ななちゃんのIT教室

教養講座：データ構造　の巻

by nara.yasuhiro@gmail.com

ななちゃんが、データ構造の
基礎（リスト、二分木）を学ぶという　お話

第 0.1 版　2017 年 6 月 21 日

フリー素材
http://freeillustration.net

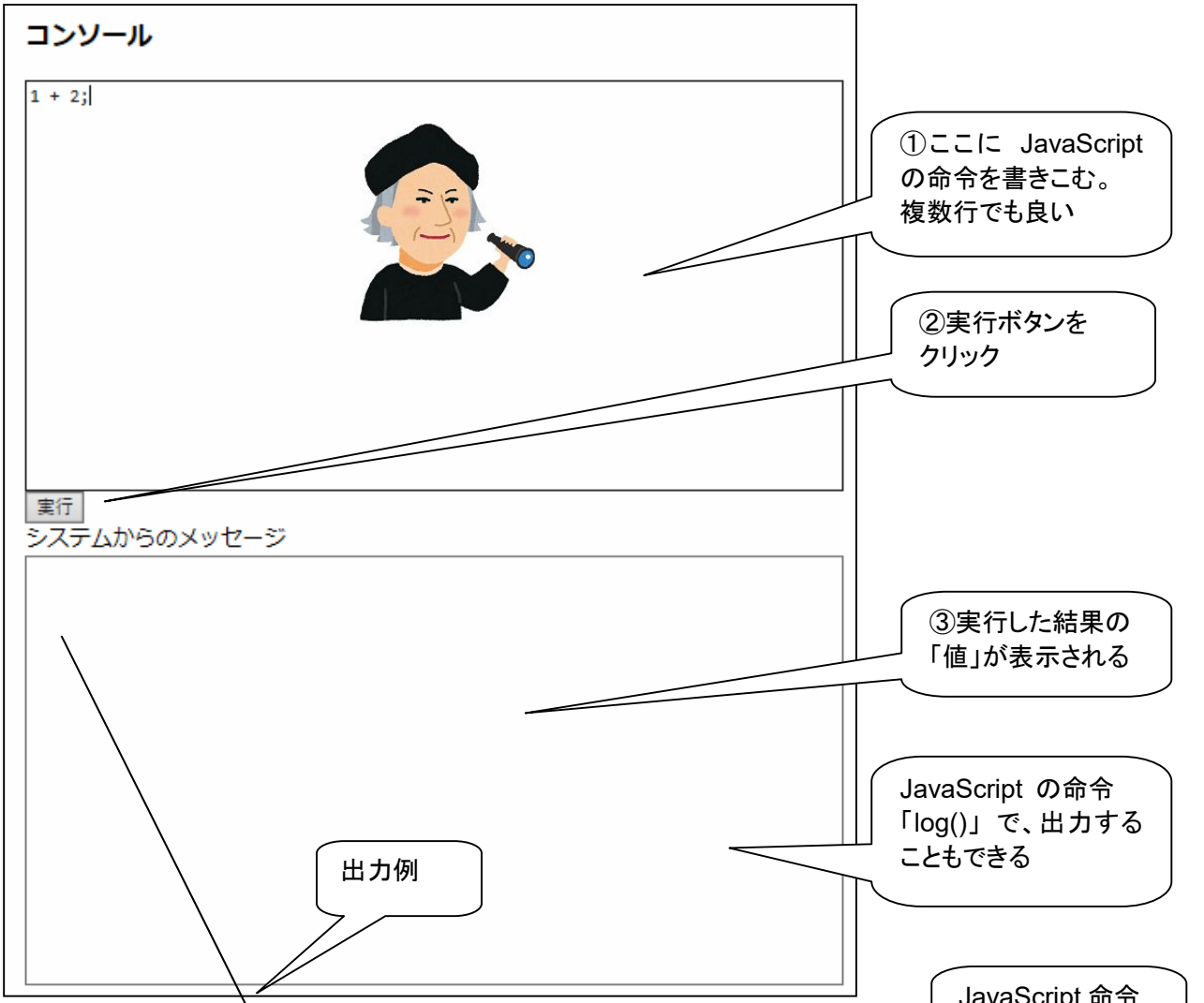
いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　秘密道具：マイ・コンソール

