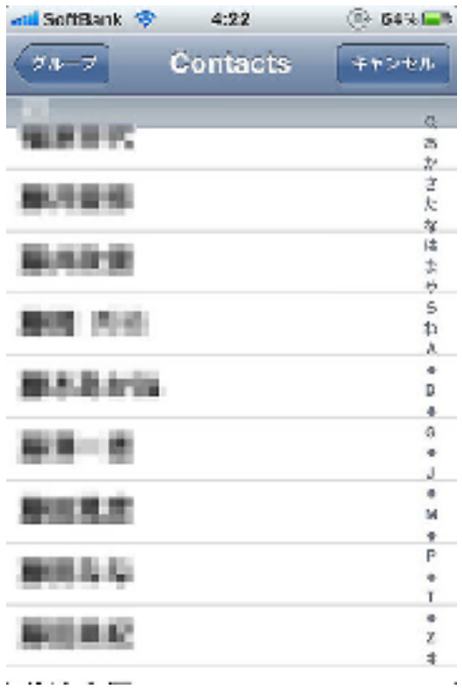
データ構造とアルゴリズム

第６回　整列

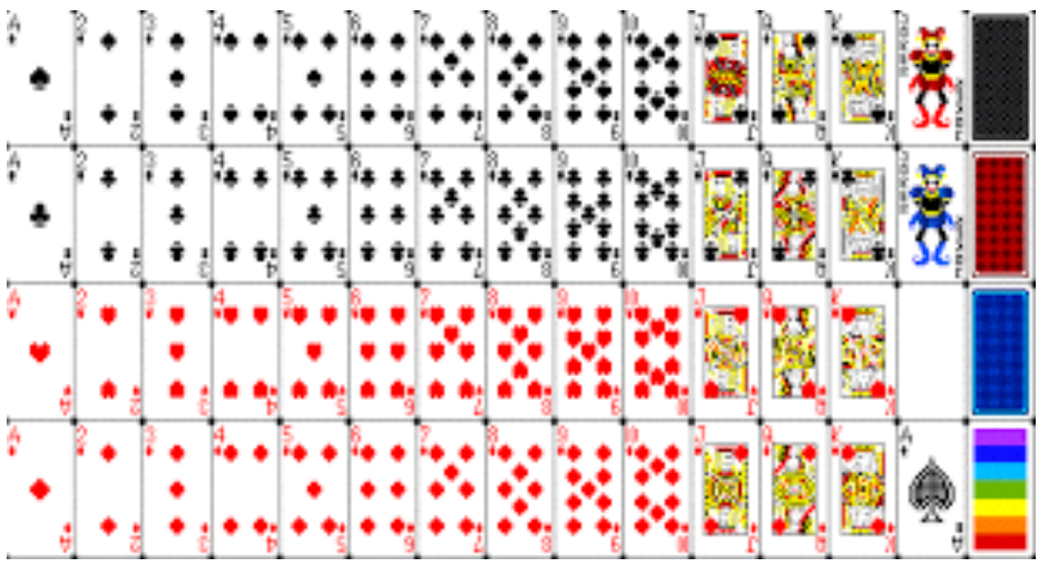
小池 英樹 (*koike@c.titech.ac.jp*)

整列(SORTING)

- アドレス帳
- 成績表
- 辞書
- ゲーム



名前	国語	算数	理科	社会	英語
織田信長	75	59	30	65	25
徳川家康	77	87	91	64	65
今川義元	65	56	46	56	43
石田光成	52	46	82	57	57
豊臣秀吉	76	90	56	75	61
伊達政宗	57	17	67	91	75



整列(SORTING)

- **内部整列 (internal sorting)**
  - すべてを主記憶上で実行する場合

- 外部整列 (external sorting)
  - 外部の補助記憶を用いる場合

各種ソートアルゴリズム

- バブルソート (bubble sort)
- 挿入ソート　(insertion sort)
- 選択ソート (selection sort)

最悪計算時間

$O(n^2)$

- シェルソート (Shell's sort)

