

**ECE 264 Spring 2023**  
***Advanced* C Programming**

# Midterm 1

- When: Thursday (16th Feb)
- How: Online via Brightspace for 24 hours from 7:30 am (16th Feb) to 7:29 am (17th Feb)
- Time : 3 hours (Expected to be done in 1 hour).
- Questions similar to quiz but expect some code to be understood or written.

# Topics for Midterm 1

- Compilation and Makefile
- Pointers
- Files
- Heap

# Makefile

- How it is organized?
- Variables in Makefile
- Can we have cyclic dependencies in Makefile?

# Pointers

- How addresses are calculated for array accesses?
  - `arr[i]`?
- Size of pointers to different types?
- Can two pointers point to same object?

# Files

- Function to read/write to file:
  - fread/write/fprintf/fscanf/fgets
- Opening mode: “r” v/s “w”?
- What does fgets return?

# Heap

- malloc/free
- Invalid accesses:
  - Use after free.
  - Double free.

# **Homework 07 qsort and Function Pointer**



# Function Address (HW01 as example)

```
bash-4.2$ gdb main
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-116
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://www.gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/shay/a/luece264/ECE264/assignment1/program.elf:
(gdb) b addop
Breakpoint 1 at 0x400e49: file add.c, line 16.
(gdb) b mulop
Breakpoint 2 at 0x400f15: file mul.c, line 9.
(gdb) b subop
Breakpoint 3 at 0x401236: file sub.c, line 9.
(gdb) b divop
Breakpoint 4 at 0x400e8b: file div.c, line 9.
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
int func1(int a, int b)
{
    return (a + b);
}
```

```
int func2(int a, int b)
{
    return (a * b);
}
```

```
typedef int (*functype)(int a, int b);
```

```
int main(int argc, char * * argv)
{
    functype ptr;
    ptr = func1;
    int c = ptr(3, 5);
    printf("c = %d\n", c);
    ptr = func2;
    int d = ptr(3, 5);
    printf("d = %d\n", d);
    return EXIT_SUCCESS;
}
```

Program's Output

```
c = 8
d = 15
```

name of the data type

return type **(\*name)**(arguments);

**a function pointer**

**a pointer for function  
(two int argument)  
return int**

# qsort in C

```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmem, size_t size,  
           int (*compar)(const void *, const void *));
```

The **qsort()** function sorts an array with *nmem* elements of size *size*. The *base* argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by *compar*, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

```
#include <stdlib.h>
```



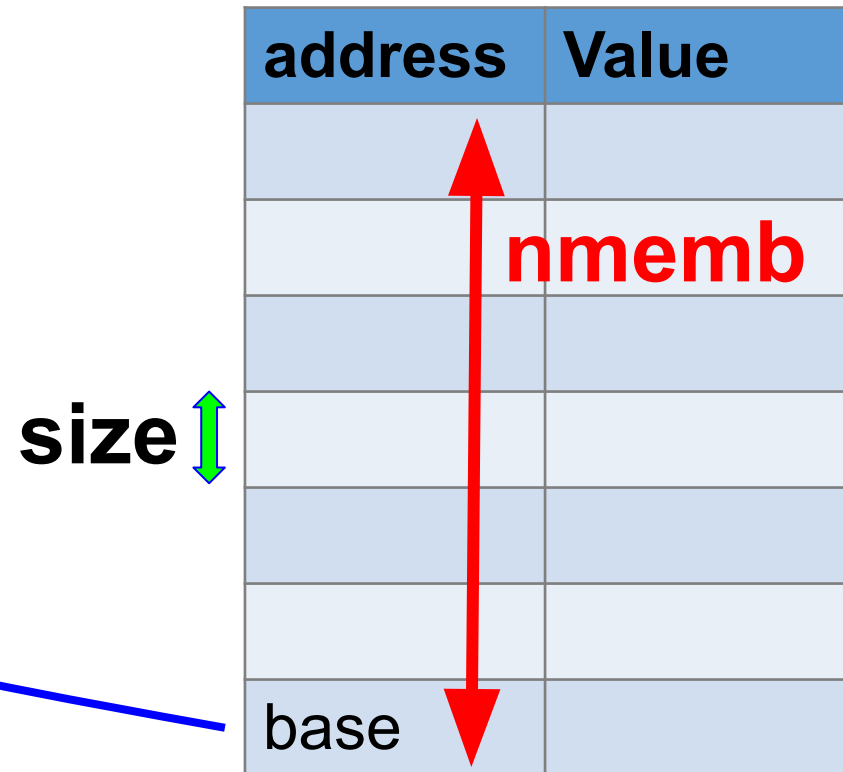
```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *));
```

void \*: it is a pointer but the type is not specified now

```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmemb, size_t size,  
          int (*compar)(const void *, const void *));
```

void \*: it is a pointer but the type is not specified now



```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *));
```

qsort: selects elements by the addresses based on  
& arr[k] = & arr[0] + k · size of an element  
sends the addresses to compar to determine  
the order

& arr[0] = base  
 $0 \leq k < nmemb$

size ↑

address	Value
base	

```
int (*compar)(const void *, const void *, void *)
```

- A function
- The function returns int
- The function takes two arguments, both are addresses
- The function must not use the left hand side rule
- Document: *The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined*
- The function should compare values, not addresses

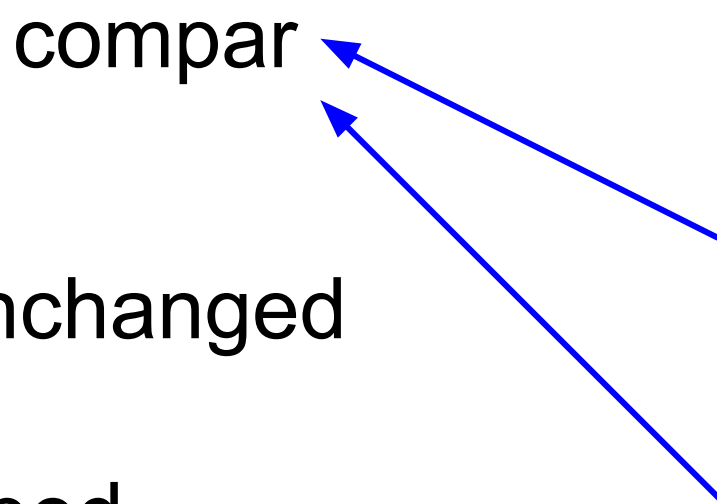
```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *));
```

compar

1.  $compar < 0$ : V1 and V2 unchanged
2.  $compar = 0$ : undecided
3.  $compar > 0$ : V1, V2 swapped

address	Value
A2	V2
A1	V1
base	





```
int compareint(const void * arg1, const void * arg2)
// to sort array of int
{
    const int * ptr1 = (const int *) arg1;
    const int * ptr2 = (const int *) arg2;
    int val1 = * ptr1;
    int val2 = * ptr2;
    if (val1 < val2) { return -1; }
    if (val1 == val2) { return 0; }
    return 1;
}
```

2 [

] 1

3 ]

1. cast void \* to the right type \*
2. get values at the two addresses
3. compare and return <0, 0, or > 0

```
int comparevector(const void *arg1, const void *arg2)
// sort vectors by x
{
    // ptr1 and ptr2 are Vector *
    const Vector * ptr1 = (const Vector *) arg1;
    const Vector * ptr2 = (const Vector *) arg2;
    int val1 = ptr1 -> x;
    int val2 = ptr2 -> x;
    if (val1 < val2) { return -1; }
    if (val1 == val2) { return 0; }
    return 1;
}
```

2

1

3

1. cast void \* to the right type \*
2. get values at the two addresses
3. compare and return <0, 0, or > 0

```
int cmpstring(const void *arg1, const void *arg2)
// sort array of strings
{
    // arg1 and arg2 are string *
    // string is char *, thus ptr1 and ptr2 are char * *
    const char * const * ptr1 = (const char * *) arg1;
    const char * const * ptr2 = (const char * *) arg2;
    2 [ const char * str1 = * ptr1; // type: string
        const char * str2 = * ptr2;
        return strcmp(str1, str2); ] 3
}
```

1

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    char str1[20];
    char str2[20];
    strcpy(str1, "First");
    strcpy(str2, "Second");
```

```
const char * chptr1 = & str1[0]; // cannot use left hand side rule
```

```
char * const chptr2 = & str1[0]; // cannot change chptr2's value
```

```
const char * const chptr3 = & str1[0]; // neither
```

```
// * chptr1 = 'C';      // not allowed
* chptr2 = 'C';      // OK
chptr1 = & str2[0];  // OK
// chptr2 = & str2[0]; // not allowed
// chptr3 = & str2[0]; // not allowed
// * chptr3 = 'C';    // not allowed
return EXIT_SUCCESS;
}
```