なな：クリじい、「データ構造」の勉強をするんだけど、便利な秘密道具はない？

クリ：あるぞ、あるぞ。定番秘密道具の「マイ・コンソール」。他の巻を読んでない読者のために、説明しよう。




```
<= 1 + 2;
=> Number number 3
<= "1" + "2"
=> String string "12"
<= 1;2;
=> Number number 2
<= var x = 1;
=> Undefined undefined undefined
<= x
=> Number number 1
<= var x; x= 1;
=> Number number 1
```

JavaScript 命令「1+2」を入力した

```
<= 1 + 2;
=> Number number 3
```

実行結果の「値」は 3

注意：var x = 1; の値は「undefined」

実行結果の「型」は Number

本　教材ではこのように圧縮表示しています

```
1 + 2;          // Number number 3
"1" + "2"       // String string "12"
1;2;            // Number number 2
var x = 1;      // Undefined undefined undefined
x               // Number number 1
var x; x= 1;    // Number number 1
```

「型」の判定方法は 2 種類 r

JavaScript 命令

実行結果の「型」と「値」

「○；○」のように、複数の JavaScript 命令がある場合、一番右の命令の型、値だけ表示される

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>コンソール</title>
 </head>
 <body>
    <h3>コンソール</h3>
    <textarea rows="19" cols="80" id=pg autofocus>1 + 2;</textarea>
    <br><input type=button onClick=go() value="実行">
    <br>システムからのメッセージ
    <br><textarea rows="20" cols="80" id=log></textarea>
    <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp    = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }
function log(s) { logd += s; }
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }
function isPrimitive(x) {
    return (typeof x)!="object";
}
function toLiteral(x) {
    if (typeIs(x)=="Number" && isNaN(x)) return "NaN";
    if (x === Infinity)                  return "Infinity";
    if ((typeIs(x)!="Symbol")&&(-x === Infinity)) return "-Infinity";
    if (typeIs(x)=="Set") return "Set("+JSON.stringify([...x])+")";
    if (typeIs(x)=="Map") return "Map("+JSON.stringify([...x])+")";
    return JSON.stringify(x);
}
function type(x) { return "" + (typeof x); }
function isInteger(n) { return n%1 === 0; }
function keys(obj) { return Object.keys(obj); }
function go() {
 logd = "";
 try {
            var v = geval(pgp.value);
            clog("<= " + pgp.value + "\n=> "
               + typeIs(v) + " " + type(v) + " " + toLiteral(v) + "\n");
            pgp.value = "";
            logp.scrollTop = logp.scrollHeight;
            pgp.focus();
 }
 catch(e) { clog("<= " + pgp.value + "\n=>! " + e + "\n");
                pgp.value = "";
                logp.scrollTop = logp.scrollHeight;
                pgp.focus();
 }
 if (logd != "") clog(logd + "\n");
}
   </script>
 </body>
</html>
```

第2回　データ構造とは

なな：データ構造って何？

先生：JavaScript の基本機能でいえば、文字列とか、配列とか、オブジェクトのようなもの。たとえば、文字列の場合、一文字ずつ分解して、配列データにしたり、オブジェクトにすることもできるわけね。どういう処理を、どういうアルゴリズムで実現するのかと、データ構造としてどれを選ぶかということが密接に関係しているの。たとえば、 各文字を書き換えることが必要なら、文字列より配列が良いとか、挿入や削除が多かったらオブジェクトのほうが良いかも知れません。プログラムのトータル性能は、 アルゴリズムとデータ構造の組み合わせで決まってくるということです。

なな：文字列とか、配列とか、オブジェクトのどれを使うか選択するということ？

先生：JavaScript 組み込みのデータ構造を使う以外にも、オブジェクトを使って、独自のデータ構造を作ることもできます。

なな：たとえば、どういうの？

先生：今回の巻では、リスト構造、二分木、ヒープ、それから、リスト構造を使ったスタックとキューを紹介しましょう。スタックとキューは、配列を使っても実現できるので、そういう意味では、上位のデータ構造というか、より抽象度の高いデータ構造と言えるかも知れません。組み合わせるアルゴリズムは、ソーティングや検索をとりあげます。

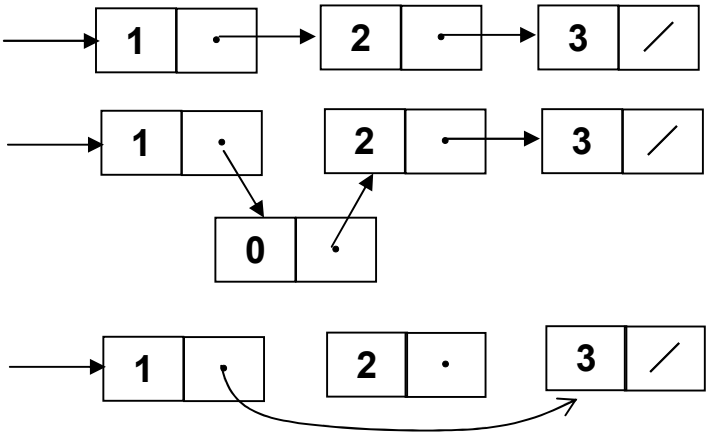
なな：どれが良いというか、高級なの？

先生：アルゴリズムとの組み合わせによります。挿入削除が頻繁に必要なアルゴリズムなのかとか、データは順番にアクセスすることが多いのか、インデックスを使って、とびとびにアクセスする必要があるかとか。

第3回　リスト構造

なな: リスト構造って何？

先生: データをバラバラにして、順にポインタで連結したデータ構造です。連結リスト（Linked list)とも呼ばれます。ポインタを書き換えるだけで、データの挿入や削除が行えるのが特徴です。配列だと、データを挿入、削除した場合、それより後ろのデータをすべて移動する必要があるので、時間がかかりますが、リスト構造では移動が不要です。リスト構造では、データアクセスには、先頭から順に探索してゆきます。配列のように、いきなり 50 番目のデータにアクセス、というようなことはできません。