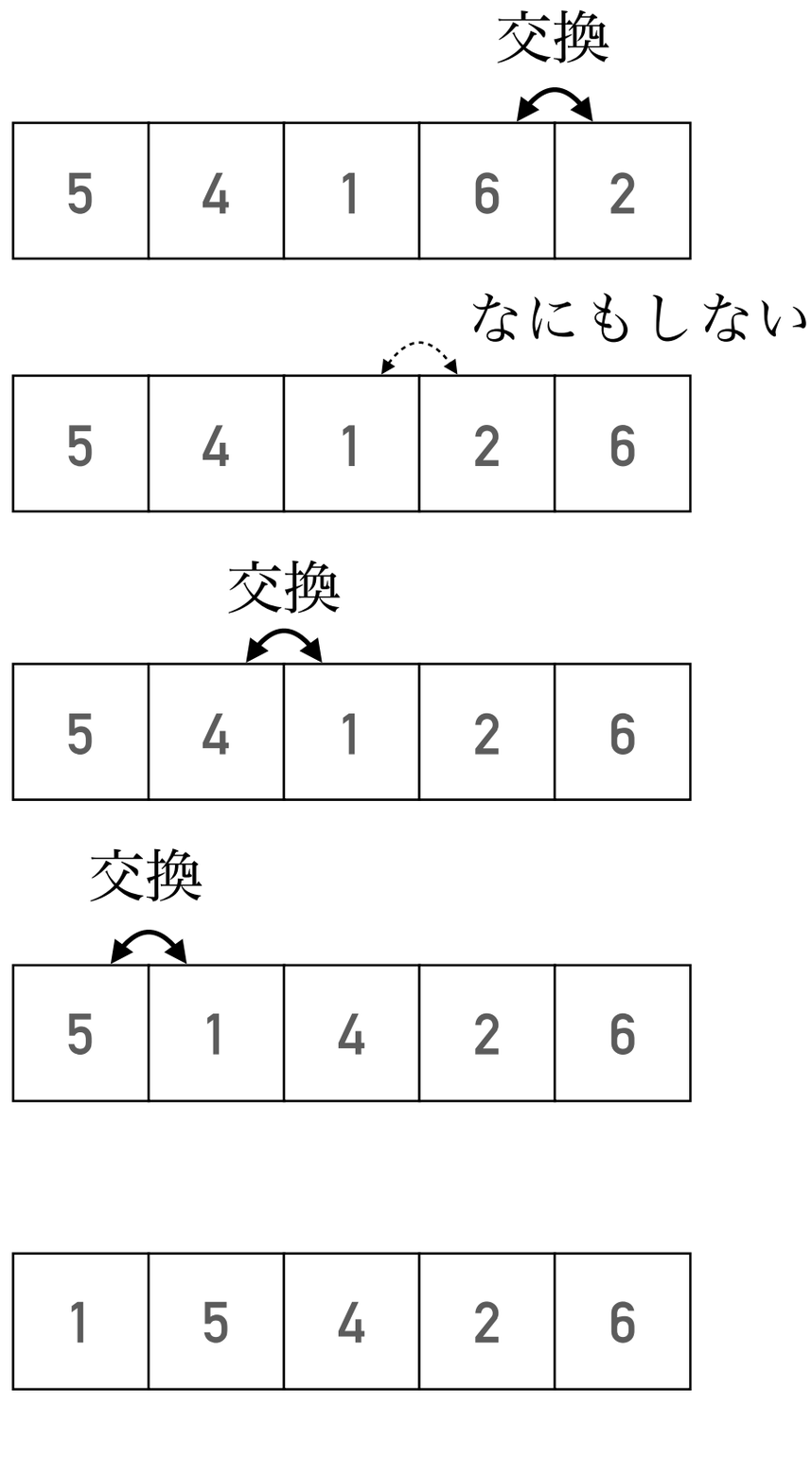
$O(n \log^2 n)$

- 2分木ソート (binary tree sort)
- ヒープソート (heap sort)
- マージソート (merge sort)
- **クイックソート (quick sort)**

$O(n \log n)$

バブルソート (BUBBLE SORT)

