

平成31年度 知能情報工学実験演習2

# 離散構造モデリング

## 第2回演習資料

坂本比呂志 平田耕一 芳野拓也

情報工学部 知能情報工学科

# 前回のプログラムの問題点: RUNの長さに強く依存する

run-length圧縮:

同一文字の連続(run)をその長さ(length)で置き換える圧縮法

AAAAAAAAA → A9

AAAAAAAAABBBBB → A9B5

AAAAAAAAABBBBBCCCCC → A9B5C6

入力にrunが含まれないときは圧縮できない(それどころかデータサイズが増加してしまう)

ABCDABCD → A1B1C1D1A1B1C1D1

汎用的なrun-length圧縮: **長いrunがないデータでもうまく圧縮したい**

## 4. 文字列のソート

# 整数のソート

(例)バブルソート

- ・array[N]に格納された整数をソートする
- ・k++しながら  $\text{array}[k+1] < \text{array}[k]$  ならば両者を入れ替える
- ・このループが終了するとarray[N]に最大値が入っている
- ・以上の手続きをarray[N-1]に対して同様に繰り返す

# バブルソートのサンプルプログラム

```
void BubbleSort(int Data[], int n){
    int i, j, tmp;
    for(i = 0; i < n - 1; i++){
        for(j = n - 1; j > i; j--){
            if(Data[j-1] > Data[j]){
                tmp = Data[j-1];
                Data[j-1] = Data[j];
                Data[j] = tmp;
            }
        }
    }
}
```

# 文字列のソート

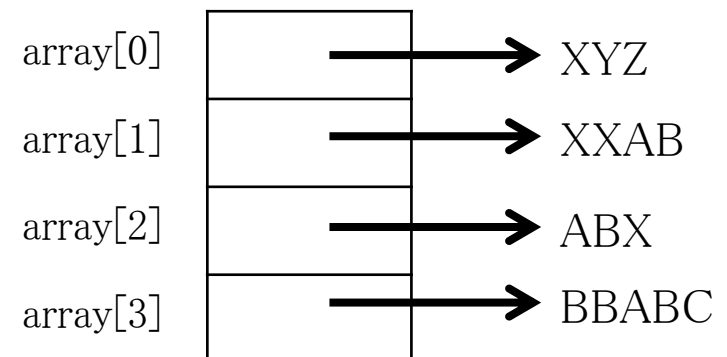
文字列のソート = 整数型ポインタ配列のソート

- ・整数配列 `array[ ]` の各要素 `array[k]` が `k` 番目の文字列へのポインタを表していると考える
- ・整数のソートでは `array[k]` と `array[k+1]` の大小比較を行ったのに対して文字列のソートでは

`array[k]` が指す文字列 `s1` と `array[k+1]` が指す `s2` を比較する

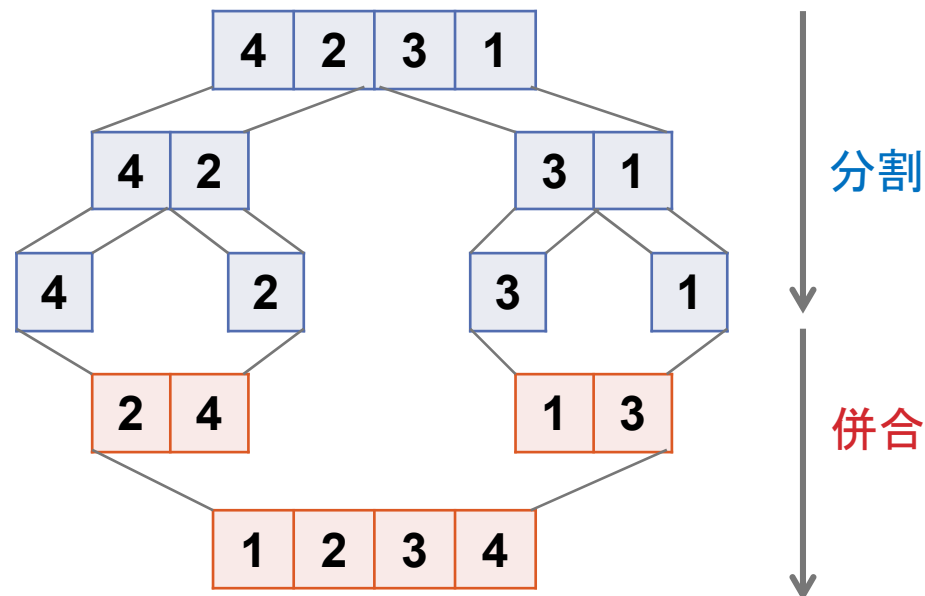
- ・文字列の大小は長さではなくアルファベットで比較 `ABBB < XY`
- ・文字列の大小比較は `strcmp(s1, s2)`

$$\left\{ \begin{array}{l} s1 < s2 \text{ のとき } \text{strcmp}(s1, s2) < 0 \\ s1 = s2 \text{ のとき } \text{strcmp}(s1, s2) = 0 \\ s1 > s2 \text{ のとき } \text{strcmp}(s1, s2) > 0 \end{array} \right.$$