```
int cmpstring(const void *arg1, const void *arg2)
// sort array of strings
{
    // arg1 and arg2 are string *
    // string is char *, thus ptr1 and ptr2 are char * *
    const char * const * ptr1 = (const char * *) arg1;
    const char * const * ptr2 = (const char * *) arg2;
    const char * str1 = * ptr1; // type: string
    const char * str2 = * ptr2;
    return strcmp(str1, str2);
}
```

```
int cmpstring(const void *arg1, const void *arg2)
// sort array of strings
{
    // arg1 and arg2 are string *
    // string is char *, thus ptr1 and ptr2 are char * *
    const char * const * ptr1 = (const char * *) arg1;
    const char * const * ptr2 = (const char * *) arg2;
    2 [ const char * str1 = * ptr1; // type: string
        const char * str2 = * ptr2;
        return strcmp(str1, str2);
    ] 3
}
```

1. cast void \* to the right type \*
2. get values at the two addresses
3. compare and return <0, 0, or > 0

# Quick Sort

- Quick Sort uses transitivity to avoid unnecessary comparisons
- transitivity: if  $a > b$  and  $b > c$ , then  $a > c$ . No need to compare  $a$  and  $c$ .
- Selection sort and bubble sort do not use transitivity. Thus, they are not as fast as quick sort.
- The algorithm of quick sort will be explained later.

# Recursion 01

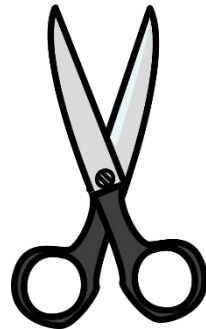
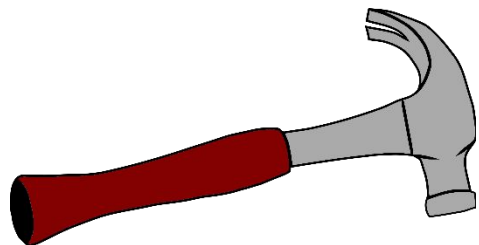


# Common Misunderstanding

- Wrong: Slow, Useless, Difficult to understand, Use too much memory, For exams only
- Truth: Recursion is slow, useless, difficult to understand, inefficient, ....., really bad, *if you do not understand it.*
- Many books do not explain recursion well.
- If you understand recursion and use it properly, it can be fast and efficient.

# Where Recursion is Used?

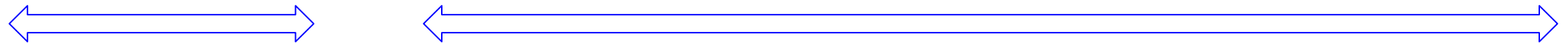
- Sorting (almost everywhere, such as quick sort)
- Strategy games (chess, go)
- Optimization
- ...
- Recursion is one of many tools. Recursion is good for solving some problems.



# Recursion is natural

- You have parents. Your parents have their parents. They have their parents. This is recursion.
- You are younger than your parents. Your parents are younger than your grandparents. Different generations are different.
- If you have no child, the recursion stops at you. If you have a child (or several children) and no grandchild, recursion stops at your child (or children).
- Three essential components of recursion: recurring patterns, changes, and stop condition.

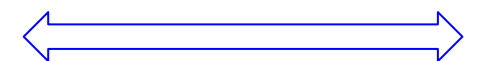
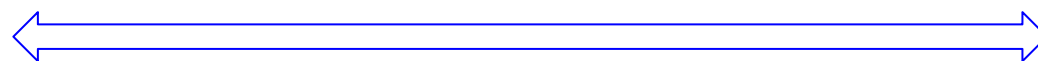
# Recursion in Quick Sort



**< R1**

**> R1**

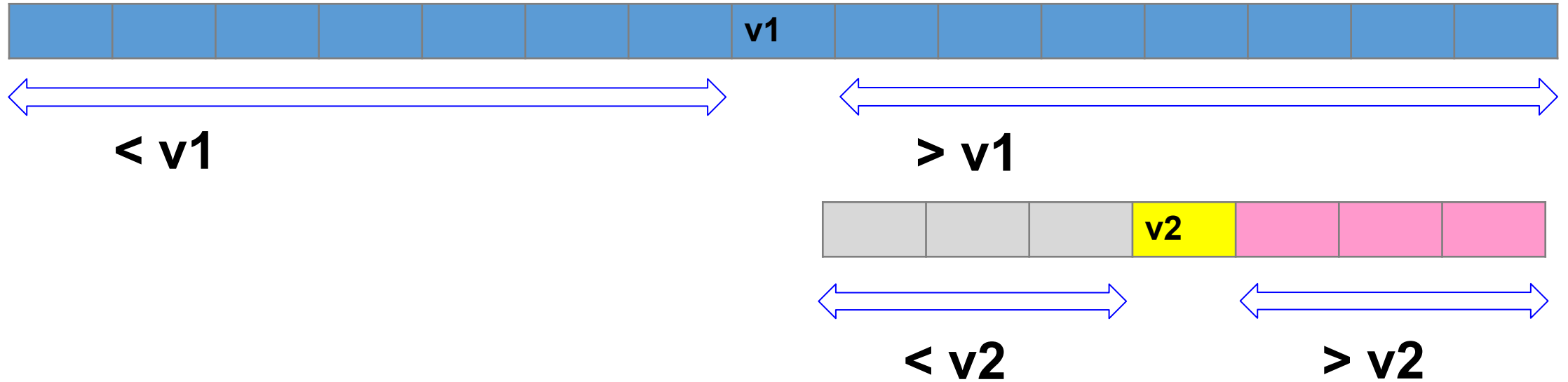