- 考え方
  - 隣り合う２つを比べて
    「軽い（小さい）」方を左に



1が泡のように浮き上がって行く

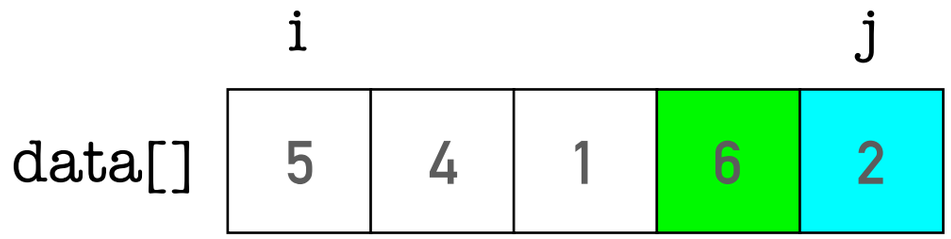
バブルソート

擬似コード

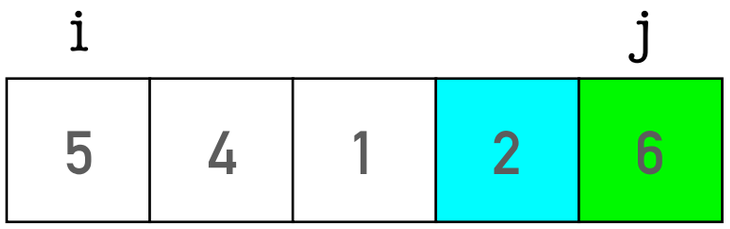
```
for (i=0; i<n-1; i++) {
    for (j=n; j>i; j--) {
        if (data[j] < data[j-1])
            data[j]とdata[j-1]を交換
    }
}
```

