

課題 2

「関数と再帰呼び出し」

伊藤 康一, 片岡 駿

2016 年度プログラミング演習A

課題2の目標

ソートィングアルゴリズムを通して、
関数の使い方を学ぼう

■ ソートィングとは？

データを指定された順序に並べ替えること

例：小さい順に並べる

0, 6, 4, 5, 3  0, 3, 4, 5, 6

さまざまなソーティングアルゴリズム

- 選択ソート
- マージソート
- ヒープソート
- クイックソート
- ．．．

ソーティングの結果自体はどれも同じ

アルゴリズムによる違いは？

- 演算回数（計算時間）
- 必要とするメモリ
- コーディングの簡潔さ

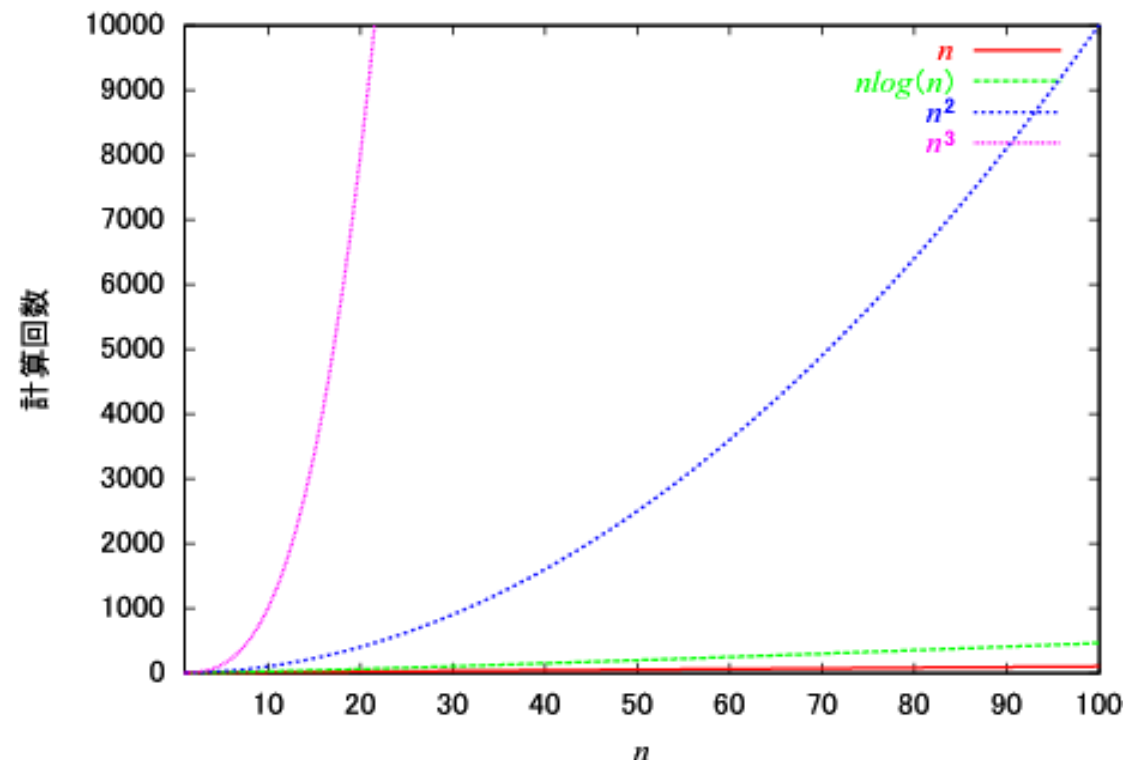
ソーティングには工夫が必要

大量のデータを扱う場合には、計算方法により計算時間に大きな差が出る

例：

データ数が $n=100$ のとき、演算が100回必要ならば、

- n^2 のとき：10,000回
- n^3 のとき：1,000,000回



さまざまなソーティングアルゴリズム

- 選択ソート
- マージソート

本課題で出題

- ヒープソート
- クイックソート
- ...

選択ソート

選択ソートアルゴリズム

[入力]

d: 整列したい値を格納している配列

n: 配列dの要素数

[擬似コード]

1. $i = 0$ から $(n-2)$ まで, 以下の処理を繰り返す
2. $j \leftarrow d[i], d[i+1], \dots, d[n-1]$ の中で最小の要素を持つインデックス番号
3. $d[i]$ と $d[j]$ の値を入れ替える

[出力]

d: 整列済みの配列

先頭と比較して小さければ置き換える

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
入力	12	19	3	8	25	18	100	1	

3を最小値候補として記憶しておく

3があったx[2]の後から比較する

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
入力	12	19	3	8	25	18	100	1	

最小値1と12を置き換え

こういう比較をすると、結果的に最小値が先頭に移る

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
入力	1	19	3	8	25	18	100	12	7回

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
入力	12	19	3	8	25	18	100	1	

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
i=0	1	19	3	8	25	18	100	12	7回