**transitivity: if  $a > b$  and  $b > c$ , then  $a > c$**



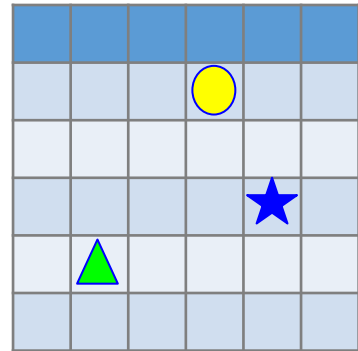
**< R2**

**> R2**

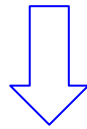
# Recursion in Binary Search (sorted data)



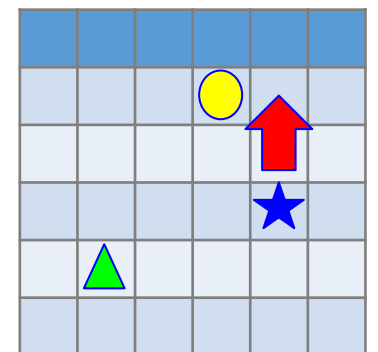
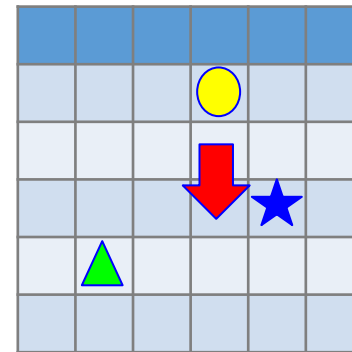
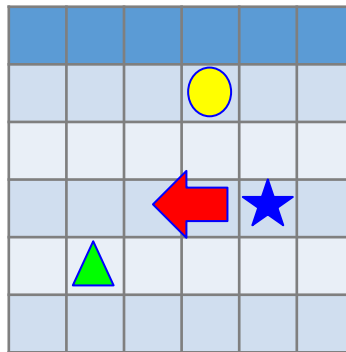
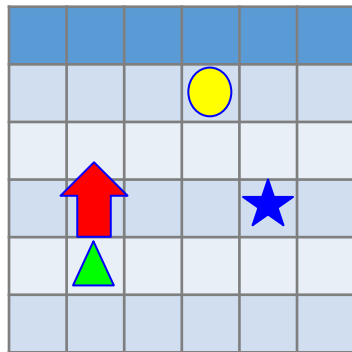
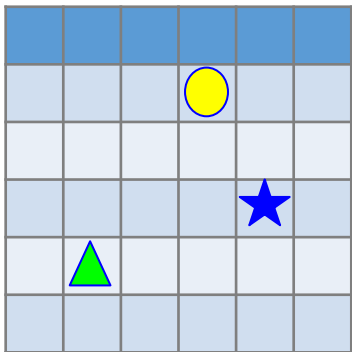
# Recursion in Board Games



current state



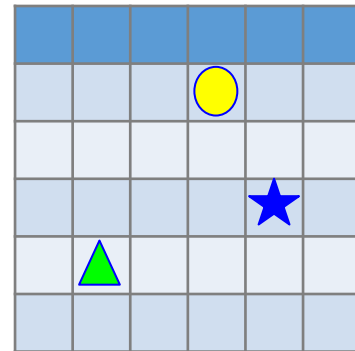
different states after one move



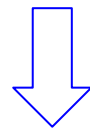
# 3 Essential Components in Recursion

- Stop condition (or conditions), also call terminating conditions: when nothing needs to be done.
- Changes.
- Recurring pattern.

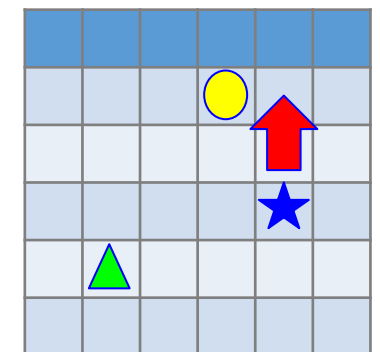
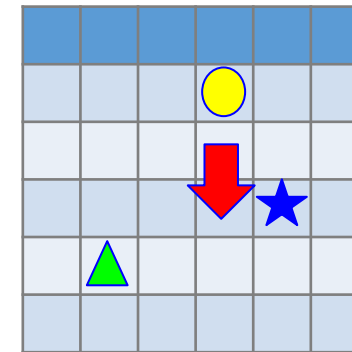
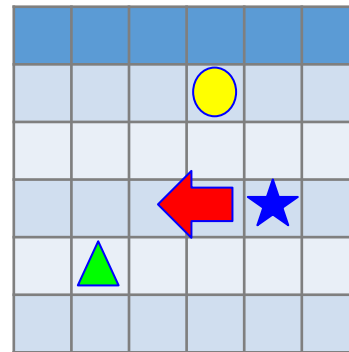
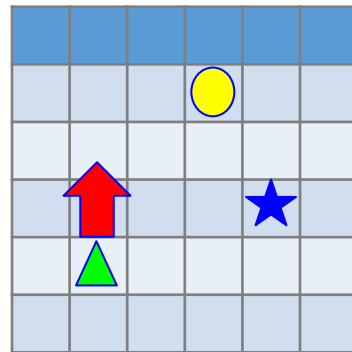
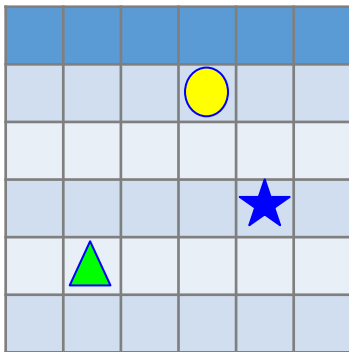
# Recursion good for “branches”



current state

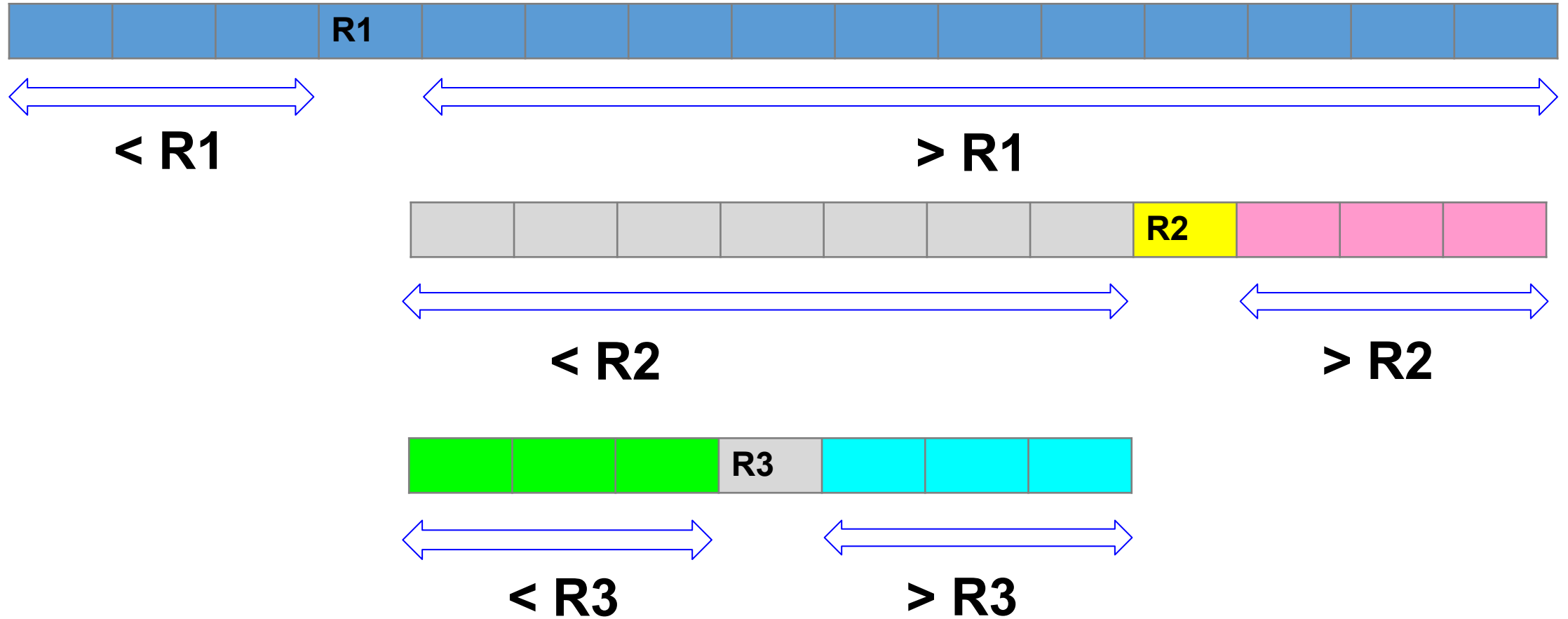


different states after one move



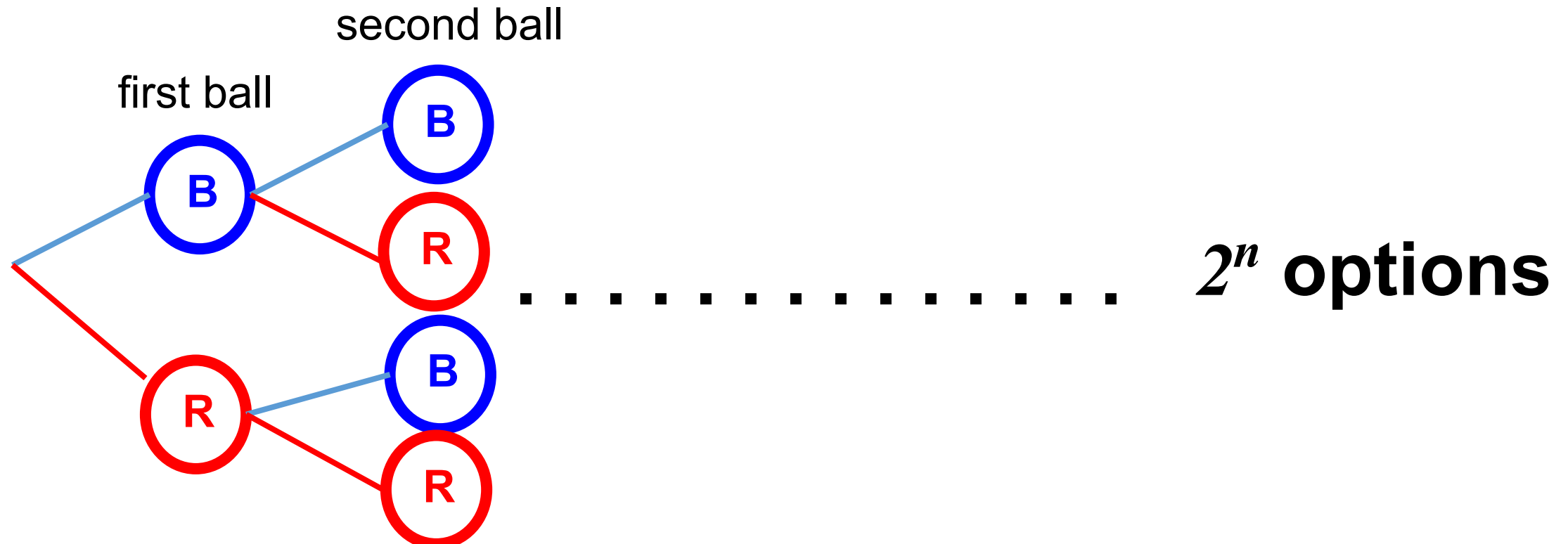


# Branch in Quick Sort (sorted data)

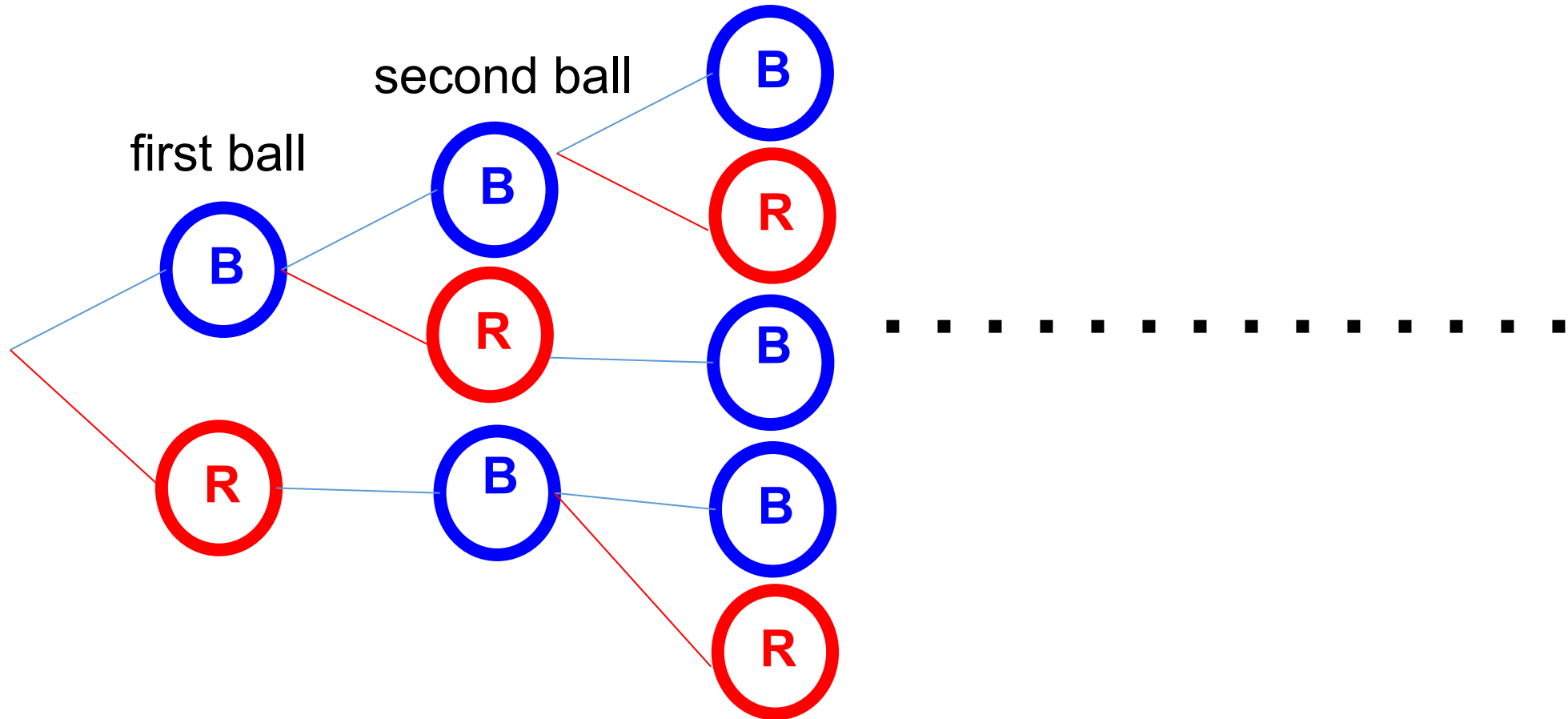


# Select Balls

- If you have an unlimited supply for red and blue balls, how many ways can you select  $n$  balls?
- Orders matter: Red – Blue is different from Blue – Red.



You have an unlimited supply for red and blue balls. Two adjacent **balls cannot be both Red**. How many ways can you select  $n$  balls? Orders matter.

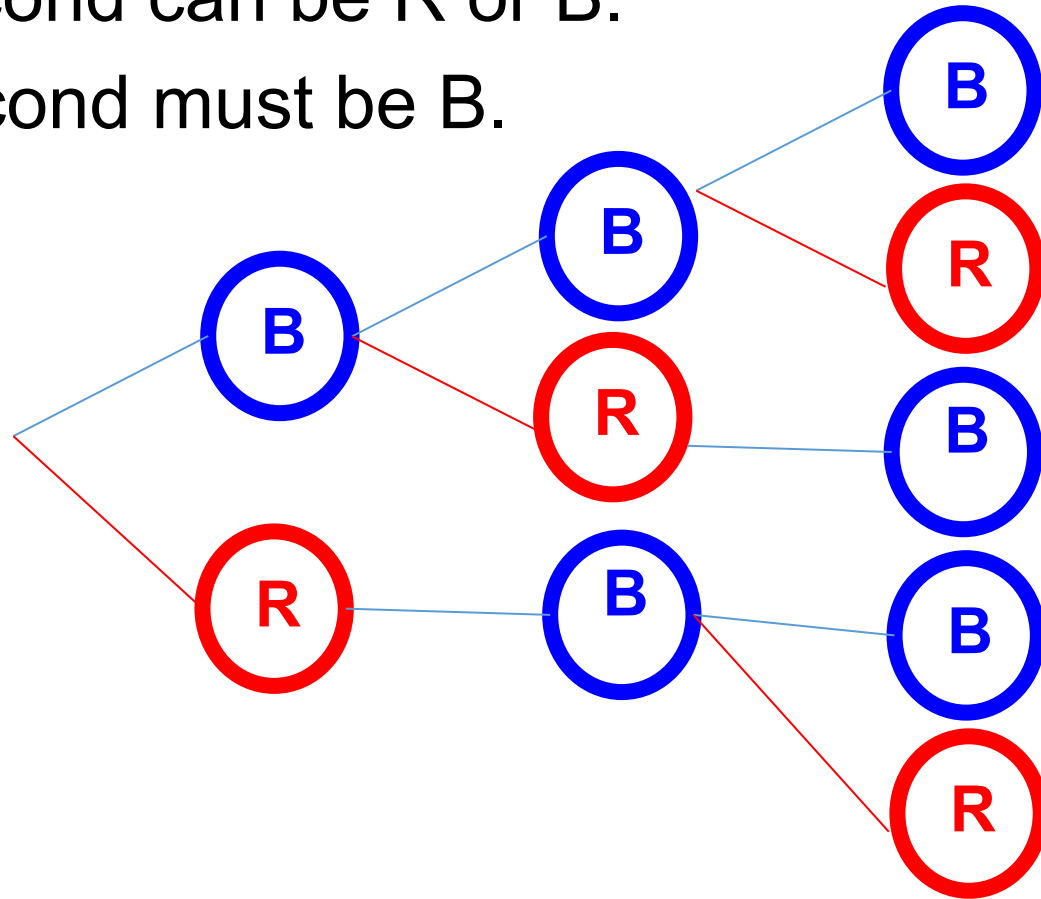


# First Approach: list answers

- one ball: two solutions: R or B
- two balls: three solutions: RB, BR, or BB
- three balls: five solutions: RBR, RBB, BRB, BBR, BBB

# Divide the problem

- If the first ball is B, the second can be R or B.
- If the first ball is R, the second must be B.



# Divide the problem

- If the first ball is B, the second can be R or B.
- If the first ball is R, the second must be B.
- Suppose  $f(n)$  means the number of options to select  $n$  balls.
- $f(1) = 2$  because there are two options to select 1 ball.
- $f(2) = 3$  because there are three options to select 2 balls.
- When  $n$  is larger, we shrink the problem slightly by selecting only one ball.

# Shrink the problem by one ball

- To select  $n$  balls, select one ball only.
- If B is selected, there is no restriction of the next ball.
- If the selected ball is B, the next can be R or B. The problem becomes selecting  $n-1$  balls.