バブルソート

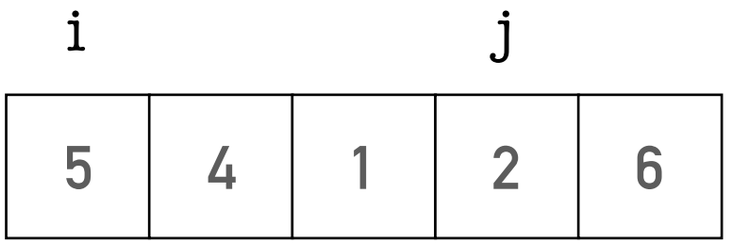
初期状態　　i=0　j=n-1



data[j-1] > data[j]なので交換



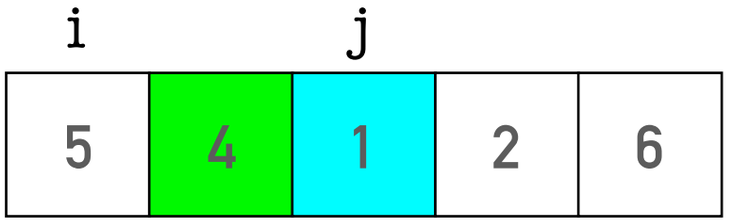
j=n-2



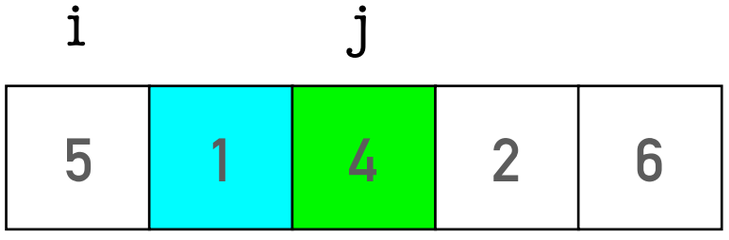
data[j-1] < data[j]なので何もしない

バブルソート

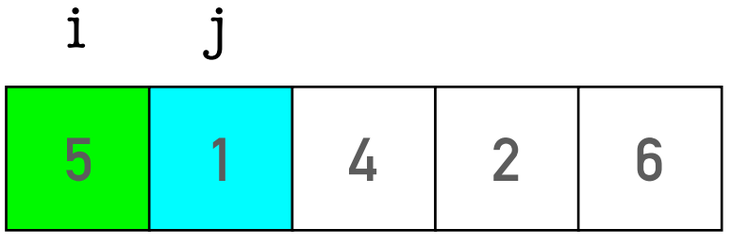
i=0　j=n-3



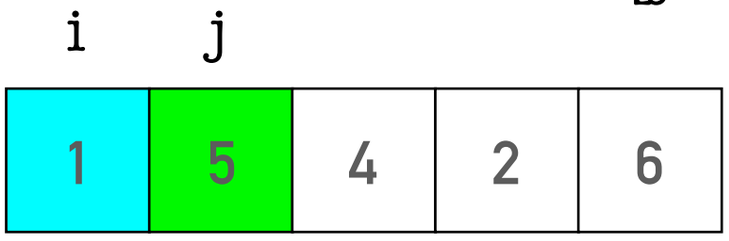
data[j-1] > data[j]なので交換



j=n-4



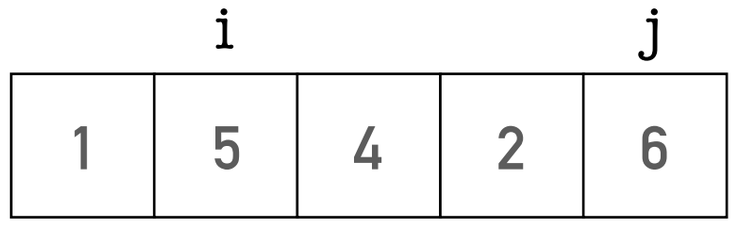
data[j-1] > data[j]なので交換



j=n-5はj=iとなるので内側のループ終了

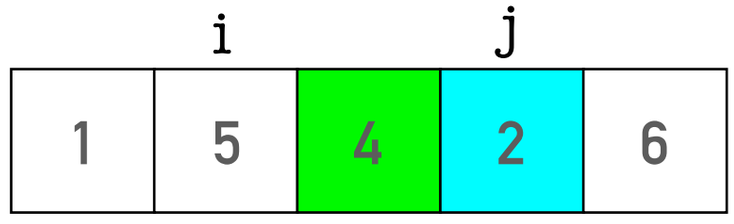
バブルソート

i=1　j=n-1



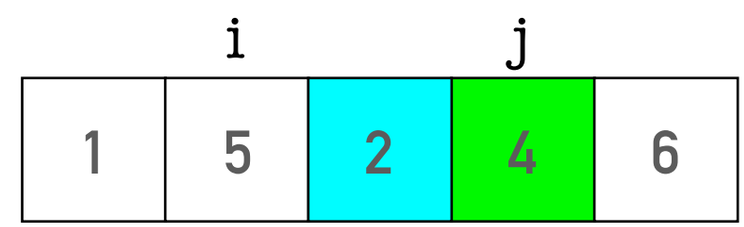
data[j-1] < data[j]なので何もしない

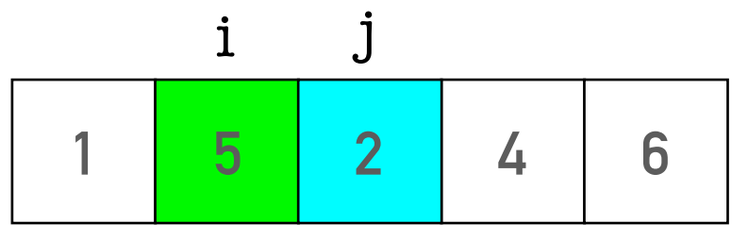
j=n-2



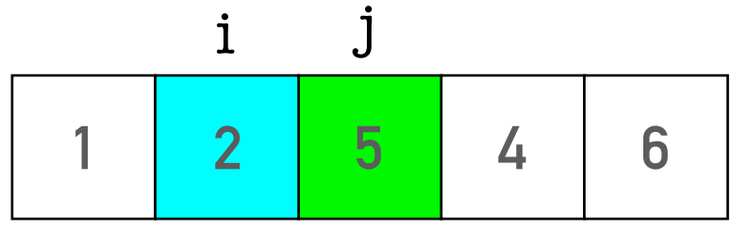
data[j-1] > data[j]なので交換

j=n-3





data[j-1] > data[j]なので交換



j=n-4はj=iとなるので内側のループ終了

バブルソート

初期状態

data[]

5	4	1	6	2

i=0の後

i

1	5	4	2	6

i=1の後

i

1	2	5	4	6

i=2の後

i

1	2	4	5	6

バブルソート

```
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;
            }
        }
    }
}
```

バブルソート

```
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;
            }
        }
    }
}
```

