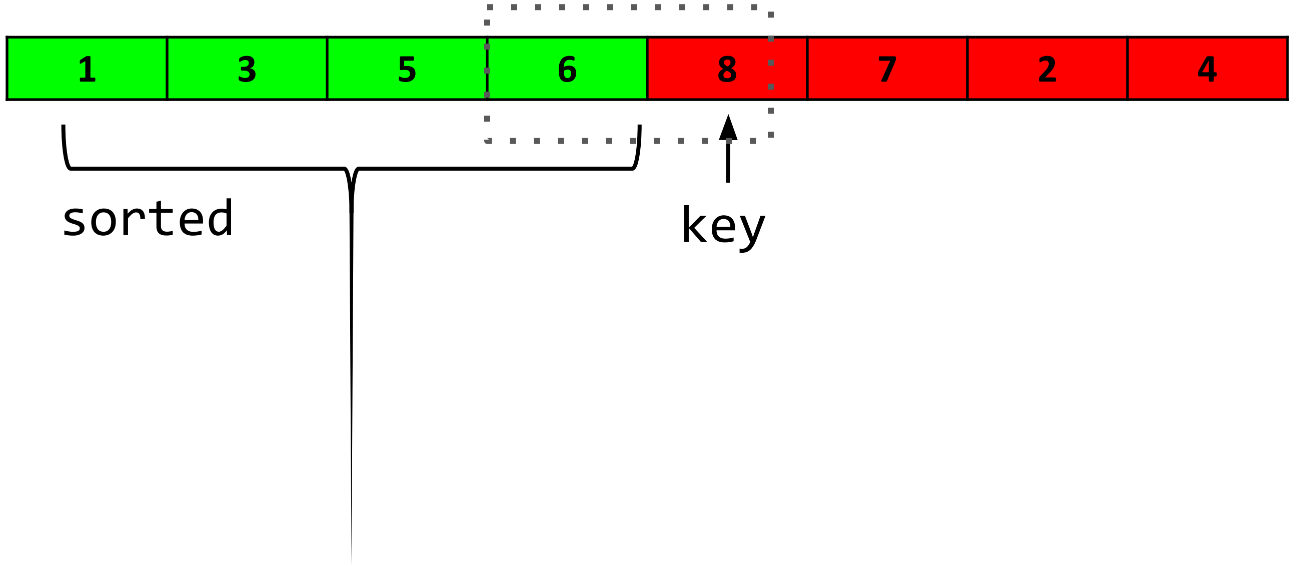
# Insertion sort - example



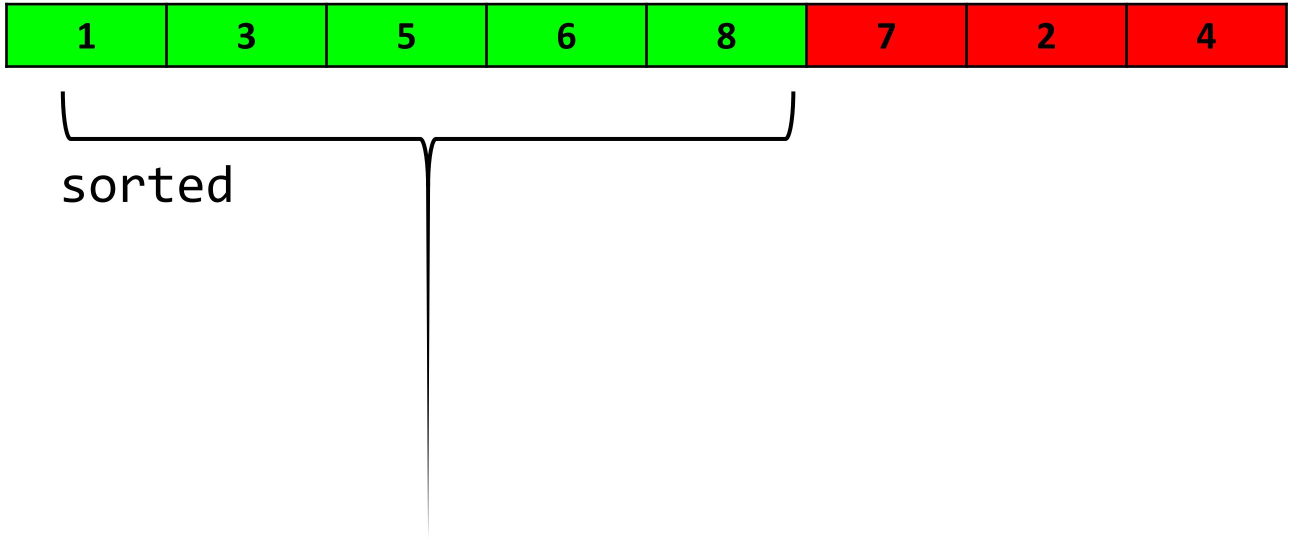
# Insertion sort - example



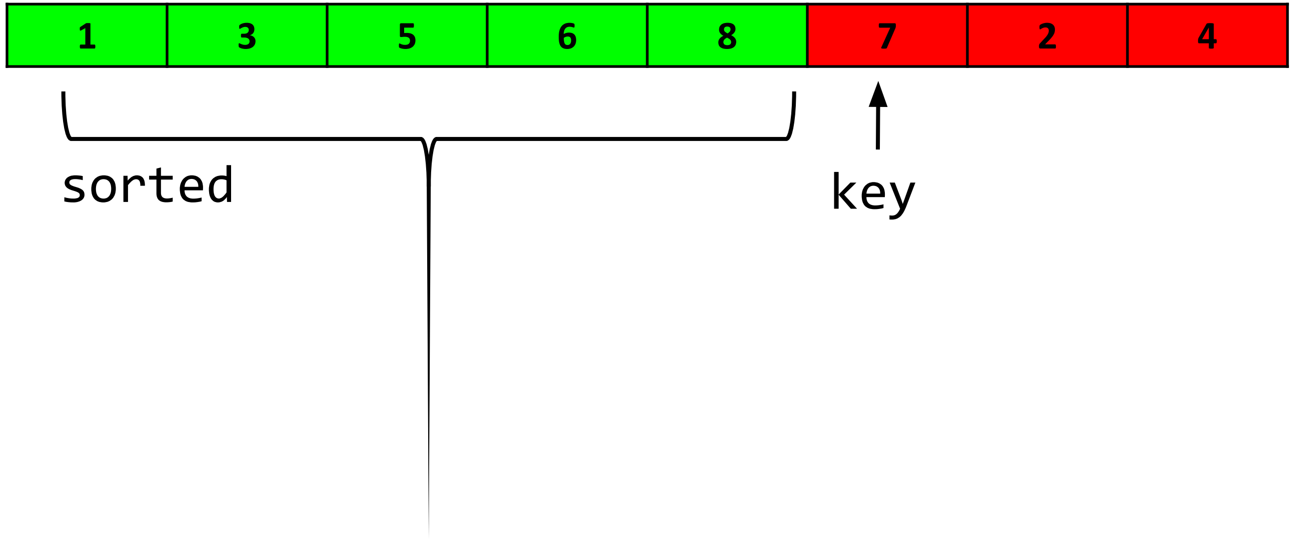
## Insertion sort - example



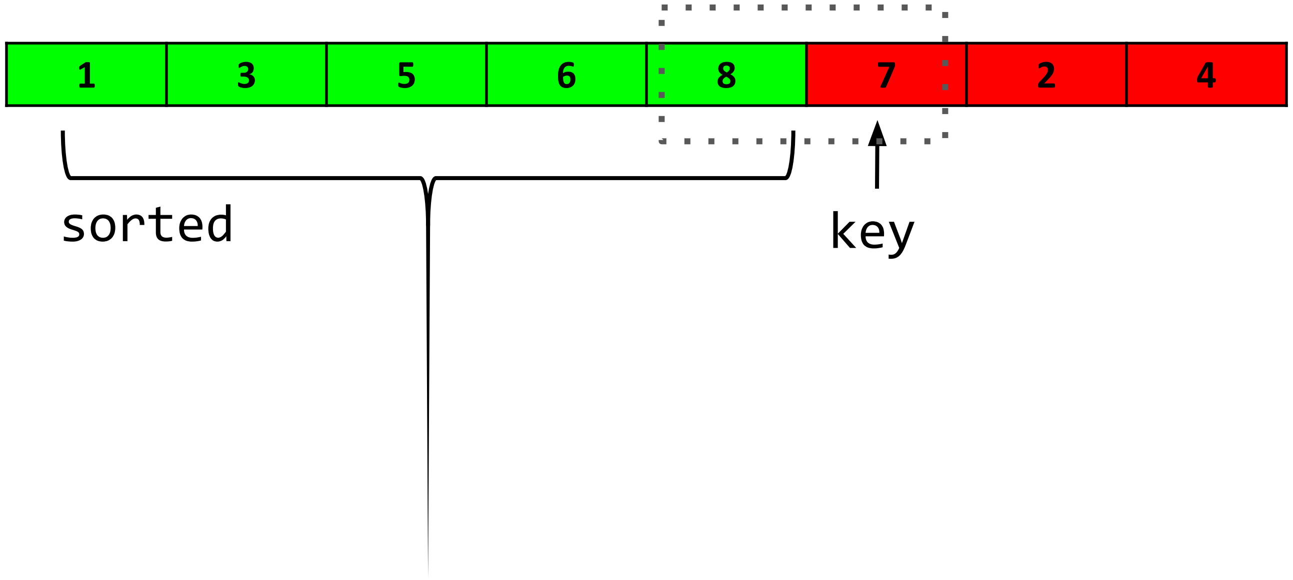
# Insertion sort - example



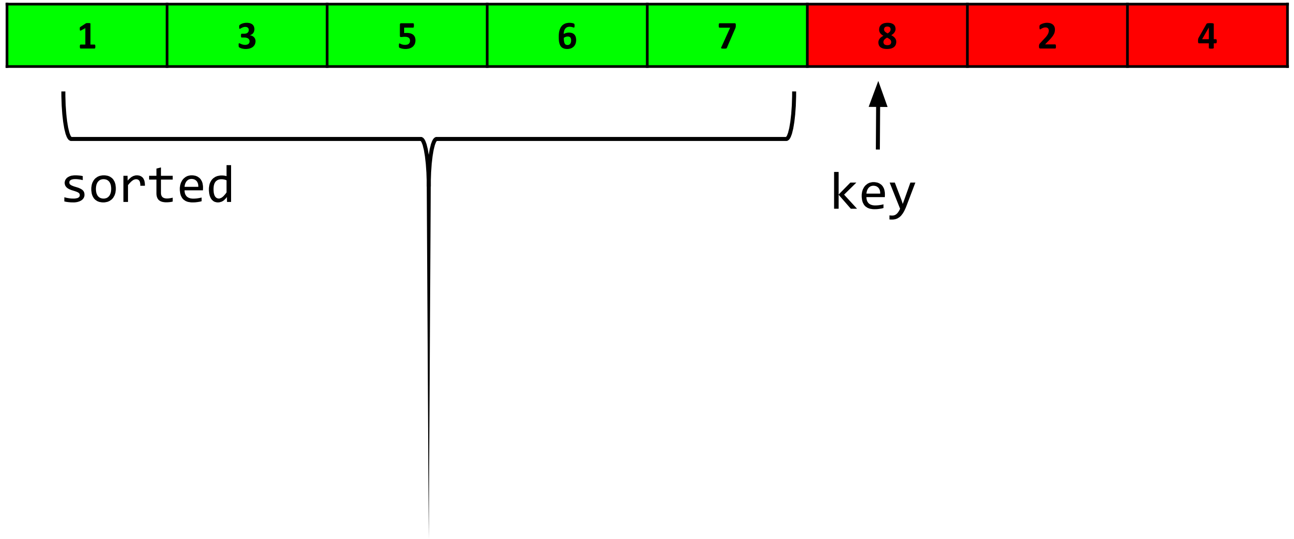