- If the selected ball is R, the next must be B. . The problem becomes selecting  $n-2$  balls.
- Suppose  $f(n)$  means the number of options to select  $n$  balls.

$$f(n) = \begin{cases} f(n-1) & \text{first ball is B} \\ f(n-2) & \text{first ball is R} \end{cases}$$

# Addition or Multiplication?

$$f(n) = \begin{cases} f(n-1) & \text{first ball is B} \\ f(n-2) & \text{first ball is R} \end{cases}$$

$$f(n) = f(n-1) + f(n-2) \text{ or}$$

$$f(n) = f(n-1) \times f(n-2)$$



# Addition or Multiplication?

$$f(n) = \begin{cases} f(n-1) & \text{first ball is B} \\ f(n-2) & \text{first ball is R} \end{cases}$$

$$f(n) = f(n-1) + f(n-2) \text{ or}$$

$$f(n) = f(n-1) \times f(n-2)$$

if the two are either A or B, then add

if the two are independent, then multiply

# Order meal in a restaurant

- If a restaurant offers 4 options of beef, 3 options of chicken, 4 options of fish, and 5 options of salad, how much options do you have?  $4 + 3 + 4 + 5 = 16$



- If there are three options of dessert, how many ways can you order meal + dessert?  $16 \times 3 = 48$ .



# Addition or Multiplication?

$$f(n) = \begin{cases} f(n-1) & \text{first ball is B} \\ f(n-2) & \text{first ball is R} \end{cases}$$

$$f(n) = f(n-1) + f(n-2) \text{ or}$$

$$f(n) = f(n-1) \times f(n-2)$$

if the two are either A or B, then add

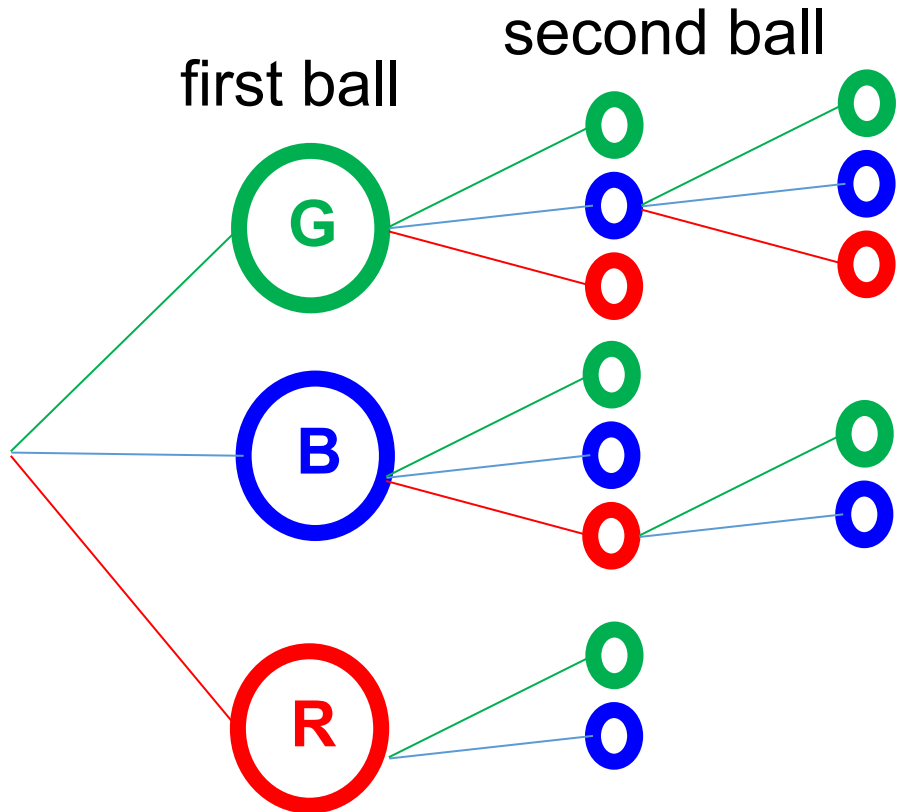
if the two are independent, then multiply

---

# Shrink the problem by one ball

- $f(1) = 2$  and  $f(2) = 3$ : stop condition: when  $n$  is 1 or 2
- $f(n) = f(n - 1) + f(n - 2)$
- $f(n - 1) = f(n - 2) + f(n - 3)$
- $f(n - 2) = f(n - 3) + f(n - 4)$
- $f(n - 3) = f(n - 4) + f(n - 5)$
- ...
- change:  $n$  becomes smaller and smaller
- recurring pattern

You have an unlimited supply for green, red, and blue balls. Two adjacent balls cannot be both Red. How many ways can you select  $n$  balls? Orders matter.



.....

# First Approach: list answers

- one ball: 3 solutions: G or R or B
- two balls: 8 solutions:
  1. RB, RG
  2. BR, BG, BB
  3. GR, GB, GG

# Divide the problem

- If the first ball is B or G, the second can be G, R, or B.
- If the first ball is R, the second can be G or B, not R.
- Suppose  $f(n)$  means the number of options to select  $n$  balls.
- $f(1) = 3$  because there are three options to select 1 ball.
- $f(2) = 8$  because there are eight options to select 2 balls.
- When  $n$  is larger, we shrink the problem slightly by selecting only one ball.

- Suppose  $f(n)$  means the number of options to select  $n$  balls.
- $g(n)$ : number of options to select  $n$  balls and first is G
- $b(n)$ : number of options to select  $n$  balls and first is B