高々$c_2(n-i)$ステップ

$$c_3 n + \sum_{i=i}^{n-1} c_2(n-i) = 1/2\, c_2 n^2 + (c_3 - 1/2 c_2)n = O(n^2)$$

iを1つ増やしてからテストすることを考慮

https://visualgo.net/en/sorting

https://www.toptal.com/developers/sorting-algorithms

挿入ソート(INSERTION SORT)

- 考え方
  - i=1からn-1まで以下を繰り返す
    - data[i]をj≦i番目の位置で

      j≦k<iに対してはdata[i]<data[k]であり，かつ

      data[i]≧data[j-1]かj=1であるような場所に移す

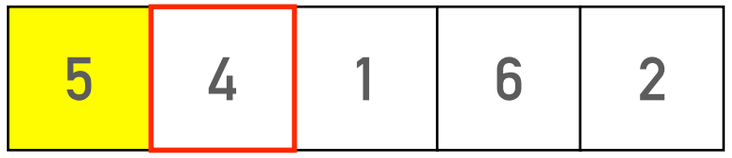
擬似コード

```
for (i=1; i<n-1; i++) {
    j = i;
    while (data[j]<data[j-1])
        data[j]とdata[j-1]を交換
        j=j-1;
}
```

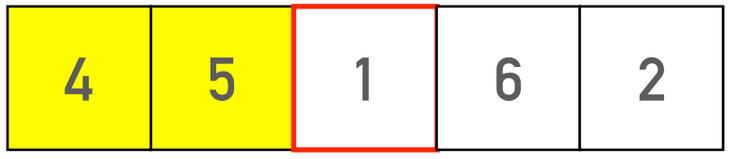
挿入ソート

初期状態

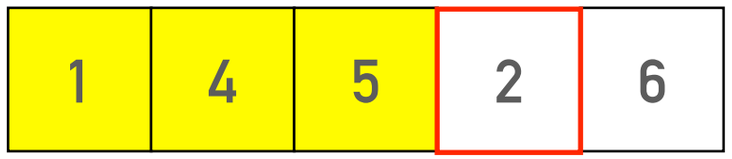
data[]



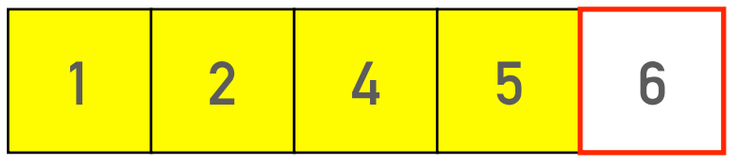
data[0] > data[1]なので交換



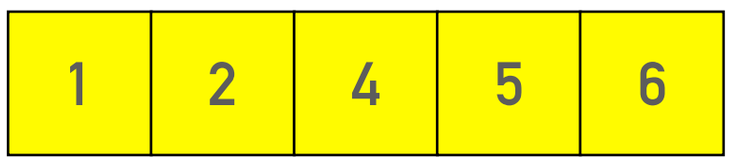
data[1] > data[2]なので
data[2]を適当な位置に挿入



data[2] > data[3]なので
data[3]を適当な位置に挿入



data[3] < data[4]なので
何もしない



ソート済

未ソート

挿入ソート(INSERTION SORT)

```
void insertionsort(int data[], int n) {
    int i, j, tmp;

    for (i=1; i<n; i++) {
        tmp = data[i];
        j=i;
        while (j>0 && data[j-1]>tmp) {
            data[j] = dat[j-1];
            j=j-1;
        }
        data[j] = tmp;
    }
}
```

挿入ソート(INSERTION SORT)

```
void insertionsort(int data[], int n) {
    int i, j, tmp;

    for (i=1; i<n; i++) {
        tmp = data[i];
        j=i;
        while (j>0 && data[j-1]>tmp) {
            data[j] = dat[j-1];
            j=j-1;
        }
        data[j] = tmp;
    }
}
```

O(i)

$$c \sum_{i=2}^{n} i$$

$O(n^2)$

選択ソート(SELECTION SORT)

- 考え方
  - i=0からn-1まで以下を繰り返す
    - data[i], …, data[n-1]の最小値を選び,
      それをdata[i]と入れ替える