# Insertion sort - example



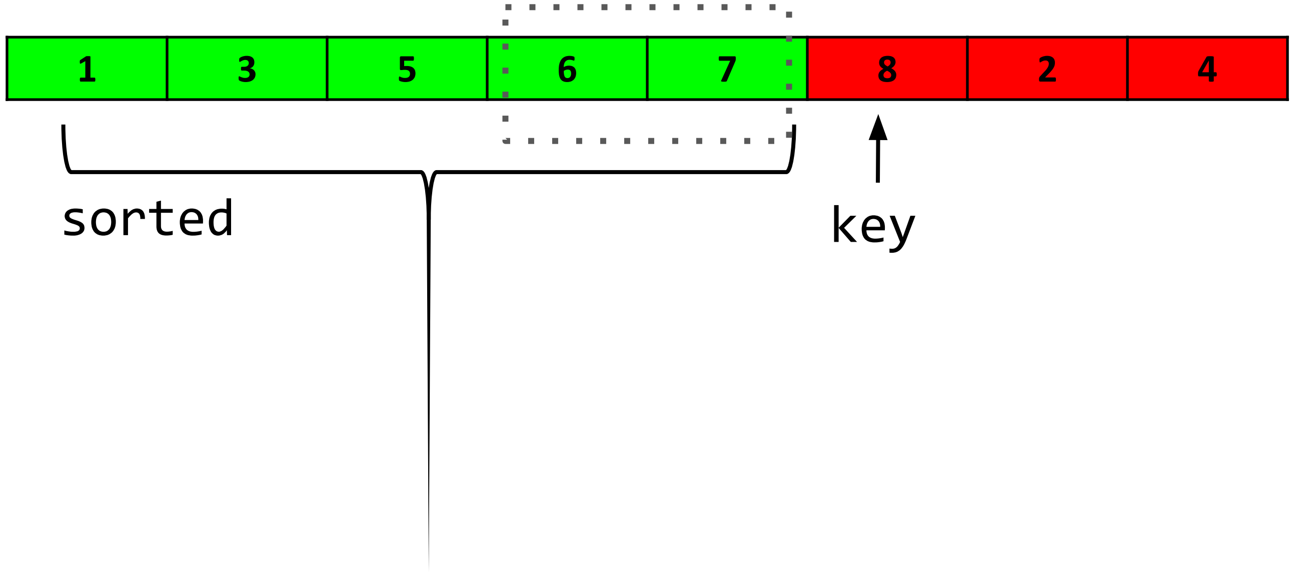
## Insertion sort - example



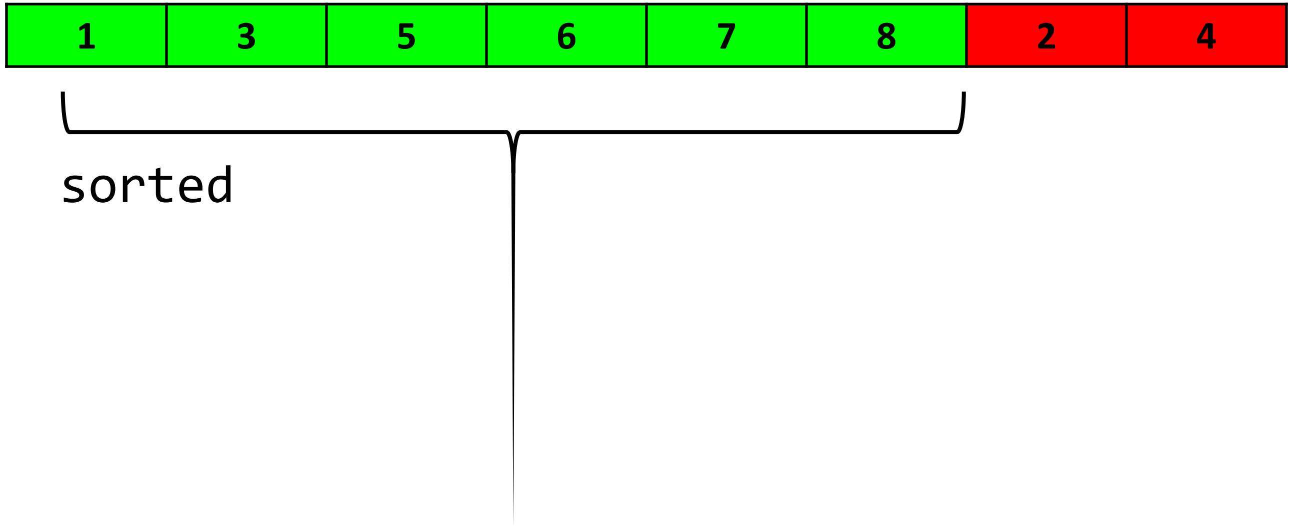
# Insertion sort - example



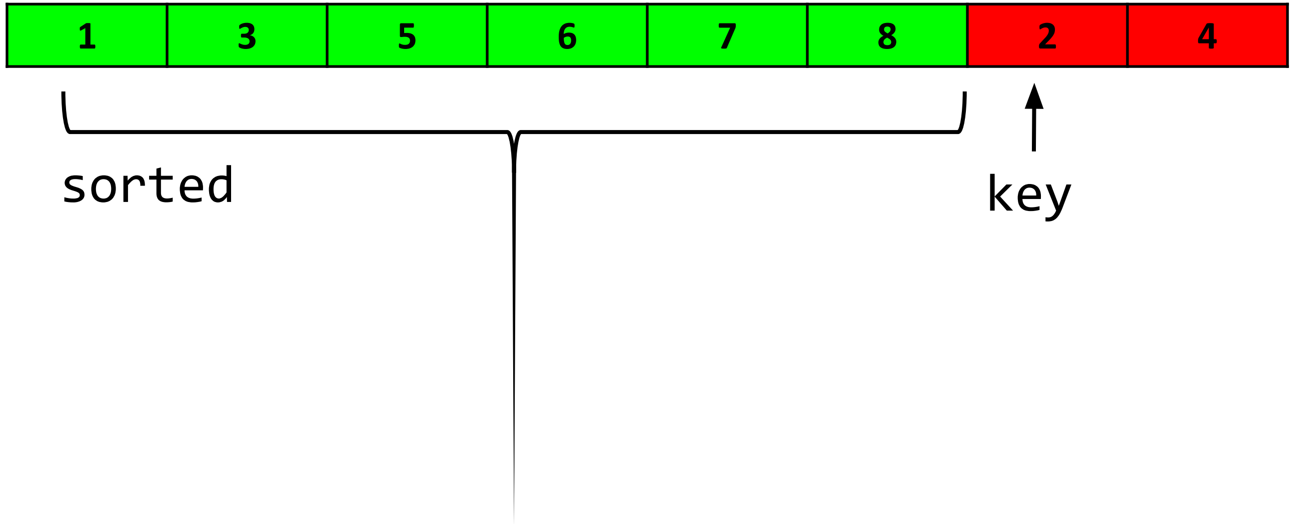
# Insertion sort - example



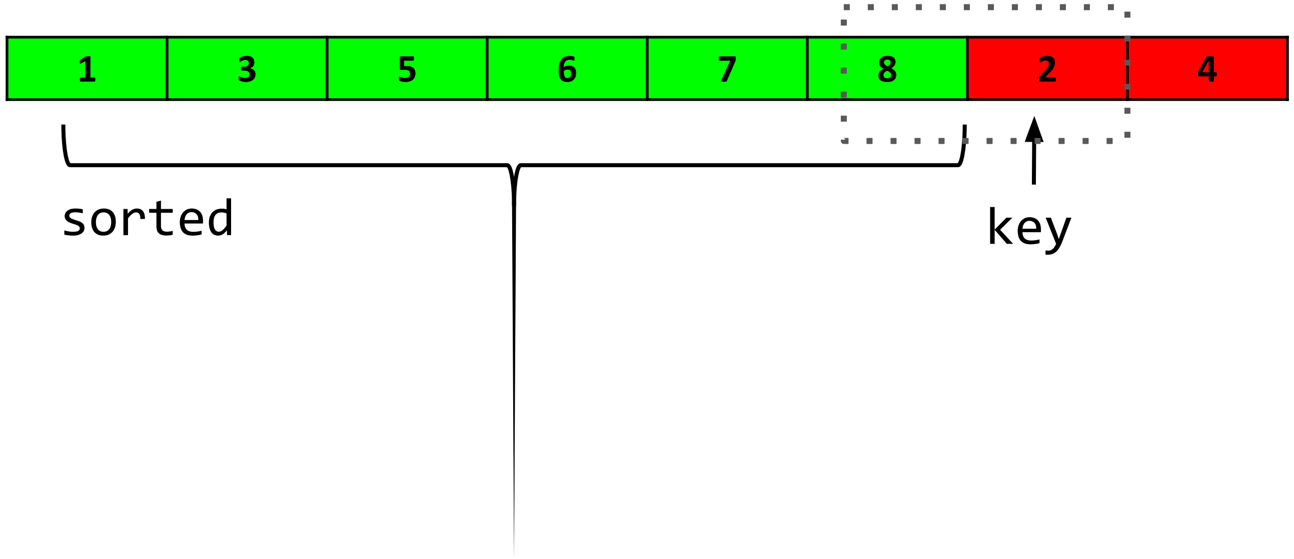
# Insertion sort - example



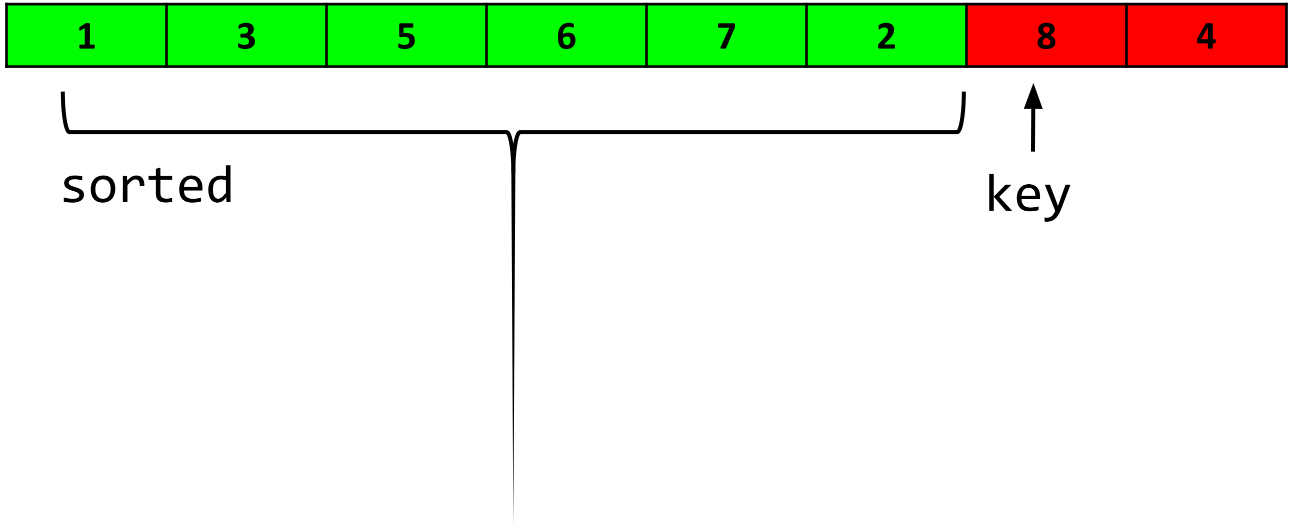