- $r(n)$ : number of options to select  $n$  balls and first is R
- $g(n) = g(n-1) + r(n-1) + b(n-1)$
- $b(n) = g(n-1) + r(n-1) + b(n-1)$
- $r(n) = g(n-1) + b(n-1)$
- $f(n) = g(n) + r(n) + b(n)$
- $g(n) = f(n-1)$
- $b(n) = f(n-1)$
- $g(1) = b(1) = r(1) = 1$



<b>n</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
g(n)	1	3	8	22	60
b(n)	1	3	8	22	60
r(n)	1	2	6	16	44
f(n)	3	8	22	60	164

# Integer Partition

Divide a positive integer into the sum of one (the original integer) of multiple positive integers.

<b>1 =</b>	<b>1</b>					<b>4 =</b>	<b>1 +</b>	<b>1 +</b>	<b>1 +</b>	<b>1</b>
<b>2 =</b>	<b>1 +</b>	<b>1</b>					<b>1 +</b>	<b>1 +</b>	<b>2</b>	
	<b>2</b>						<b>1 +</b>	<b>2 +</b>	<b>1</b>	
<b>3 =</b>	<b>1 +</b>	<b>1 +</b>	<b>1</b>				<b>1 +</b>	<b>3</b>		
	<b>1 +</b>	<b>2</b>					<b>2 +</b>	<b>1 +</b>	<b>1</b>	
	<b>2 +</b>	<b>1</b>					<b>2 +</b>	<b>2</b>		
	<b>3</b>						<b>3 +</b>	<b>1</b>		
							<b>4</b>			

# How many ways can n be positioned?

n	1	2	3	4	...
partitions	1	2	4	8	?

# How many ways can $n$ be positioned?

$n$	1	2	3	4	...
partitions	1	2	4	8	?

- wrong ways to solve the problem: It **seems** that the answer is  $2^{n-1}$  ways to partition  $n$
- Why is this invalid? You cannot observe some examples to reach a conclusion.
- West Lafayette has no snow from May to September, can you conclude that it will not snow?
- For any number of  $(x, y)$  pairs, there is an infinite number of polynomials (with sufficient degrees) passing the pairs.

# Decide the first number

- If the original number is  $n$ , the first number can be  $1, 2, \dots, n$
- The remaining number is  $n-1, n-2, \dots, 0$
- Let  $f(n)$  be the number of ways to partition number  $n$
- If the first number is  $1$ , there are  $f(n-1)$  ways to partition  $n - 1$
- If the first number is  $2$ , there are  $f(n-2)$  ways to partition  $n - 2$
- ...
- If the first number is  $n-1$ , there are  $f(1)$  ways to partition  $1$
- If the first number is  $n$ , nothing is left
- $f(n) = f(n-1) + f(n-2) + \dots + f(1) + 1$

- $f(1) = 1$

- $f(n) = f(n-1) + f(n-2) + \dots + f(1) + 1$

- $f(n + 1) = f(n) + f(n-1) + f(n-2) + \dots + f(1) + 1$

- $f(n+1) - f(n) = f(n)$

- $f(1) = 1$

- ~~$f(n) = f(n-1) + f(n-2) + \dots + f(1) + 1$~~

- ~~$f(n + 1) = f(n) + f(n-1) + f(n-2) + \dots + f(1) + 1$~~

- $f(n+1) - f(n) = f(n)$

- $f(n+1) = 2f(n)$

- $f(n) = 2^{n-1}$

# Three components in recursion

- Stop Condition:  $f(1) = 1$
- Recurring pattern:  $f(n) = f(n-1) + f(n-2) + \dots + f(1) + 1$
- Changes:  $f(n)$  is expressed by  $n - 1, n - 2 \dots$



# Recursive Functions