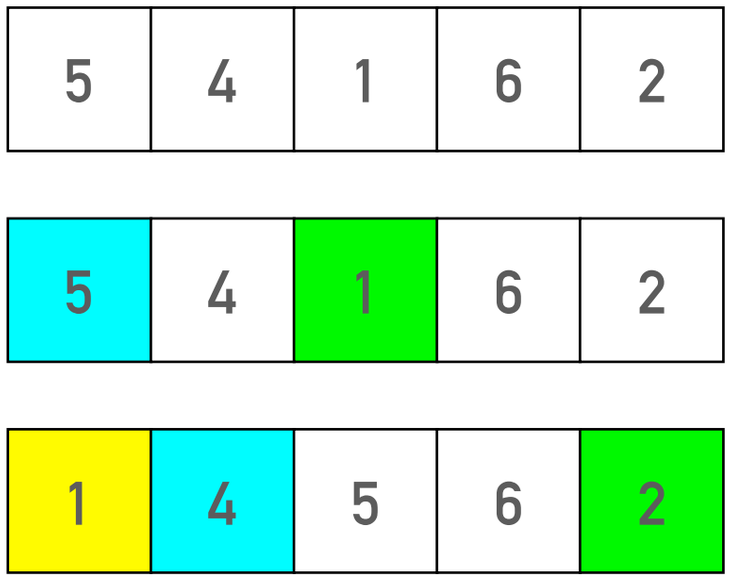
擬似コード

```
for (i=0; i<n; i++) {
    minindex = i;
    min = data[i];
    for (j=i+1; j<n; j++) {
        if (data[j] < min) {
            min = data[j];
            minindex = j;
        }
    data[i]とdata[minindex]を交換
}
```

選択ソート

初期状態

data[]



data[0]とそれ以外の最小値
data[2]=1を交換

data[1]とそれ以外の最小値
data[4]=2を交換

data[2]とそれ以外の最小値
data[4]=4を交換

data[3]とそれ以外の最小値
data[4]=5を交換

ソート済　　比較元　　最小値

選択ソート(SELECTION SORT)

```
void selectionsort(int data[], int n) {
    int i, j, minindex, min, tmp;

    for (i=0; i<n-1; i++) {
        minindex = i;
        min = data[i];
        for (j=i+1; j<n; j++) {
            if (data[j]<min) {
                min = data[j];
                minindex=j;
            }
        }
        tmp = data[i];
        data[i] = data[minindex];
        data[minindex] = tmp;
    }
}
```

選択ソート(SELECTION SORT)

```
void selectionsort(int data[], int n) {
    int i, j, minindex, min, tmp;

    for (i=0; i<n-1; i++) {
        minindex = i;
        min = data[i];
        for (j=i+1; j<n; j++) {
            if (data[j]<min) {
                min = data[j];
                minindex=j;
            }
        }
        tmp = data[i];
        data[i] = data[k];
        data[k] = tmp;
    }
}
```