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
i=1	1	3	19	8	25	18	100	12	6回

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
i=2	1	3	8	19	25	18	100	12	5回

結果的にいつも最小値が先頭に移っていく

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	比較
入力	12	19	3	8	25	18	100	1	
i=0	1	19	3	8	25	18	100	12	7回
i=1	1	3	19	8	25	18	100	12	6回
i=2	1	3	8	19	25	18	100	12	5回
i=3	1	3	8	12	25	18	100	19	4回
i=4	1	3	8	12	18	25	100	19	3回
i=5	1	3	8	12	18	19	100	25	2回
i=6	1	3	8	12	18	19	25	100	1回
								合計	28回

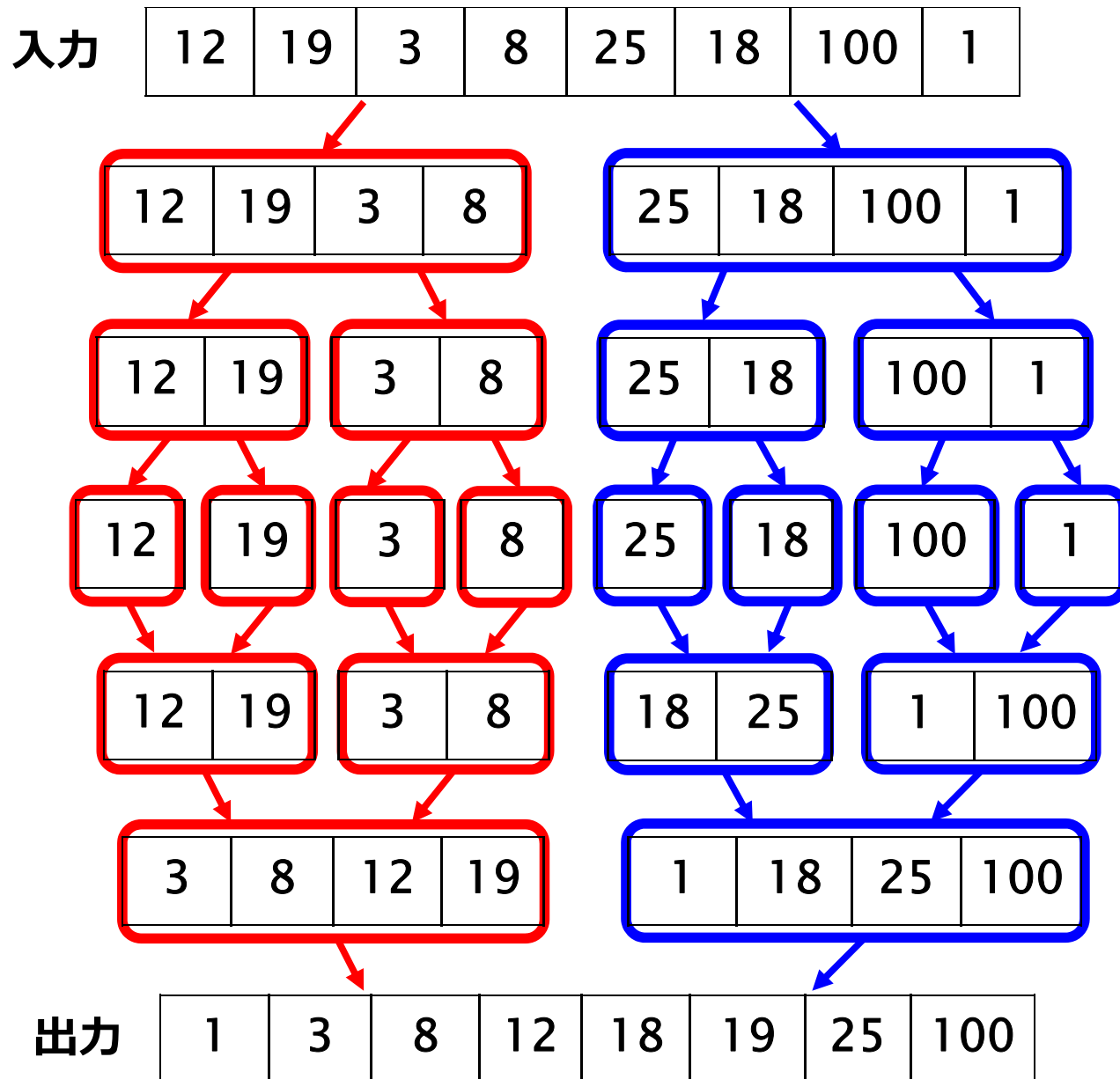
選択ソートの欠点

同じ比較を何回も行うので

無駄が多い・・・

マージソート

マージソートによるソートの流れ



マージ (merge) :
混合する, 結合する

半分ずつの要素に分割

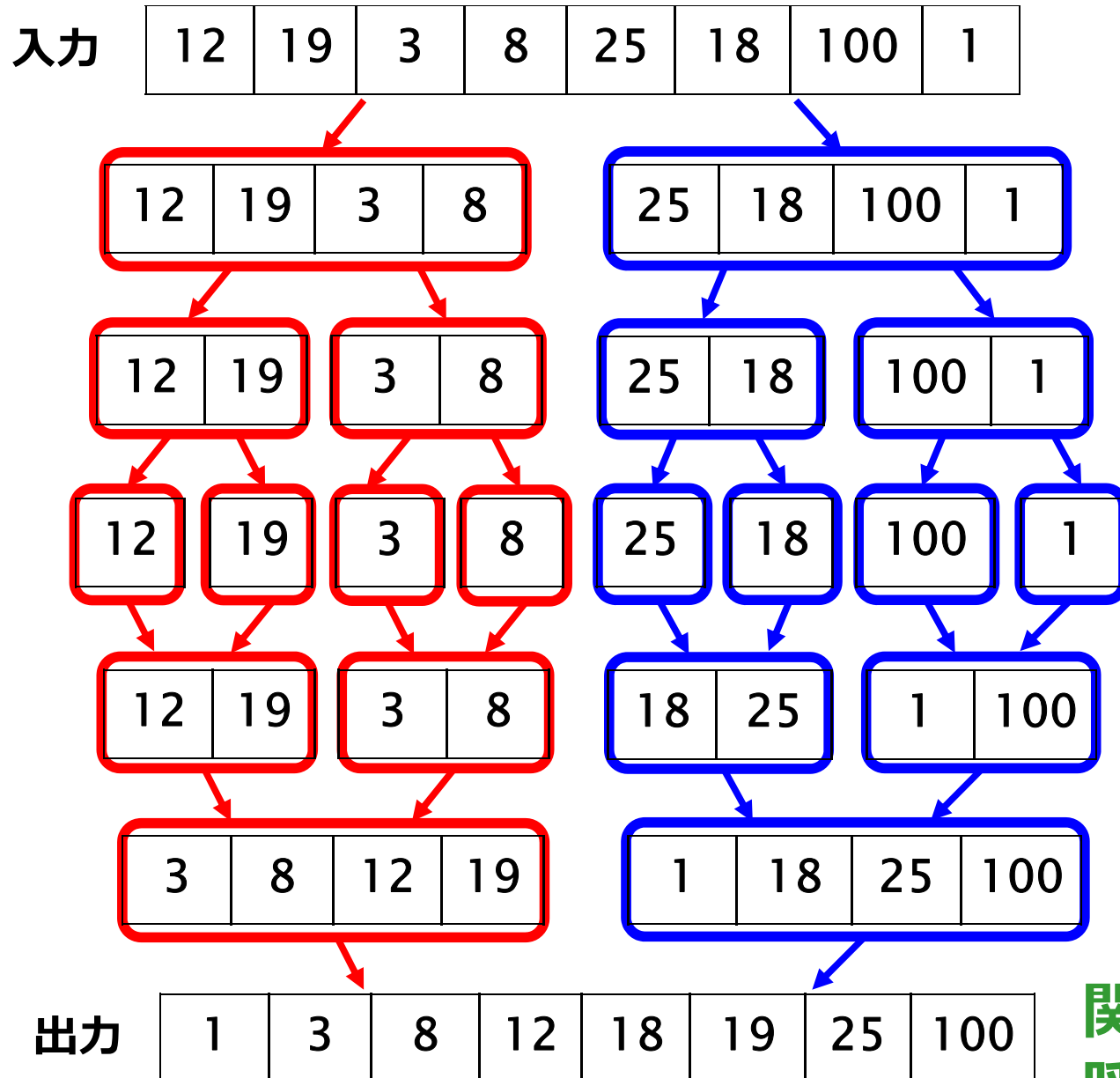


1個になるまで分割
→ 小さい順にソート済



分割し終わった後に,
ソートされた要素を