```
function(arguments)
{
  check arguments for stop conditions
  if the stop conditions are not met.
  {
    change the arguments
    call function using the new arguments
  }
}
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
int f(int n)
{
    if (n <= 0) // stop condition
    {
        return 0;
    }
    int x = f(n - 1);
    int y = x + n;
    return y;
}
```

```
int main(int argc, char * * argv)
{
    int a = f(3);
    printf("a = %d\n", a);
    return EXIT_SUCCESS;
}
```

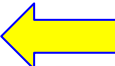
To understand recursive functions,  
we need to understand stack memory



**change from n to n - 1**

```
#include <stdio.h>
#include <stdlib.h>

int f(int n)
{
    if (n <= 0) // stop condition
    {
        return 0;
    }
    int x = f(n - 1);
    int y = x + n;
    return y;
}

int main(int argc, char * * argv)
{
    int a = f(3); 
    printf("a = %d\n", a);
    return EXIT_SUCCESS;
}
```

Frame	Symbol	Address	Value
main	a	100	U

```

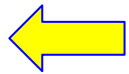
#include <stdio.h>
#include <stdlib.h>

int f(int n)
{
    if (n <= 0) // stop condition
    {
        return 0;
    }
    int x = f(n - 1);
    int y = x + n;
    return y;
}

int main(int argc, char * * argv)
{
    int a = f(3);
    printf("a = %d\n", a);
    return EXIT_SUCCESS;
}

```

Frame	Symbol	Address	Value
f	n	200	3
	value address 100		
	return location		
main	a	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1); ←
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

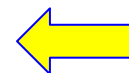
Frame	Symbol	Address	Value
f	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1); ←
11     int y = x + n; ←
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a); ←
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1); ←
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
return location line 18			
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1); ←
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

- f is called twice
- two frames in stack
- the frames are not “merged”

Frame	Symbol	Address	Value
f	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 -

```

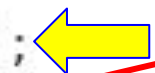
Frame	Symbol	Address	Value
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int f(int n)
5 {
6     if (n <= 0) // stop condition
7     {
8         return 0;
9     }
10    int x = f(n - 1);
11    int y = x + n;
12    return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1); ←
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	400	1
	value address <b>304</b>		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address <b>204</b>		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

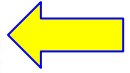
```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition ←
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int a
16 {
17     int a = f(3)
18     printf("a =
19     return EXIT_
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	500	0
	value address 404		
	return location line 11		

Frame	Symbol	Address	Value
f	y	408	U
	x	404	U
	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int a
16 {
17     int a = f(3)
18     printf("a =
19     return EXIT_
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	500	0
	value address 404		
	return location line 11		

Frame	Symbol	Address	Value
f	y	408	U
	x	404	U
	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0; ←
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int a
16 {
17     int a = f(3)
18     printf("a =
19     return EXIT_
20 }
21 _

```

Frame	Symbol	Address	Value
f	n	500	0
	value address 404		
	return location line 11		

Frame	Symbol	Address	Value
f	y	408	U
	x	404	U → 0
	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	408	U
	x	404	0
	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	408	1
	x	404	0
	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	408	1
	x	404	0
	n	400	1
	value address 304		
	return location line 11		
f	y	308	U
	x	304	U → 1
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n; ←
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
<del>f</del>	<del>y</del>	<del>408</del>	<del>1</del>
	<del>x</del>	<del>404</del>	<del>0</del>
	<del>n</del>	<del>400</del>	<del>1</del>
	<del>value address 304</del>		
	<del>return location line 11</del>		
f	y	308	U
	x	304	1
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	308	3
	x	304	1
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	308	3
	x	304	1
	n	300	2
	value address 204		
	return location line 11		
f	y	208	U
	x	204	U → 3
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n; ←
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
<del>f</del>	<del>y</del>	<del>308</del>	<del>3</del>
	<del>x</del>	<del>304</del>	<del>1</del>
	<del>n</del>	<del>300</del>	<del>2</del>
	<del>value address 204</del>		
	<del>return location line 11</del>		
f	y	208	U
	x	204	3
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	208	6
	x	204	3
	n	200	3
	value address 100		
	return location line 18		
main	a	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	208	6
	x	204	3
	n	200	3
	value address 100		
	return location line 18		
main	a	100	<del>U</del> <b>6</b>



```

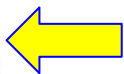
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int n)
5  {
6      if (n <= 0) // stop condition
7          {
8              return 0;
9          }
10     int x = f(n - 1);
11     int y = x + n;
12     return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17     int a = f(3);
18     printf("a = %d\n", a);
19     return EXIT_SUCCESS;
20 }
21 _

```

Frame	Symbol	Address	Value
f	y	208	6
	x	204	3
	n	200	3
	value address 100		
	return location line 18		
main	a	100	6

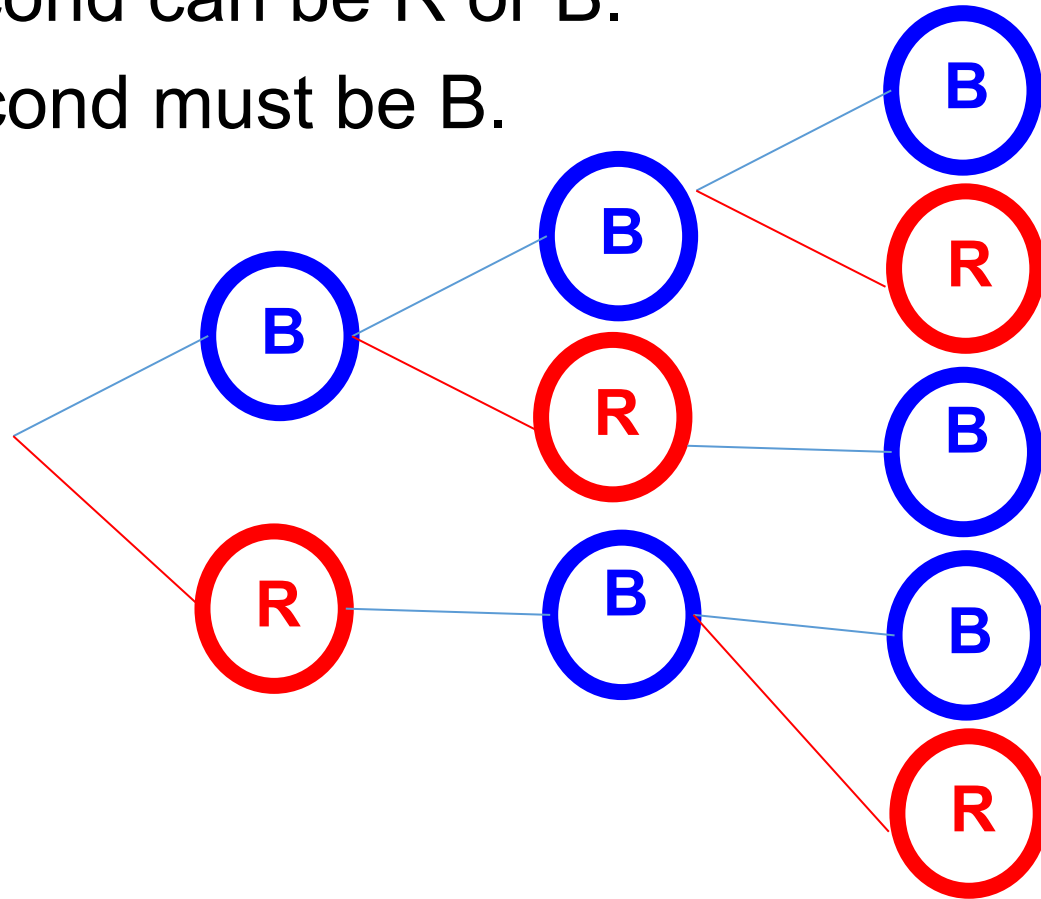
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int f(int n)
5 {
6     if (n <= 0) // stop condition
7     {
8         return 0;
9     }
10    int x = f(n - 1);
11    int y = x + n;
12    return y;
13 }
14
15 int main(int argc, char * * argv)
16 {
17    int a = f(3);
18    printf("a = %d\n", a);
19    return EXIT_SUCCESS;
20 }
21 _
```

Frame	Symbol	Address	Value
main	a	100	6



# Divide the problem

- If the first ball is B, the second can be R or B.
- If the first ball is R, the second must be B.



# Shrink the problem by one ball

- To select  $n$  balls, select one ball only.
- If B is selected, there is no restriction of the next ball.
- If the selected ball is B, the next can be R or B. The problem becomes selecting  $n-1$  balls.
- If the selected ball is R, the next must be B. . The problem becomes selecting  $n-2$  balls.
- Suppose  $f(n)$  means the number of options to select  $n$  balls.

$$f(n) = \begin{cases} f(n-1) & \text{first ball is B} \\ f(n-2) & \text{first ball is R} \end{cases}$$

# Shrink the problem by one ball

- $f(1) = 2$  and  $f(2) = 3$ : stop condition: when  $n$  is 1 or 2
- $f(n) = f(n - 1) + f(n - 2)$
- $f(n - 1) = f(n - 2) + f(n - 3)$
- $f(n - 2) = f(n - 3) + f(n - 4)$
- $f(n - 3) = f(n - 4) + f(n - 5)$
- ...
- change:  $n$  becomes smaller and smaller
- recurring pattern

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 int f(int n)
5 {
6     if (n == 1) { return 2; }
7     if (n == 2) { return 3; }
8     int a = f(n - 1);
9     int b = f(n - 2);
10    int c = a + b;
11    return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16    int x = f(4); ←
17    printf("x = %d\n", x);
18    return EXIT_SUCCESS;
19 }
```