```
function LNode(first, second) {
    this.f = first;
    this.s = second;
}
LNode.prototype.disp = function d() {
    if (this.f === null) return "";
    if (this.s === null) return this.f + "";
    return this.f + "," + d.call(this.s);
};
LNode.prototype.dp = function d() {
    return "<" + this.disp() + ">";
};
LNode.prototype.insert = function (first) {
    if (this.f === null) { this.f = first; this.s = null; return this; }
    var temp = this.s;
    this.s = new LNode(this.f,this.s);
    this.f = first;
    return this;
};
LNode.prototype.advance = function () {
    if (this.s === null) return new LNode(null,null);
    return this.s;
};
LNode.prototype.delete = function () {
    if (this.s === null) { this.f = null; return this; }
    this.f = this.s.f;
    this.s = this.s.s;
    return this;
};
LNode.prototype.append = function (first) {
    if (this.f === null) { this.f = first; this.s = null; return this; }
    this.s = new LNode(first,this.s);
    return this.s;
};
```

1 つのデータ（ノード）のコンストラクタ。
f (first) がデータ部分、s (second) がポインタ部分。

リスト構造全体を表示するメソッド

データを一つ、注目ノードの前に追加するメソッド

注目ノードを一つ先に移動するメソッド

データを一つ削除するメソッド

データを一つ、注目ノードの後に追加するメソッド

注目点を、追加ノードに移動

第4回 検索、ソーティング

先生: リストを利用したソーティングプログラムです。先頭からデータを見てゆき、新データより大きいデータを発見したら、その手前に挿入します。空のリストからはじめ、ひとつずつデータを挿入してゆくことで、昇順にならんだリストが構築されます。

```
LNode.prototype.step = function self(d) {
    var save = this;
    if (this.f === null)          this.f = d;
    else if (this.f > d)          this.insert(d);
    else if (this.s === null) this.append(d);
    else                          self.call(this.advance(),d);
    return save;
};
var node = new LNode(null,null); node.dp();     // String string "<>"
node.step(1).step(2).step(5).step(3).dp();      // String string "<1,2,3,5>"