# 高速なソートアルゴリズム マージソート

- ・整数配列を2分割する:
- ・arrayが分割できない(要素がひとつ)ならば何もせずリターン
- ・要素が2つならその大小を比較して必要なら入れ替える
- ・要素が3つ以上ならそれをさらに二つに分割して再帰的に以上を繰り返す



# マージソートのサンプルプログラム

```
void Msort(int Data[ ], int temp[ ], int left, int right)
```

```
{
```

```
    int mid, i, j, k;
```

```
    if(left >= right)
```

```
    return;
```

```
    mid = (left + right)/2;
```

```
    Msort(Data,temp,left,mid);
```

```
    Msort(Data,temp,mid+1,right);
```

```
    for(l = left; l <= right; l++)
```

```
        temp[l] = Data[l];
```

```
    i = left;
```

```
    j = mid + 1;
```

```
    k = left;
```

```
    while(i <= mid || j <= right){
```

```
        if(j > right || (i <= mid && temp[i] <= temp[j])){
```

```
            Data[k] = temp[i];
```

```
            i++;
```

```
        }else{
```

```
            Data[k] = temp[j];
```

```
            j++;
```

```
        }
```

```
        k++;
```

```
    }
```

```
}
```



# マージソートのサンプルプログラム VER.2

```
void Msort(int Data[ ], int temp[ ],
           int left, int right)
{
    int mid, i, j, k;

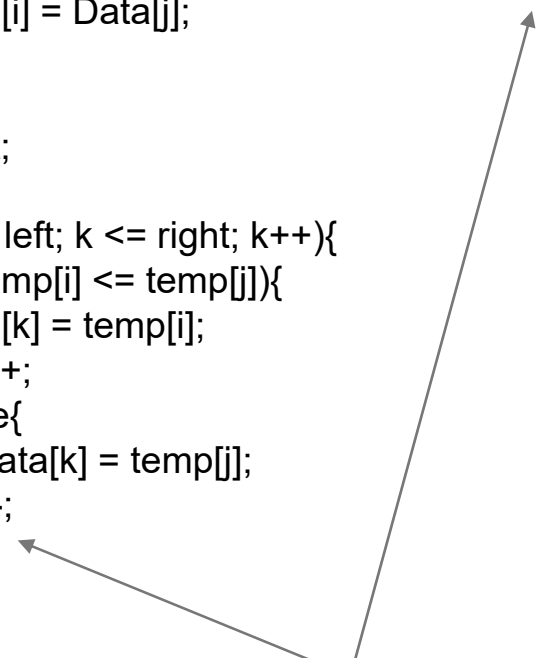
    if (left >= right)
        return;

    mid = (left + right) / 2;
    Msort(Data, temp, left, mid);
    Msort(Data, temp, mid + 1, right);