Frame	Symbol	Address	Value
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1); ←
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
return location line 17			
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1); ←
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	312	U
	b	308	U
	a	304	U
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; } ←
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	400	2
	value address 304		
	return location line 9		
f	c	312	U
	b	308	U
	a	304	U
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
return location line 17			
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; } ←
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	400	2
	value address 304		
	return location line 9		
f	c	312	U
	b	308	U
	a	304	<del>U</del> <b>3</b>
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
return location line 17			
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2); ←
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
<del>f</del>	<del>n</del>	<del>400</del>	<del>2</del>
	<del>value address 304</del>		
	<del>return location line 9</del>		
f	c	312	U
	b	308	U
	a	304	3
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
return location line 17			
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2); ←
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	400	1
	value address <b>308</b>		
	return location line <b>10</b>		
f	c	312	U
	b	308	U
	a	304	3
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; } ←
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	400	1
	value address 308		
	return location line 10		
f	c	312	U
	b	308	2
	a	304	3
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b; ←
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
<del>f</del>	<del>n</del>	<del>400</del>	<del>1</del>
	<del>value address 308</del>		
	<del>return location line 10</del>		
f	c	312	U
	b	308	2
	a	304	3
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U



```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	312	5
	b	308	2
	a	304	3
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	U
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	312	5
	b	308	2
	a	304	3
	n	300	3
	value address 204		
	return location line 9		
f	c	212	U
	b	208	U
	a	204	<del>U</del> <b>5</b>
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2); ←
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	212	U
	b	208	U
	a	204	5
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	300	2
	value address <b>208</b>		
	return location line <b>10</b>		
f	c	212	U
	b	208	U
	a	204	5
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; } ←
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	300	2
	value address 208		
	return location line 10		
f	c	212	U
	b	208	U
	a	204	5
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; } ←
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	n	300	2
	value address 208		
	return location line 10		
f	c	212	U
	b	208	3
	a	204	5
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	212	8
	b	208	3
	a	204	5
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c; ←
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value
f	c	212	8
	b	208	3
	a	204	5
	n	200	4
	value address 100		
	return location line 17		
main	x	100	U



```

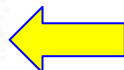
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x); ←
18     return EXIT_SUCCESS;
19 }

```

Frame	Symbol	Address	Value	
f	c	212	8	
	b	208	3	
	a	204	5	
	n	200	4	
	value address 100			
	return location line 17			
main	x	100	8 ←	

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  int f(int n)
5  {
6      if (n == 1) { return 2; }
7      if (n == 2) { return 3; }
8      int a = f(n - 1);
9      int b = f(n - 2);
10     int c = a + b;
11     return c;
12 }
13
14 int main(int argc, char * * argv)
15 {
16     int x = f(4);
17     printf("x = %d\n", x);
18     return EXIT_SUCCESS;
19 }
```

Frame	Symbol	Address	Value
main	x	100	8



# List Options for Selecting Balls

# Output of the Program

RBRB

RBBR

RBBB

BRBR

BRBB

BBRB

BBBR

BBBB

- Red and Blue balls
- Two adjacent balls must not be both Red

```
int main(int argc, char * * argv)
{
    if (argc < 2)
    {
        return EXIT_FAILURE;
    }
    int numball = (int) strtol(argv[1], NULL, 10);
    char * colors = malloc(sizeof(char) * numball);
    selectBall(colors, numball, 0);
    free (colors);
    return EXIT_SUCCESS;
}
```

# selectBall

```
void selectBall(char *selected, int total, int numselected) {
```

- selected: Array of selected balls.
- total: Total number of balls to be selected.
- numselected: Number of balls selected.

# Stopping condition

```
void selectBall(char *selected, int total, int numselected) {  
  
    if (total == numselected) {  
        int i;  
        for (i=0; i < total; i++)  
        {  
            printf("%c", selected[i]);  
        }  
        printf("\n");  
        return;  
    }  
}
```

# Selecting Ball

```
void selectBall(char *selected, int total, int numselected) {
```

- How many ways to select a ball?  
2: Red or Blue



# Selecting Ball

```
void selectBall(char *selected, int total, int numselected) {
```

- How many ways to select a ball?  
2: Red or Blue

# Selecting Ball

```
void selectBall(char *selected, int total, int numselected) {
```

```
    selected[numselected] = 'R';  
    selectBall(selected, total, numselected+1);
```

```
    selected[numselected] = 'B';  
    selectBall(selected, total, numselected+1);
```

Is this okay?

# Selecting Ball

```
void selectBall(char *selected, int total, int numselected) {
```

**Two adjacent balls must not be both  
Red**

```
    selected[numselected] = 'R';  
    selectBall(selected, total, numselected+1);
```

```
    selected[numselected] = 'B';  
    selectBall(selected, total, numselected+1);
```

**Is this okay?**

# Selecting Ball

```
void selectBall(char *selected, int total, int numselected) {
```

```
    selected[numselected] = 'R';
```

```
    selected[numselected+1] = 'B';
```

```
    selectBall(selected, total, numselected+2);
```

```
    selected[numselected] = 'B';
```

```
    selectBall(selected, total, numselected+1);
```

Is this okay?

# Selecting Ball

```
void selectBall(char *selected, int total, int numselected) {
```

**What happens if this is the last ball? i.e., numselected == (total - 1)**

```
selected[numselected] = 'R';
```

```
selected[numselected+1] = 'B';
```

```
selectBall(selected, total, numselected+2);
```

```
selected[numselected] = 'B';
```

```
selectBall(selected, total, numselected+1);
```

Is this okay?

# Integer Partition

Divide a positive integer into the sum of one (the original integer) or multiple positive integers.

<b>1 =</b>	<b>1</b>					<b>4 =</b>	<b>1 +</b>	<b>1 +</b>	<b>1 +</b>	<b>1</b>
<b>2 =</b>	<b>1 +</b>	<b>1</b>					<b>1 +</b>	<b>1 +</b>	<b>2</b>	
	<b>2</b>						<b>1 +</b>	<b>2 +</b>	<b>1</b>	
<b>3 =</b>	<b>1 +</b>	<b>1 +</b>	<b>1</b>				<b>1 +</b>	<b>3</b>		
	<b>1 +</b>	<b>2</b>					<b>2 +</b>	<b>1 +</b>	<b>1</b>	
	<b>2 +</b>	<b>1</b>					<b>2 +</b>	<b>2</b>		
	<b>3</b>						<b>3 +</b>	<b>1</b>		
							<b>4</b>			

# Decide the first number

- If the original number is  $n$ , the first number can be  $1, 2, \dots, n$
- The remaining number is  $n-1, n-2, \dots, 0$
- Let  $f(n)$  be the number of ways to partition number  $n$
- If the first number is  $1$ , there are  $f(n-1)$  ways to partition  $n - 1$
- If the first number is  $2$ , there are  $f(n-2)$  ways to partition  $n - 2$
- ...
- If the first number is  $n-1$ , there are  $f(1)$  ways to partition  $1$
- If the first number is  $n$ , nothing is left
- $f(n) = f(n-1) + f(n-2) + \dots + f(1) + 1$