$$O(n\text{-}i) \qquad c\sum_{i=1}^{n-1}(n-i) = cn(n-1)$$

$$= O(n^2)$$

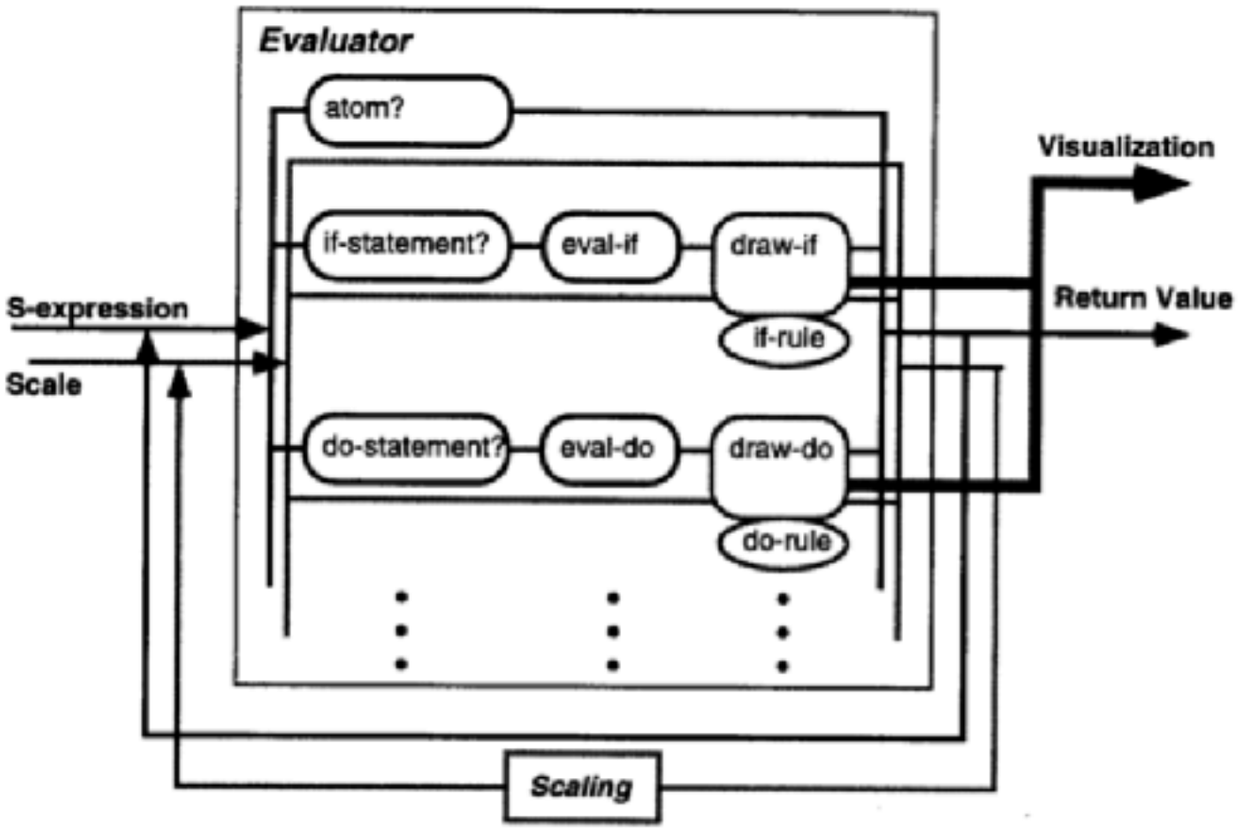
計算量

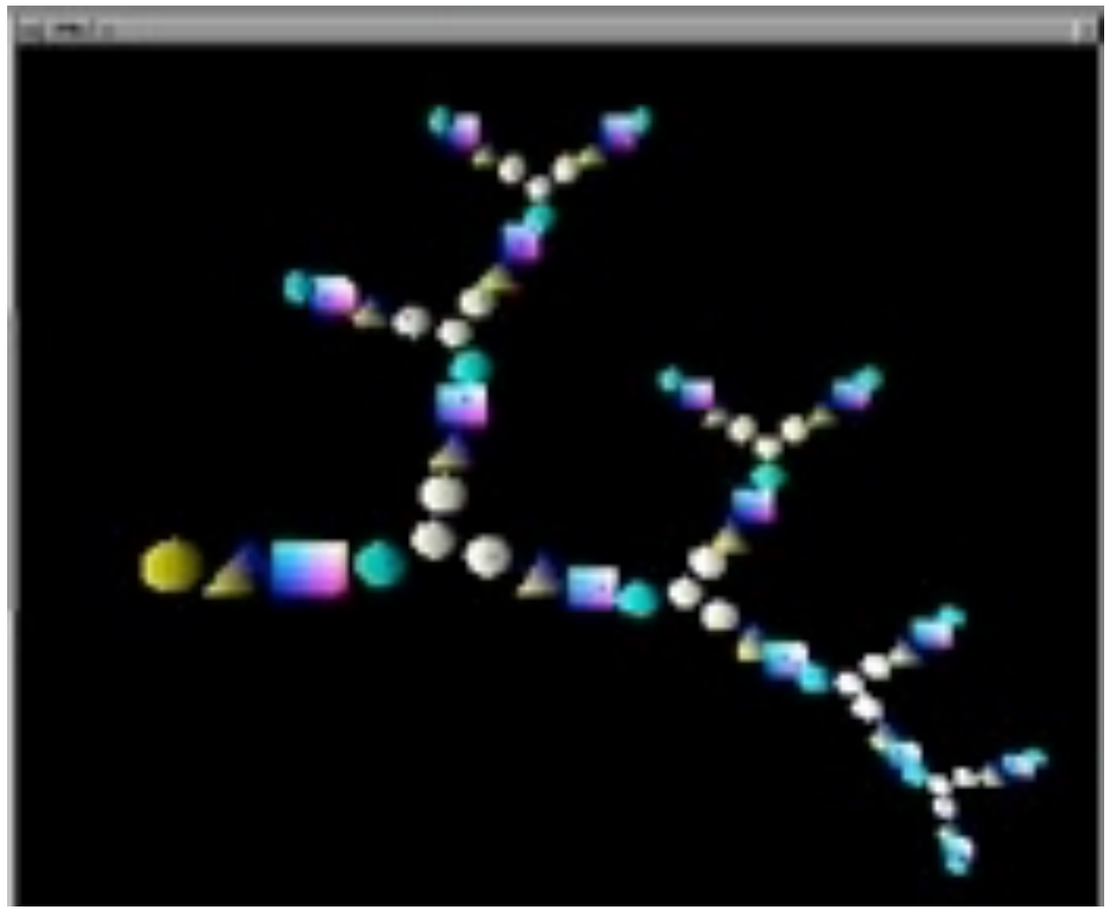
- バブルソート，挿入ソート，選択ソートともに $O(n^2)$
- n個の要素からなる入力列のいくつかに対して$\Omega(n^2)$
- レコード（データの型）が大きいと，データ交換にかかる時間が他のステープより長くなることがある．
- nが小さい時 $(n<10^3)$は，後述する複雑なアルゴリズムを作ることは時間の無駄で，わかりやすい単純なソートを使うべき．