[3,8,6,5,4,9,2,7,1].reduce(function(prev,cur) { return prev.step(cur); },
                           new LNode(null,null)).dp();
                                                // String string "<1,2,3,4,5,6,7,8,9>"
```

場所をみつけて新データを挿入

末尾到達→新データを追加

再帰呼び出し

先生: すでに作られている、昇順にデータが並んでいるリストデータの中から、特定のデータを検索し、みつかったら削除するプログラムです。

```
var node = new LNode(null,null);
node.append(1).append(2).append(3).append(4).append(5);
node.dp();                  // String string "<1,2,3,4,5>"

function remove(d) {
    var n = node;
    while (true) {
        if (n.f == d)          n.delete();
        if (n.s === null)    break;
        n = n.advance();
    }
}

remove(3);
node.disp();     // String string "<1,2,4,5>"
```

データがみつかったら削除

動作確認。<1,2,3,4,5>から3 を削除

第5回　スタックとキュー

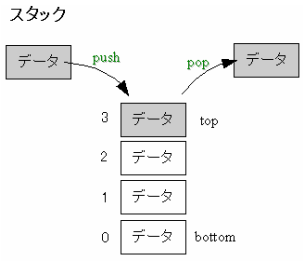
先生: リストを使って、スタックと、キューを作ってみました。まず、スタック。 FILO (First In Last Out) ともいいます。

なな: リストを使って、スタックが作れるの？

先生: プッシュは insert、ポップは、advance や、delete で実現できます。それぞれの例です。 A、B、C の順にプッシュしてからポップすると、C、B、A の順にデータが得られます。たとえば、<B,A>の状態のスタックで、stack.f で、先頭のデータを取りだすことができます。

```
var stack = new LNode(null,null);
stack.insert("A").insert("B").insert("C").dp();     // String string "<C,B,A>"
(stack = stack.advance()).dp();                     // String string "<B,A>"
(stack = stack.advance()).dp();                     // String string "<A>"
(stack = stack.advance()).dp();                     // String string "<>"

var stack = new LNode(null,null);
stack.insert("A").insert("B").insert("C").dp();     // String string "<C,B,A>"
stack.delete().dp();                                // String string "<B,A>"
stack.delete().dp();                                // String string "<A>"
stack.delete().dp();                                // String string "<>"
```



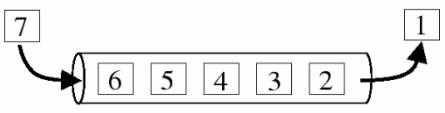
先生: これは、キューの例です。キューは、FIFO (First In First Out) とも呼ばれます。配列を使うと、大量データの移動が必要になってしまいます。リストを使えば、データ移動は不要です。この例では、A、B、C の順に入力し、A、B、C の順に取り出しています。