# **How to Solve Problems using Recursion**



# Decide to use recursion

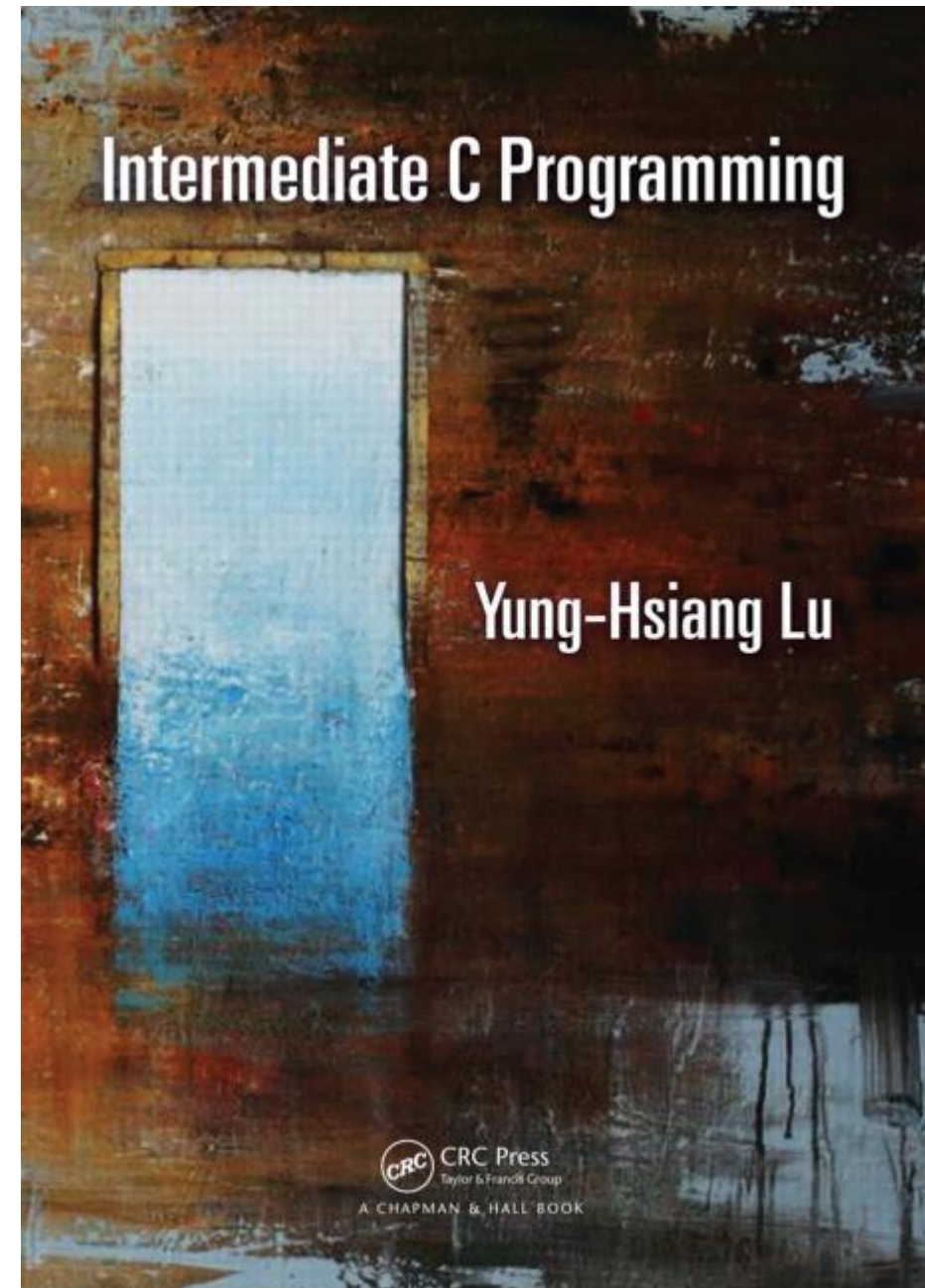
- Is the number of iterations known? Yes: consider for
- Is the condition controlled by one variable? Yes: consider while
- Are there branches? Yes: consider recursion
  - Ball selection: Is the previous ball red or blue?
  - Integer partition: the first number can be 1, 2, ..., n
- Does the problem look like itself after simplification?
- Can the problem become smaller gradually?

# Design Recursion Solutions

- Identify the stop conditions or conditions
- Recognize the patterns of the changes
- Determine what controls the recursion (the changes)

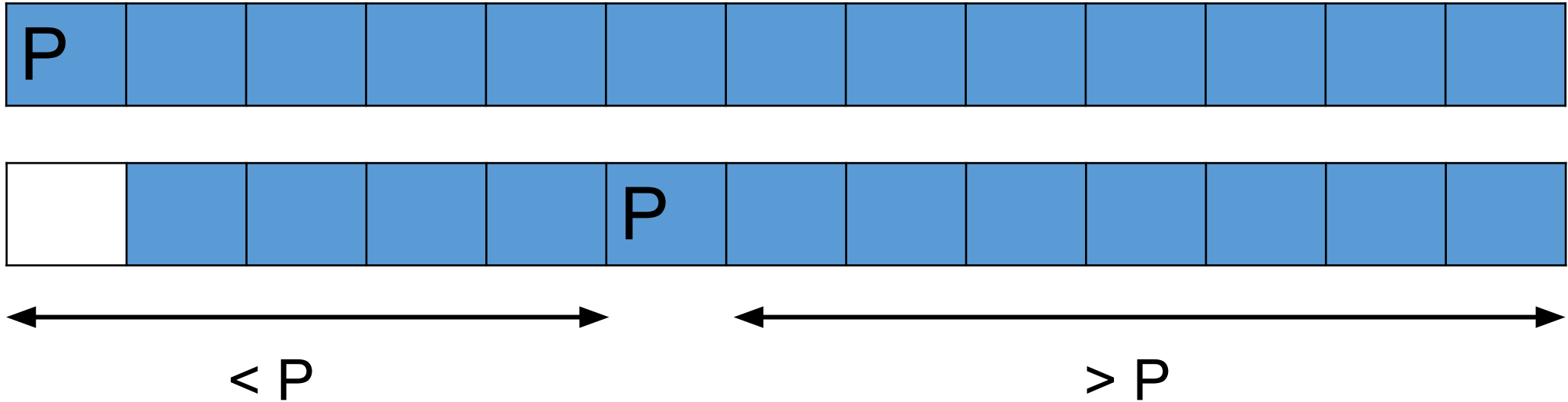
# Success needs practice

- This book has many examples.
- Electronic copy is available in Purdue Library.



# Quick Sort

# Quick Sort



Transitivity: If  $a > b$  and  $b > c$ , then  $a > c$ .

No need to compare  $a$  and  $c$ .

This is how **quick sort** works.

# Method

1. Find a pivot (usually the first element)
2. Keep two indexes: low and high
3. Move low toward high if the value is smaller than the pivot
4. Move high toward low if the value is greater than the pivot
5. If low and high not crossed yet, swap
6. Continue until high and low cross
7. Move pivot
8. Sort part smaller than pivot
9. Sort part greater than pivot

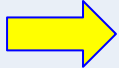






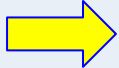
index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p	low										high


index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p		low									high

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low								high

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p	low										high

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p		low									high

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low								high

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low							high	

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low						high		


index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low						high		

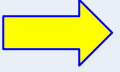
index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p			low			<b>swap</b>			high		

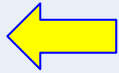


index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low						high		


index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p			low			<b>swap</b>			high		

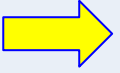


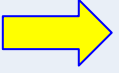
index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p				low					high		

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	23	8	31	6	42	28	16	51	33
variable	p			low						high		

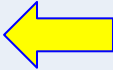
index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p			low				<b>swap</b>		high		

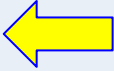


index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p				low					high		

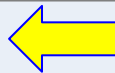
index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low				high		


index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low			high			

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low			high			

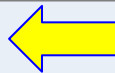
index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low	high					




index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low			high			

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low	high					

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	6	31	42	28	23	51	33
variable	p					low	high	 <b>swap</b>				

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low			high			

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	31	6	42	28	23	51	33
variable	p					low	high					

index	0	1	2	3	4	5	6	7	8	9	10	11		
value	19	7	12	16	8	6	31	42	28	23	51	33		
variable	p					low	high	 swap						

index	0	1	2	3	4	5	6	7	8	9	10	11
value	19	7	12	16	8	6	31	42	28	23	51	33
variable	p					high	low					

index	0	1	2	3	4	5	6	7	8	9	10	11
value	6	7	12	16	8	19	31	42	28	23	51	33
variable						high	low					

index	0	1	2	3	4	5	6	7	8	9	10	11
value	6	7	12	16	8	19	31	42	28	23	51	33
variable	< 19						> 19					

```
void quickSort(int * arr, int len)
{
    quickSortHelp(arr, 0, len - 1);
}
static void quickSortHelp(int * arr, int first, int last)
{
    // [first, last]: range of valid indexes (not last - 1)
    if (first >= last) // no need to sort one or no element
    {
        return;
    }
    int pivot = arr[first];
    int low = first + 1;
    int high = last;
```

# “Helper” functions in recursion

- It is common to create “helper” functions in recursion in order to get the right arguments (often more arguments)
- The arguments can be used to keep track of progress
- The helper functions may be static. They are visible only within the same file. They cannot be called from any other file.
- Do not use static variables. Static variables keep values from previous calls and can be confusing.
- You can use static functions, not static variables.

```
while (low < high)
{
    while ((low < last) && (arr[low] <= pivot))
    {
        // <= so that low will increment when arr[low]
        // is the same as pivot, using < will stop
        // incrementing low when arr[low] is the same
        // as pivot and the outer while loop will not stop
        low ++;
    }
    while ((first < high) && (arr[high] > pivot))
    {
        high --;
    }
}
```

```
    if (low < high)
    {
        swap (& arr[low], & arr[high]);
    }
} // while (low < high)
if (pivot > arr[high])
{
    swap(& arr[first], & arr[high]);
}
quickSortHelp(arr, first, high - 1);
quickSortHelp(arr, low, last);
}
```