## Insertion sort - example



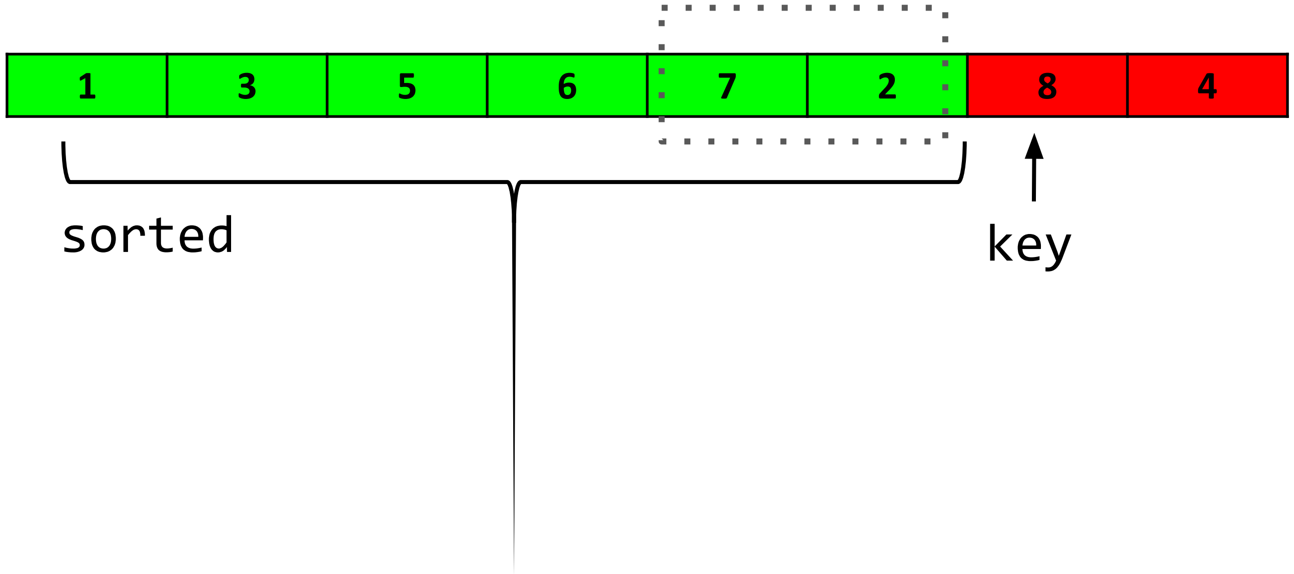
# Insertion sort - example



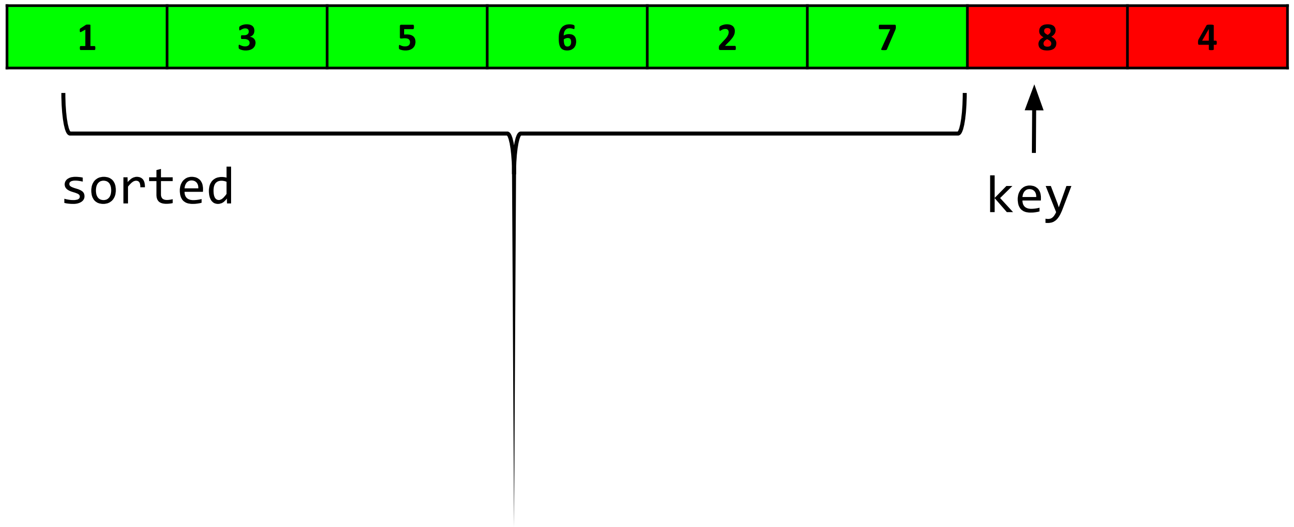
## Insertion sort - example



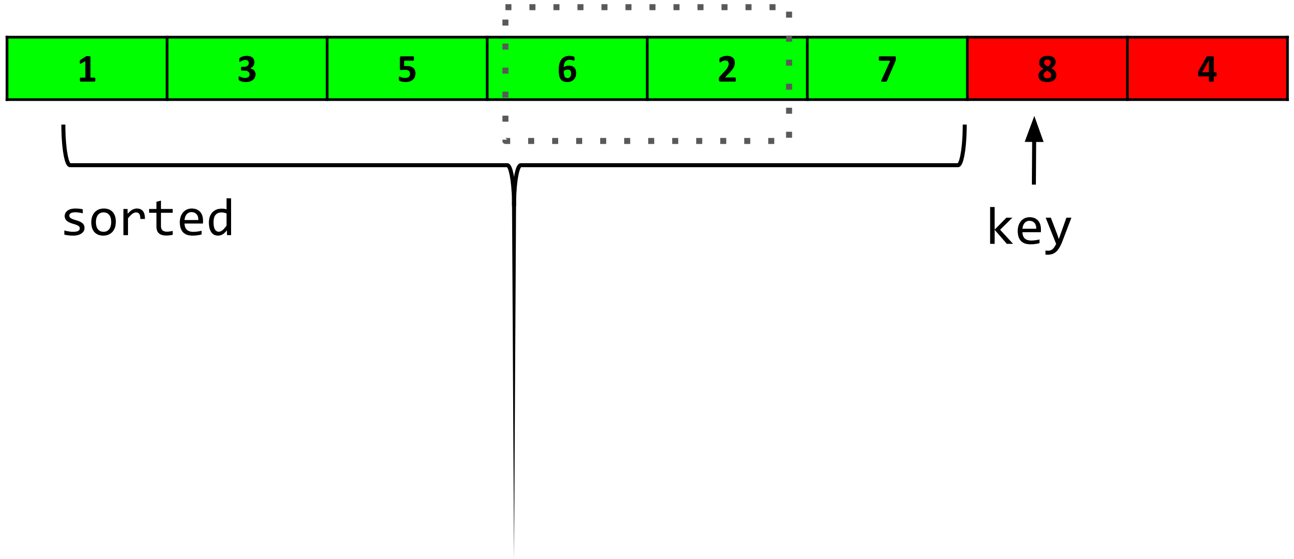
# Insertion sort - example



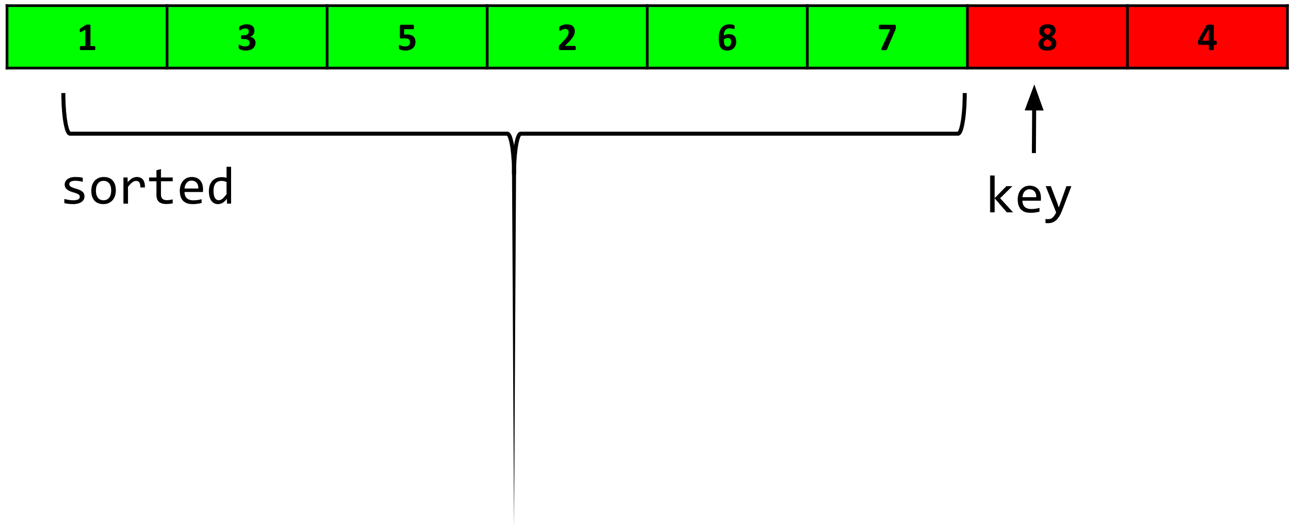
# Insertion sort - example



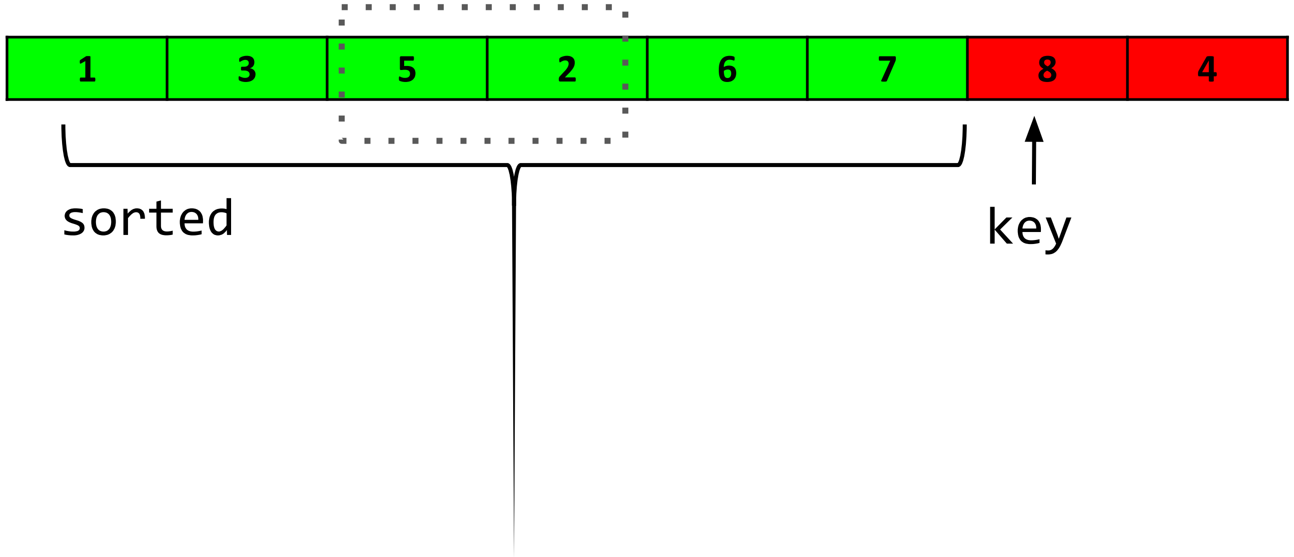