マージしていく

マージソートによるソートの流れ



繰り返す

半分ずつの要素に分割

1個になるまで分割
→ 小さい順にソート済

繰り返す

分割し終わった後に、
ソートされた要素を
マージしていく

関数化と関数の再帰
呼び出しで実現

再帰呼び出しとは？

手続きや関数内で自分自身を呼び出すこと

 短く簡潔なプログラムを実現できる

具体的には,

- ◆ merge関数

- ◆ mergesort関数

を定義し, 必要に応じて再帰的に呼び出す

merge関数

/* 部分配列 $d[p] \sim d[q-1]$ と $d[q] \sim d[r-1]$ をマージする */

merge (d, p, q, r)

n1 = q - p

n2 = r - q

n1個の要素からなる配列Lと, n2個の要素からなる配列Rを確保する

for i = 0 to n1-1

 L[i] = d[p + i]

for i = 0 to n2-1

 R[i] = d[q + i]

i = 0

j = 0

k = p

while (i < n1 and j < n2)

 if L[i] < R[j] then

 d[k] = L[i]

 i = i + 1

 k = k + 1

 else

 d[k] = R[j]

 j = j + 1

 k = k + 1

if i < n1 then

 copy L[i] ... L[n1-1] to d[k] ... d[r-1]

else

 copy R[j] ... R[n2-1] to d[k] ... d[r-1]