```

```
    for (i = left; i <= mid; i++)
        temp[i] = Data[i];
    for (i = mid + 1, j = right; i <= right; i++, j--)
        temp[j] = Data[j];

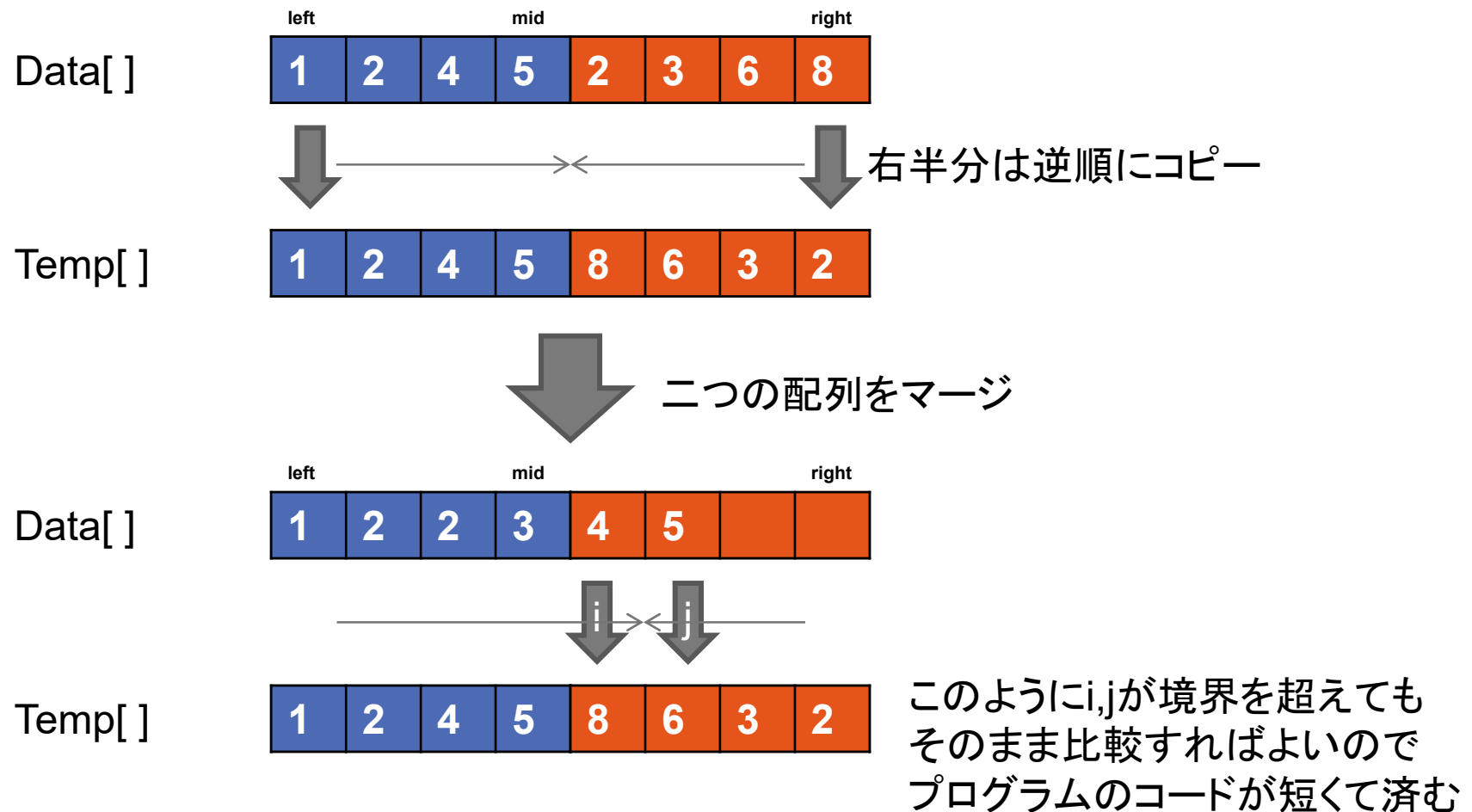
    i = left;
    j = right;

    for (k = left; k <= right; k++){
        if (temp[i] <= temp[j]){
            Data[k] = temp[i];
            i++;
        }else{
            Data[k] = temp[j];
            j--;
        }
    }
}
```



このjが減少していることに注意

## jが減少していることの意味



## 課題3

入力ファイル(Test3.txt)の文字列をソートして出力ファイルに書きだすプログラムを作成せよ. ソートアルゴリズムはバブルソートでよい.

なお、mysortの引数は適当に変更してもよい.

# 課題3の提出方法

提出するもの

- Makefile
- func.h
- main.c
- check.c
- mysort.c
- Test3.txt (オリジナルデータ)
- Test3\_sorted.txt (Test3.txtをソートしたファイル)

これらをフォルダ“kadai3\_学籍番号”にまとめて、ZIPで圧縮

(アドレス) [experiment@donald.ai.kyutech.ac.jp](mailto:experiment@donald.ai.kyutech.ac.jp)

(件名) kadai3\_学籍番号

## 課題4

課題3の内容をマージソートによって実現しTest4.txtをソートせよ.

※課題3の内容を保存して作成せよ.

※課題4を提出すれば, 課題3を提出する必要はないものとする.

# 課題4の提出方法

提出するもの

- Makefile
- func.h
- main.c
- check.c
- mysort.c
- Test4.txt (オリジナルデータ)
- Test4\_sorted.txt (Test4.txtをソートしたファイル)

これらをフォルダ“kadai4\_学籍番号”にまとめて、ZIPで圧縮

(アドレス) [experiment@donald.ai.kyutech.ac.jp](mailto:experiment@donald.ai.kyutech.ac.jp)

(件名) kadai4\_学籍番号