```
var fifo = new LNode(null,null);
fifo.append("A").append("B").append("C");
fifo.dp();                  // String string "<A,B,C>"
fifo.delete().dp();         // String string "<B,C>"
fifo.delete().dp();         // String string "<C>"
fifo.delete().dp();         // String string "<>"
```

なな: 順番が変わらないんだったら、何に使うの？

先生: データが不規則なタイミングで発生する場合、それを等間隔に取り出せたり、コンピュータが処理できしだい、ただちに次のデータを取り出せるよう、データを先行的にキューにためておいたりできます。また、コンピュータで処理したデータをキューにためておき、あとでゆっくりディスクに書き込むなどという用途に使えます。

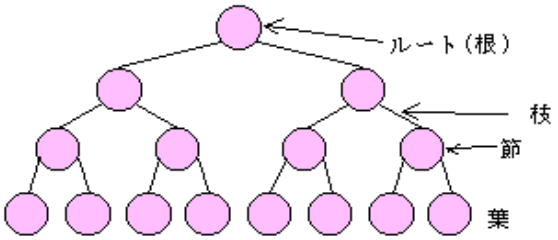
なな: バッファとか、キャッシュとかいうものね。



第6回 二分木

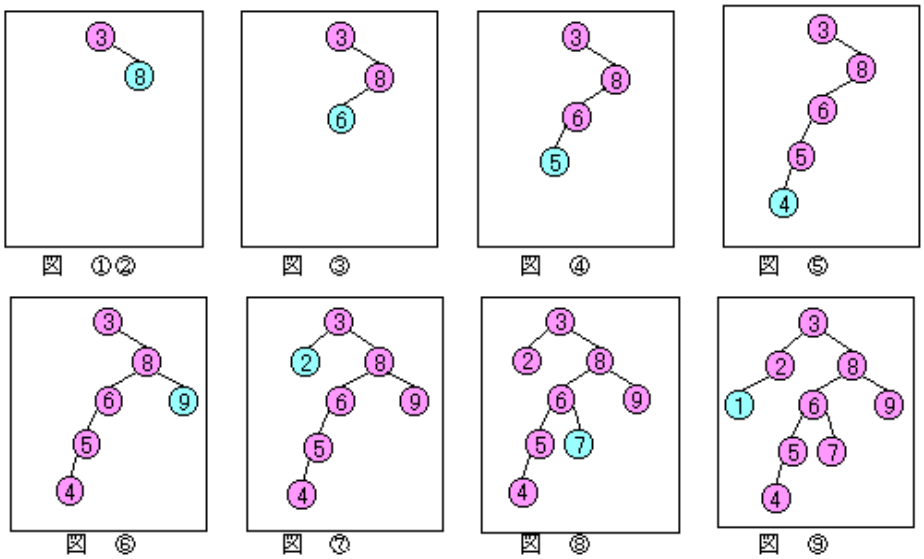
なな: 二分木って何？

先生: 下図のような形のデータ構造を二分木(binary tree)といいます。それぞれの節(○)(ノード)は、最大2つの枝を持ち、その枝先には、さらに節を持ちます。枝を持たない節を「葉」ともいいいます。

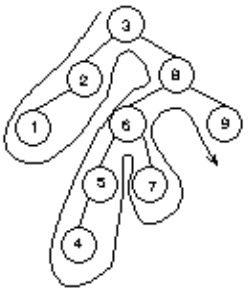


例:データ 3,8,6,5,4,9,2,7,1 を順に二分木に格納する。

1. まず、3 をルートに格納する(図①)。
2. 8 は 3 より大きいので、その右側の節に格納する(図②)。
3. 6 は 3 より大きいので右へ、次の 8 より小さいので、その左側に格納する(図③)。
4. 5 は 3 より大きいので右へ、次の 8 より小さいので左へ、次の 6 より小さいのでその左に格納(図④)。
5. 4 は 3 より大きいので右へ、次の 8 より小さいので左へ、次の 6 より小さいので左へ、次の 5 より小さいので、その左に格納(図⑤)。
6. 9 は 3 より大きいので右へ、次の 8 より大きいので、その右側に格納する(図⑥)。
7. 2 は 3 より小さいので、その左側に格納する(図⑦)。
8. 7 は 3 より大きいので右へ、次の 8 より小さいので左へ、次の 6 より大きいので、その右に格納(図⑧)。
9. 1 は 3 より小さいので左へ、次の 2 より小さいので、その左側に格納する(図⑨)。



上の例のように格納したデータ 3,8,6,5,4,9,2,7,1 を小さい順(昇順)に整列します。それぞれの節に枝があるときは、左側の方が小さいので、葉に到達するまで、左の枝をたどってゆきます。例の場合、まず 1 が得られます。①の節には枝がないので②に戻ります。②の節には右の枝がないのでさらに戻ります。次に③の節の右の枝をたどってゆきます。このように、ルートから左の枝をたどって下へ降りてゆきます。葉まで到達したら上へ戻ってゆきます。戻ったときに右の枝があればその枝をたどって下に降ります。これを繰り返しデータを取得することで昇順に整列することができます。



リスト構造と大きくことなる点は、データを先頭から連続して見るのでなく、とびとびに見られることです。

```
function btree(key,left,right) {
    this.k = key;
    this.l = left;
    this.r = right;
}

btree.prototype.disp = function self() {
    if (this.k === null) return "";
    return self.call(this.l) + " " + this.k + " " + self.call(this.r);
}

btree.prototype.addBT = function self(n) {
    if (this.k === null) {
        this.k = n;
        this.l = new btree(null,null,null);
        this.r = new btree(null,null,null);
    }
    else if (this.k < n) this.r = self.call(this.r,n);
    else                 this.l = self.call(this.l,n);
    return this;
}