L[0] L[1] L[2] L[3]

3	8	12	19
---	---	----	----


R[0] R[1] R[2] R[3]

1	18	25	100
---	----	----	-----

1	3	8	12	18	19	25	100
d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]

mergesort関数

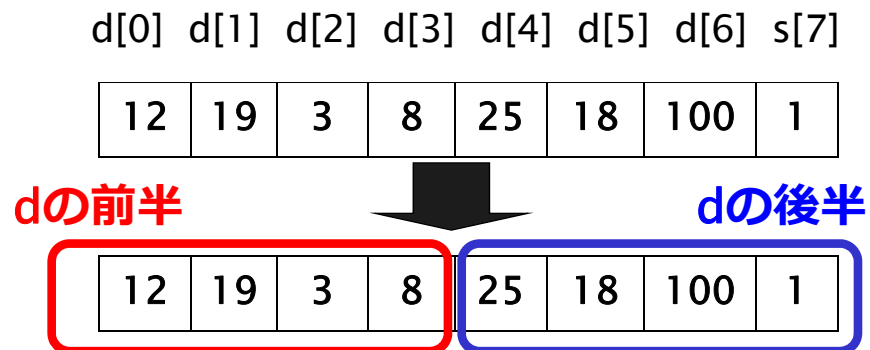
/* 部分配列 $d[p] \sim d[r-1]$ をソートする */



```
mergesort ( d, p, r )  
if  $p + 1 < r$   
   $q = ( p + r ) / 2$   
  mergesort ( d, p, q )  
  mergesort ( d, q, r )  
  merge ( d, p, q, r )
```

再帰呼び出し：

mergesort関数を繰り返し使う



mergesort関数の動作について 具体例で確認しよう

※ 以降の具体例では, mergesort() を
sort() と記載しています

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

sort(0, 7)は, 次の配列a, bをマージして得られる.

配列a = sort(0, 3) = { ?, ?, ?, ? }

配列b = sort(4, 7) = { ?, ?, ?, ? }

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)									
sort(2, 3)									
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

sort(0, 3)は, 次の配列a, bをマージして得られる.

配列a = sort(0, 1) = { ?, ? }

配列b = sort(2, 3) = { ?, ? }

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)									
sort(2, 3)									
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

sort(0, 1)は、次の配列a, bをマージして得られる。

配列a = sort(0, 0) = { 12 }

配列b = sort(1, 1) = { 19 }

要素数が1つの配列はすでにソート済み

マージソートの例題

	0	1	2	3	4	5	6	7	
	12	19	8	25	18	100	1		比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	
sort(2, 3)									
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	<u>12</u>	<u>19</u>	3	8	25	18	100	1	1回
sort(2, 3)									
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)									
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

sort(2, 3)は, 次の配列a, bをマージして得られる.

配列a = sort(2, 2) = { 3 }

配列b = sort(3, 3) = { 8 }

要素数が1つの配列はすでにソート済み

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	配列a	3	8	配列b	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	<u>3</u>	<u>8</u>	25	18	100	1	1回
sort(0, 3)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

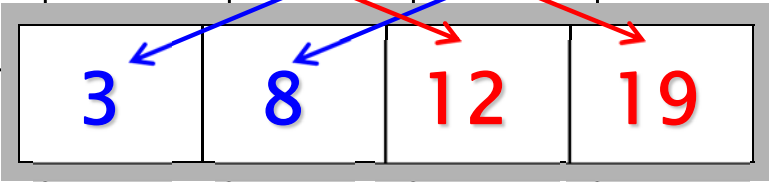
	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	<u>3</u>	<u>8</u>	25	18	100	1	1回
sort(0, 3)	12	19	3	8	25	18	100	1	
<div> <div>配列a</div> <div>配列b</div> <div> sort(0, 3)は、次の配列a, bをマージして得られる。 配列a = sort(0, 1) = { 12, 19 } 配列b = sort(2, 3) = { 3, 8 } </div> </div>									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	12	19	3	8	25	18	100	1	
sort(4, 7)									
sort(0, 7)									

配列a

配列b



比較回数 2回

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	<u>3</u>	<u>8</u>	<u>12</u>	<u>19</u>	25	18	100	1	2回
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)									
sort(6, 7)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	25	18	100	1	
sort(6, 7)									
sort(4, 7)									
sort(0, 7)									

配列a

配列b

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	<u>18</u>	<u>25</u>	100	1	1回
sort(6, 7)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	18	25	100	1	1回
sort(6, 7)									
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	1	配列a		100	1
sort(6, 7)	3	8	12	19	18	25	100	1	配列b
sort(4, 7)									
sort(0, 7)									