# Insertion sort - example



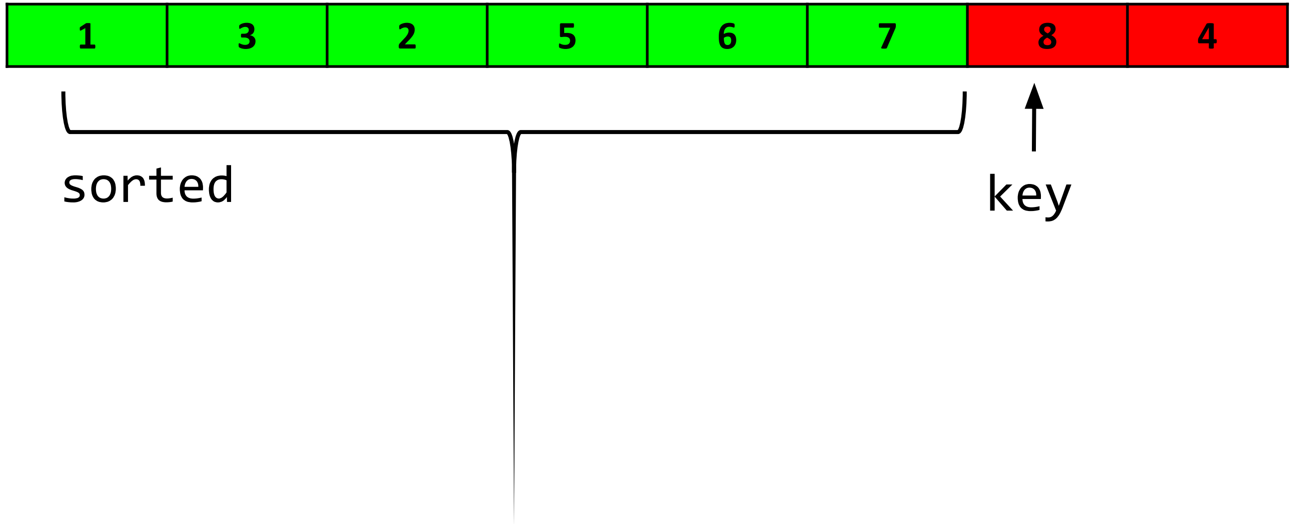
## Insertion sort - example



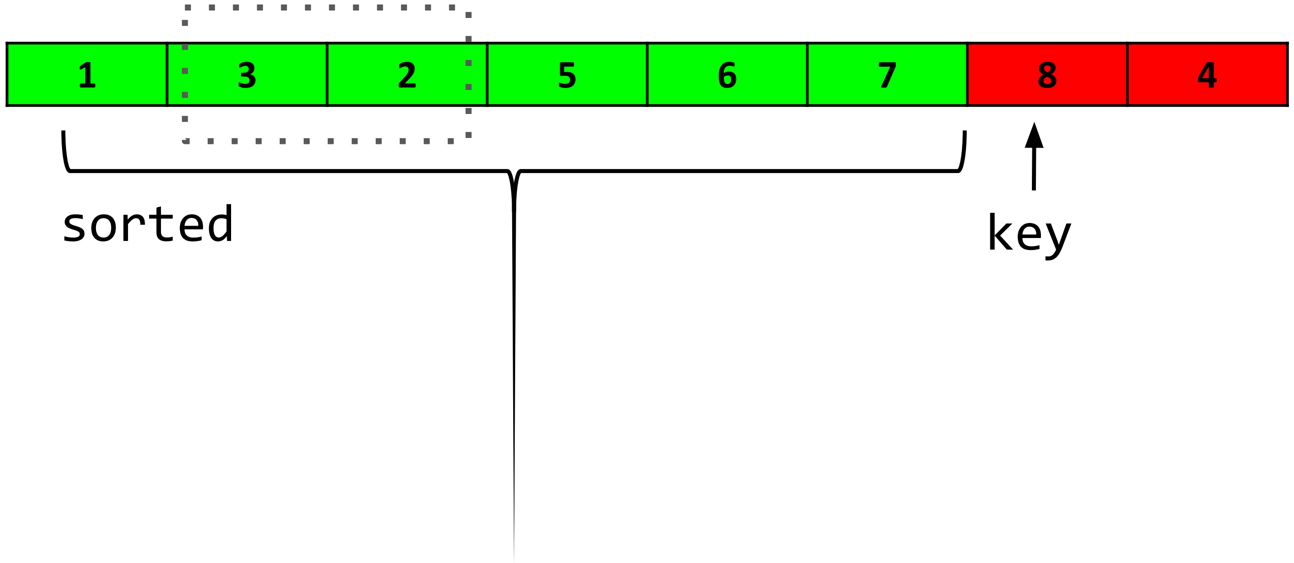
# Insertion sort - example



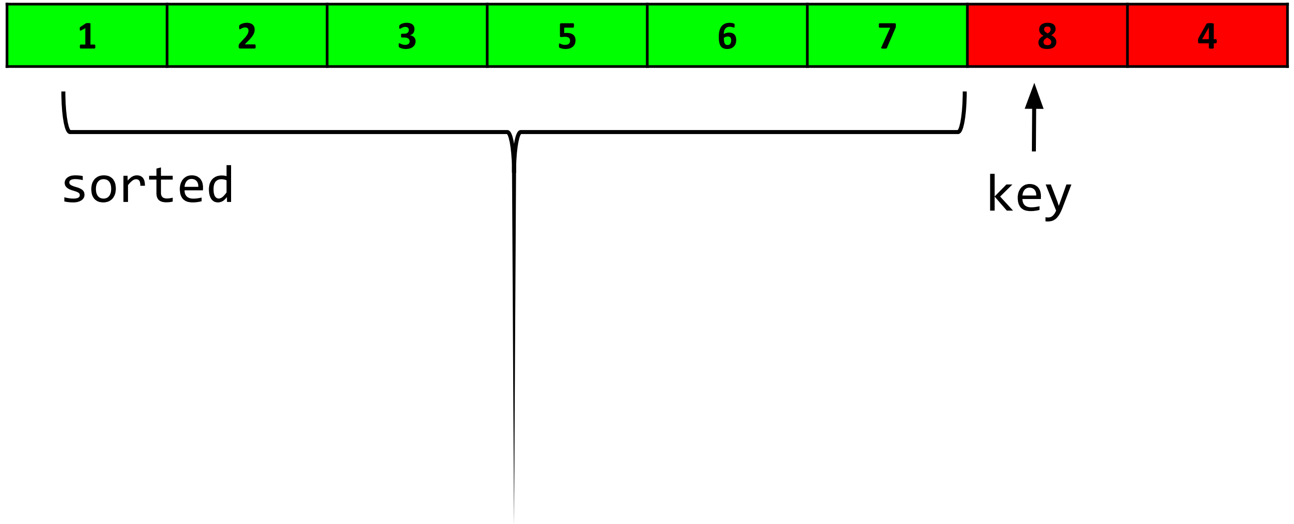
## Insertion sort - example



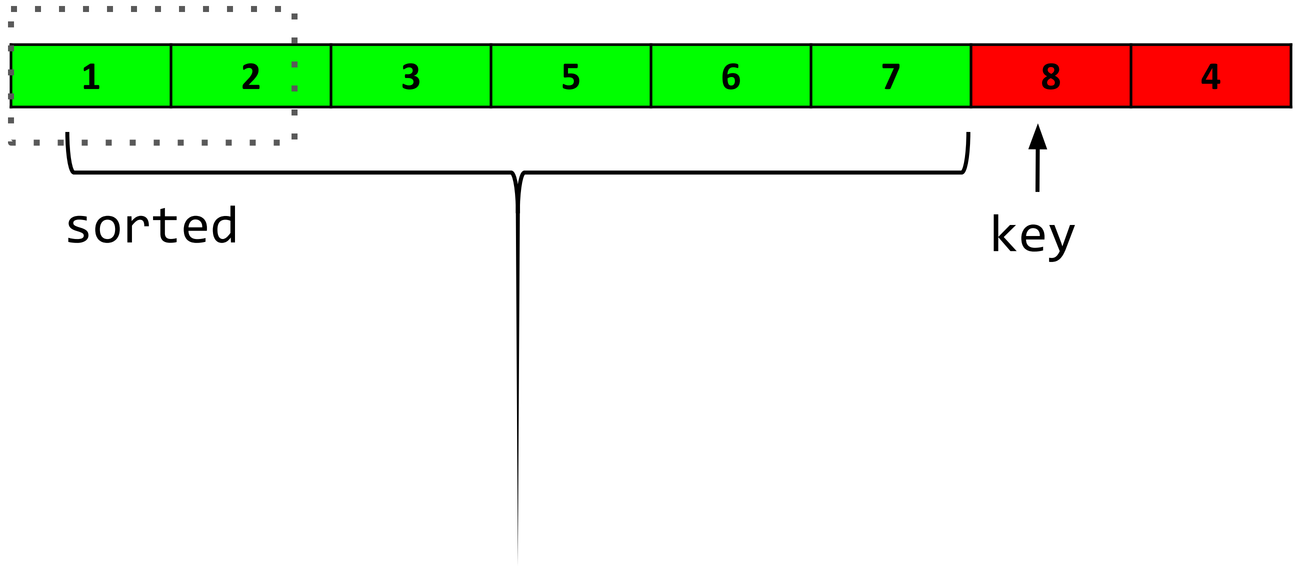
## Insertion sort - example



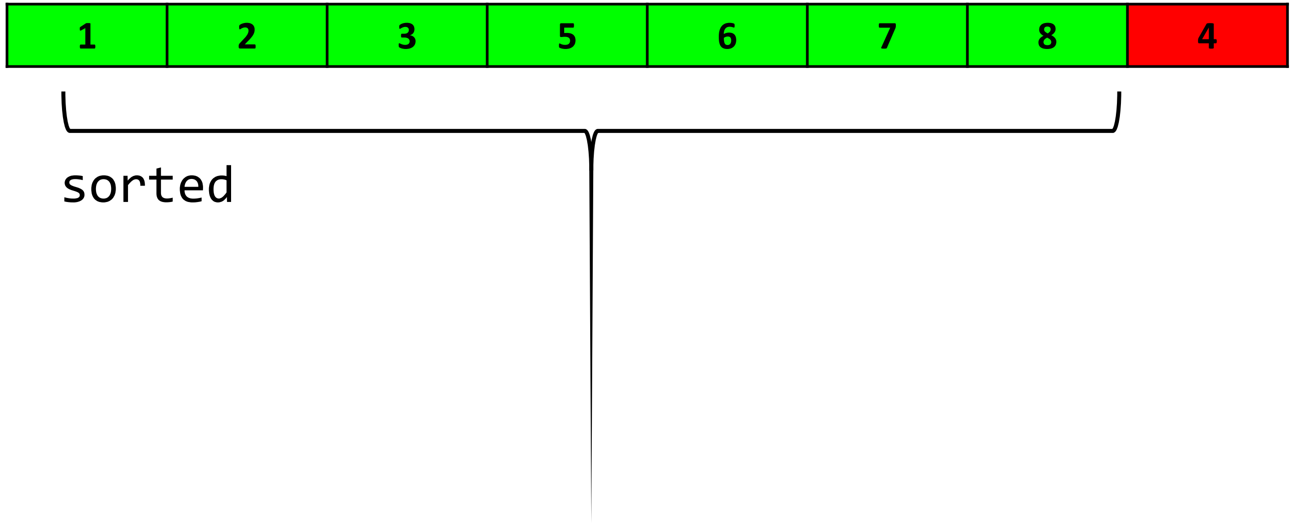