(new btree(null,null,null)).addBT(3).addBT(8).addBT(6).addBT(5).addBT(4)
                           .addBT(9).addBT(2).addBT(7).addBT(1).disp();
            // String string " 1   2   3   4   5   6   7   8   9 "

[3,8,6,5,4,9,2,7,1].reduce(function(prev,cur) { return prev.addBT(cur); },
                           new btree(null,null,null)).disp();
            // String string " 1   2   3   4   5   6   7   8   9 "
```

二分木ノードの
コンストラクタ

二分木の整列表示

新しいデータを
二分木に格納する

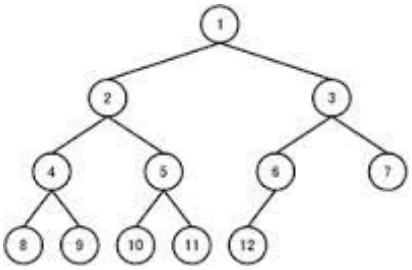
ECMAScript　2015
の reduce を使う場合

第7回　ヒープソート

なな：ヒープって何？

先生：ヒープ（heap）は、「累積」、「積み重なったもの」という意味です。データ構造としては、親の値が子供の値以上であるような完全2分木です。2分探索木と異なり、左のノードが右のノードより小さいとは限りません。ヒープの各ノードに、上→下、左→右に、A1、A2、、、と、番号をつけると、

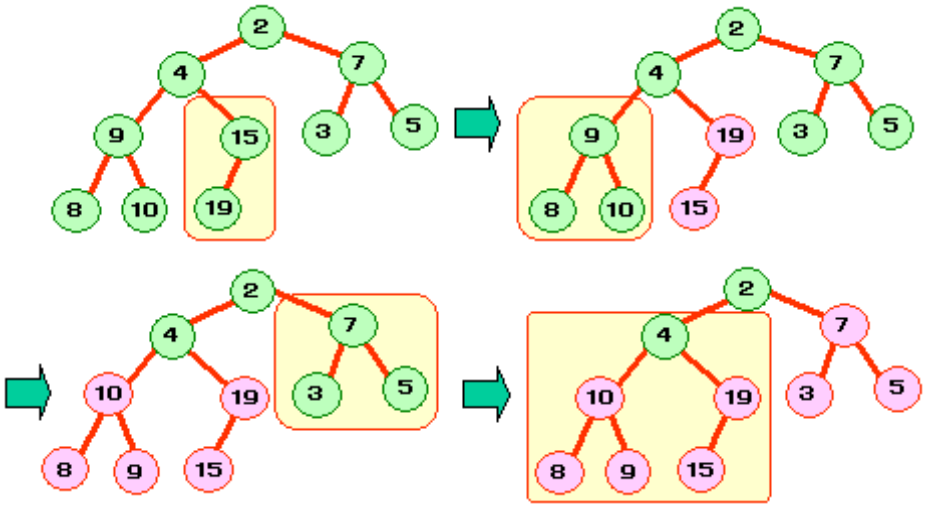
Ai の親は、Ai/2　　i/2 は、整数割り算、切捨て
Ai の左の子は，Ai*2
Ai の右の子は，Ai*2+1



という関係になります。これを、番号順に配列要素に対応させることができます。

末尾から先頭に向かって、親と子のデータを順に比較。
子のほうが大きかったら、親とデータを交換。
上端のデータが最大になるので、それを取り出す。
末尾のデータを上端に移し、末尾を削除。

以上の処理を、データがひとつになるまで繰り返します。



```
var x = [1,12,3,4,15,6,10,8,9,7,11,2,13,14,5];
var i, j, tmp, s = ""

for (i =   x.length; i >= 1; i--) {
   for (j = Math.floor(i/2); j >= 1; j--) {
          if (x[j-1] < x[j*2-1])                    { tmp = x[j-1]; x[j-1] = x[j*2-1]; x[j*2-1] = tmp; }
          if (((j*2+1) <= i) && (x[j-1] < x[j*2])) { tmp = x[j-1]; x[j-1] = x[j*2];   x[j*2] = tmp; }
   }
   s += x[0] + " "; x[0] = x[i-1];
}
s;                  // String string "15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 "
```