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	18	25	100	1	1回
sort(6, 7)	3	8	12	19	18	25	<u>1</u>	<u>100</u>	1回
sort(4, 7)									
sort(0, 7)									

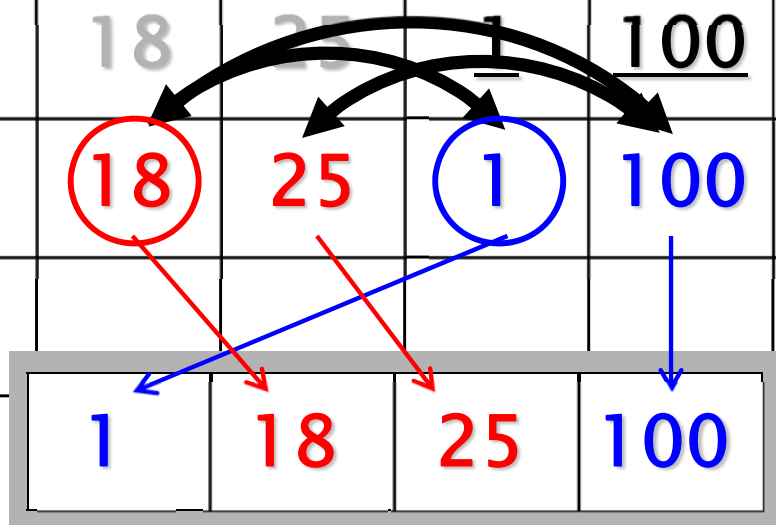
マージソートの例題

	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	18	25	100	1	1回
sort(6, 7)	3	8	1	配列a	18	25	<u>1</u>	<u>100</u>	配列b
sort(4, 7)	3	8	12	19	18	25	1	100	
sort(0, 7)									

マージソートの例題

	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	比較回数 3回				1回
sort(6, 7)	3	8	12	19	18	25	1	100	1回
sort(4, 7)	3	8	12	19	18	25	1	100	
sort(0, 7)									

比較回数 3回



マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	18	25	100	1	1回
sort(6, 7)	3	8	12	19	18	25	1	100	1回
sort(4, 7)	3	8	12	19	<u>1</u>	<u>18</u>	<u>25</u>	<u>100</u>	3回
sort(0, 7)									

マージソートの例題

	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	18	25	100	1	1回
sort(6, 7)	3	8	12	19	18	25	1	100	1回
so	3	8	12	19	1	18	25	100	配列b
sort(0, 7)	3	8	12	19	1	18	25	100	

配列a

配列b

マージソートの例題

	0	1	2	3	4	5	6	7	
入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	比較回数 6回			5	10	比較なし	
sort(6, 7)									2回
sort(4, 7)									2回
sort(0, 7)	3	8	12	19	1	18	25	100	

マージソートの例題

0 1 2 3 4 5 6 7

入力	12	19	3	8	25	18	100	1	比較回数
sort(0, 1)	12	19	3	8	25	18	100	1	1回
sort(2, 3)	12	19	3	8	25	18	100	1	1回
sort(0, 3)	3	8	12	19	25	18	100	1	2回
sort(4, 5)	3	8	12	19	18	25	100	1	1回
sort(6, 7)	3	8	12	19	18	25	1	100	1回
sort(4, 7)	3	8	12	19	1	18	25	100	3回
sort(0, 7)	<u>1</u>	<u>3</u>	<u>8</u>	<u>12</u>	<u>18</u>	<u>19</u>	<u>25</u>	<u>100</u>	6回
								合計	15回

問題2-1 関数と動的メモリ確保

`int func(int x);` ← 関数のプロトタイプ宣言
戻り値の型 関数名 (引数の型)

`int main()`
{
 `int a;`
 `printf("%d * 2 = %d", a, func(a));`
 `return 0;`
}

int型関数の戻り値はint型の変数と同じ
↓

`int func(int x)` ← 関数の定義
{
 `return 2 * x;` ← returnで値を返す
}

関数を使う前に宣言をする必要がある

問題2-1 関数と動的メモリ確保

```
int func( int x ) ← 戻り値をint型とする場合
{
    return 2 * x;
}
```

```
double func(double x) ← 戻り値をdouble型とする場合
{
    return 2*x;
}
```

```
void func(int x) ← 戻り値がない場合
{
    printf( "%d\n", 2*x );
}
```

関数の型は、戻り値の型に従って適切に決めよう

(補足) int型のmain関数とは・・・

先ほどの例 (赤字に注目)

```
int main()  ← int型として定義されている
{
    int a;
    printf( "%d * 2 = %d", a, func(a) );
    return 0; ← プログラムの最後で0を返す
}
```

- Int型のmain関数から返される値はプログラム終了時にOSに渡されるエラーコードになる
 - ◆ 0ならば正常終了
 - ◆ 0以外ならば異常終了

問題2-1 関数と動的メモリ確保

メモリ確保の問題

- 変数宣言時に余裕を持って記憶領域を確保しなければならない
- 確保した領域よりも1バイトでも多く使うとエラーになる

動的メモリ割り当て (malloc, free)