## Insertion sort - example



## Insertion sort - example



## Insertion sort - example

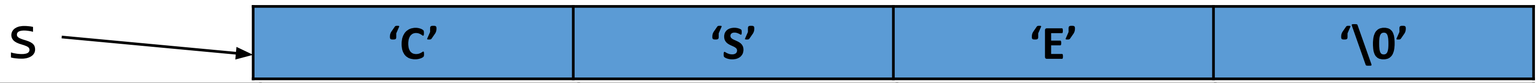


## Recap: Strings and Variables

# Strings

- Array of char
- Terminated by the null character `‘\0’` as per convention
- Example:

```
char s[]=“CSE”;
```



| Address | 0x7fffc510 | 0x7fffc511 | 0x7fffc512 | 0x7fffc513 |
|---|---|---|---|---|
| Value | 69 | 67 | 69 | 0 |

## Strings - Initializing

- `char s1[3];`
- `s1[0]=‘H’; //ASCII 72`
- `s1[1]=‘i’; //ASCII 105`
- `s1[2]=‘\0’; //ASCII 0`
- `char s2[]=“Hi”;`
- `char s3[]={‘H’,’i’,’\0’};`
- `char* s4=“Hi”;`
- `char s5[]= {72, 105,0};`

# String Literals

- `String Literals`
- Example:
  - `printf(“Hello World\n”);`
  - `char *s =“Hi”;`
- On data segment (initialized)
  - Cannot modify them

# Exercise – Identifying memory segments (strings)

```
void oat(char pie)
{
   char ham;
   char bun[4];
   char* ice = “pop”;
   static char egg = 1;
   static char nut;
}

char jam = 2;
char tea;
```

parameter

Local variable

Statically allocated array / local variable

String literal

*Address (still a local variable)*

*Static variable*

*Static variable*

*Global variable*

*Global variable*

- Print the length of a string using `strlen`

```
#include<string.h>
…
char s[]=“Hello”;
printf(“%d\n”,strlen(s));
```

- Use format specifier %s to print string values

```
printf(“%s\n”,s);
```