- 必要なメモリを必要なときに空いているメモリ空間から確保し, 不要になったら解放する

問題2-1 関数と動的メモリ確保

`void *malloc(size_t size)`

`size_t` は、符号のない整数型 (unsigned intと同じ)

戻り値は、void型へのポインタ

使用例：x[10]のためのメモリ領域をmallocで確保

`int *x;`

`x = (int *)malloc(sizeof(int) * 10);`

`if (x==NULL) {`

`printf("メモリが確保できません\n");`

`return 1;`

`}`

(処理)

`free(x);`

int型の大きさの要素
を10個確保する



問題2-1 関数と動的メモリ確保

■ C言語の最近の追加機能（C99）

（プログラム例）

```
void func(){  
    int n;  
    scanf("%d",&n);  
    int d[n];  
    ...  
}
```

メモリの開放も自動で行われて便利

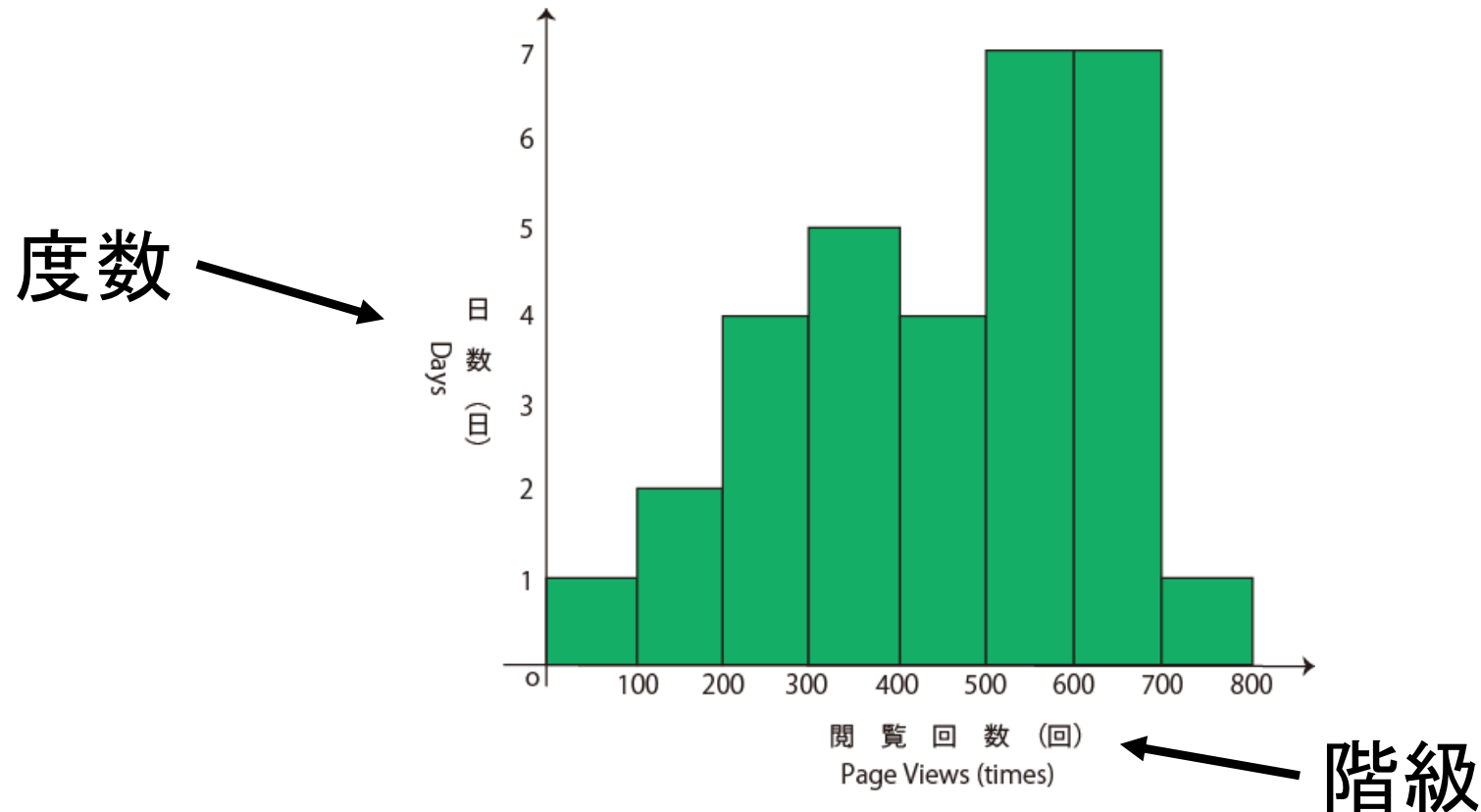
注意点

- mallocによる方法と違い、グローバル変数としては使えない（関数内部でしか動作しない）
- コンパイラによってはきちんと対応しきっていない（使えない場合がある）

問題に指定のある場合には必ずmallocを用いること

問題2-1 ヒストグラム

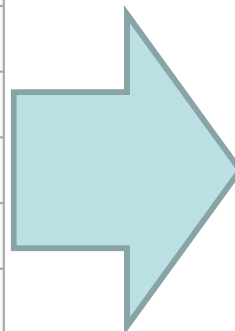
- ヒストグラムとは、片方の軸に度数、もう片方の軸に階級（ビン）をとったグラフの一種である



ウィキペディア日本語版の記事「ヒストグラム」(当記事)の2013年1月の閲覧回数

問題2-1 ヒストグラム

日	閲覧回数	日	閲覧回数
1	78	16	625
2	126	17	606
3	156	18	483
4	231	19	377
5	215	20	370
6	304	21	587
7	484	22	667
8	544	23	643
9	566	24	756
10	545	25	505
11	478	26	436
12	258	27	399
13	225	28	611
14	373	29	679
15	620	30	575
		31	565



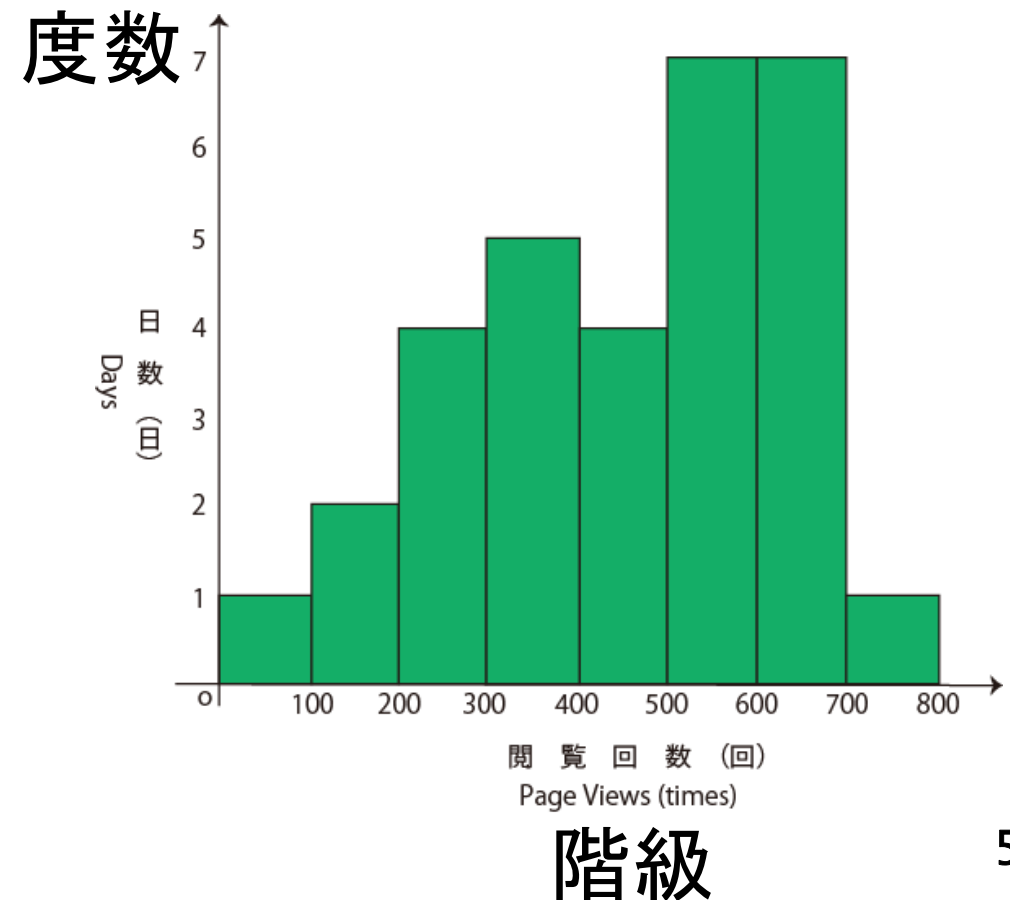
階級	度数
閲覧回数	その回数を記録した日数
0 - 99	1
100 - 199	2
200 - 299	4
300 - 399	5
400 - 499	4
500 - 599	7
600 - 699	7
700 - 799	1

ウィキペディア日本語版の記事「ヒストグラム」(当記事)の2013年1月の閲覧回数

問題2-1 ヒストグラム

- ヒストグラムとは、片方の軸に度数、もう片方の軸に階級（ビン）をとったグラフの一種である

階級	度数
閲覧回数	その回数を記録した日数
0 - 99	1
100 - 199	2
200 - 299	4
300 - 399	5
400 - 499	4
500 - 599	7
600 - 699	7
700 - 799	1



問題2-1 四捨五入

■ 型変換（キャスト）

データを一時的に別の型に評価したいときに利用する

例：

```
int a=5, b=2; c;
```

```
double d, e=12.345;
```

```
c = 1 / b;
```

整数除算なので c は 2 になる

```
d = a / b;
```

整数除算なので d は 2.0 になる

```
d = (double)a / b;
```

a の値が double 型なので d は 2.5 になる

```
d = e;
```

d は 12.345

```
c = (int)e;
```

(int) の効果で小数部切捨てになり c は 12

```
d = (int)e;
```

(int) の効果で小数部切捨てになり d は 12.0

```
d = e - (int)e;
```

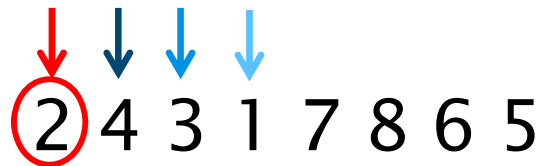
小数部取り出しで d は 0.345 になる

型変換（キャスト）を使った四捨五入を考えよう

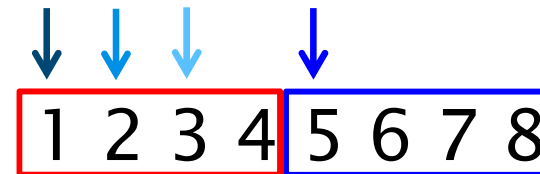
問題2-2, 2-5 比較回数の解析

- 「配列データ同士の大小関係の比較」の回数を解析

選択ソート



マージソート



要素数が増えたとき、比較回数はどれくらい増えるか？

- 配列データは、malloc 関数を使って動的に確保する
- データファイルの先頭にデータ数が格納されている
- ソートするデータは、先頭以降のデータであることに注意する

問題2-3 マージソートのアルゴリズムの説明

■ 今日説明したマージソートの動作を面接員に説明する

必ず、**計算過程の配列を記したレポート**（手書きでもよい）を
各アルゴリズムについて用意し、それを用いて説明すること

マージソート

2 4 3 1 7 8 6 5



説明

2 4 3 1 7 8 6 5



説明

●
●
●



説明

1 2 3 4 5 6 7 8

**A4レポート用紙
1枚以内でまとめること**

問題2-4 再帰呼び出し

■ 関数内で自分自身を呼び出すこと

例：階乗計算

```
int kaijyo(int n)
```

```
{  
    if (n==1)  
        return 1;  
    else  
        return n*kaijyo(n-1);  
}
```

もしnが1なら答えは1

そうでなければ答えはn×(n-1)の階乗

漸化式を解くために利用できる

階乗の計算は、以下のような漸化式で表現できる

$$\underline{f(n)} = \begin{cases} 1 & n = 1 \\ n \cdot \underline{f(n-1)} & otherwise \end{cases}$$

問題2-6, 2-7 比較回数の理論値の計算

- 選択ソート

- ◆ 一般式（要素の個数 n で表す）

- マージソート

- ◆ 最大値・最小値

- ◆ 漸化式

- 理論値と実験値の比較

問題2-8 行列データのソート【発展問題】

- malloc 関数を用いて、データ格納に必要なメモリ領域（2次元配列）を動的に確保する
- 行列データをソートするためのmatrixsort 関数を作成する
- データファイルの先頭行は、「行番号あるいは列番号 i , 列(1)あるいは行(2), 昇順(1)あるいは降順(2)」である
つまり、**i行（あるいはi列）が昇順（あるいは降順）になるように行列全体を並べ替えることを意味する**
- 選択ソートとマージソートのどちらを使用してもよい

問題2-8 行列データのソート【発展問題】

- データファイルの先頭行は、「行番号あるいは列番号 i , 列(1)あるいは行(2), 昇順(1)あるいは降順(2)」である
つまり, i 行 (あるいは i 列) が昇順 (あるいは降順) になるように行列全体を並べ替えることを意味する

	1	2	3	4
1	5	1	2	6
2	12	7	9	3
3	8	7	15	11
4	2	1	9	4

“2 1 2” の場合 . . .

「データの 2 行目 (1) が降順 (2) になるように列をソートする」

→ 「2列目と3列目を入れ替えることで, 2行目が降順に並ぶ」

	1	3	2	4
1	5	2	1	6
2	12	9	7	3
3	8	15	7	11
4	2	9	1	4

面接について

2 週目



4 週目



2週目でのみ採点

問題2-1, 2-2, 2-3

問題2-4, 2-5, 2-6, 2-7
発展問題 2-8

面接を受けるときの注意事項

- **C言語の用語や関数の動作をあらかじめ調べてくること**
 - ◆プログラムのソースコードに説明を書いてかまわないので、面接時に説明できるようにすること
- **課題に関する問題文をきちんと読むこと**
 - ◆問題文を読むことが解答への近道になることもあります
- **2週目の面接時に、途中だったとしても、取り組んでいる問題を持ってきましたよう**
 - ◆困っていることやわからないことがあれば、面接の時にアドバイスをするので、途中のプログラムでも遠慮なく持ってきましたよう