ALGORITHM ANIMATION

- Marc H. Brown and Robert Sedgewick, A system for algorithm animation, SIGGRAPH Computer Graphics, 18, 3, 1984.
- John Stasko, TANGO: a framework and system for algorithm animation, ACM SIGCHI Bulletin, 21, 3, 1990.
- Hideki Koike and Manabu Aida, A bottom-up approach for visualizing program behavior, Proc. on 1995 IEEE Sym. on Visual Languages, 1995.

BOTTOM UP VISUALIZATION

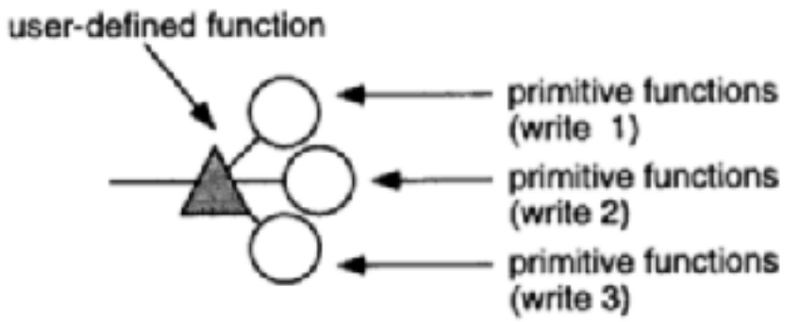




関数block

```
(define (test)
  (write 1)
  (write 2)
  (write 3))
```

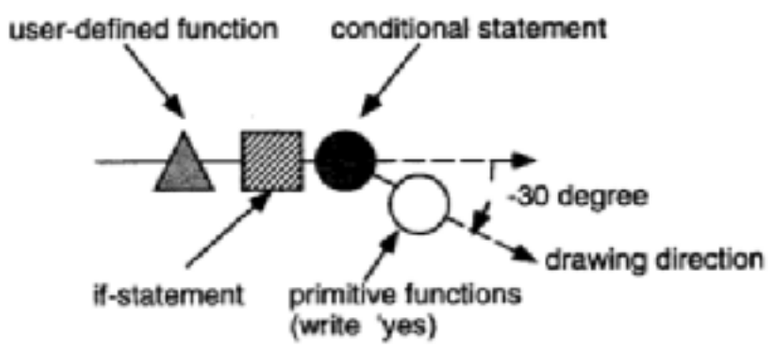
Whenever the function `(test)` is evaluated, the picture shown in Figure 2 is drawn.



条件分岐

```
(define (if-test n)
  (if (= n 1)
      (write 'yes)
      (write 'no)))
```

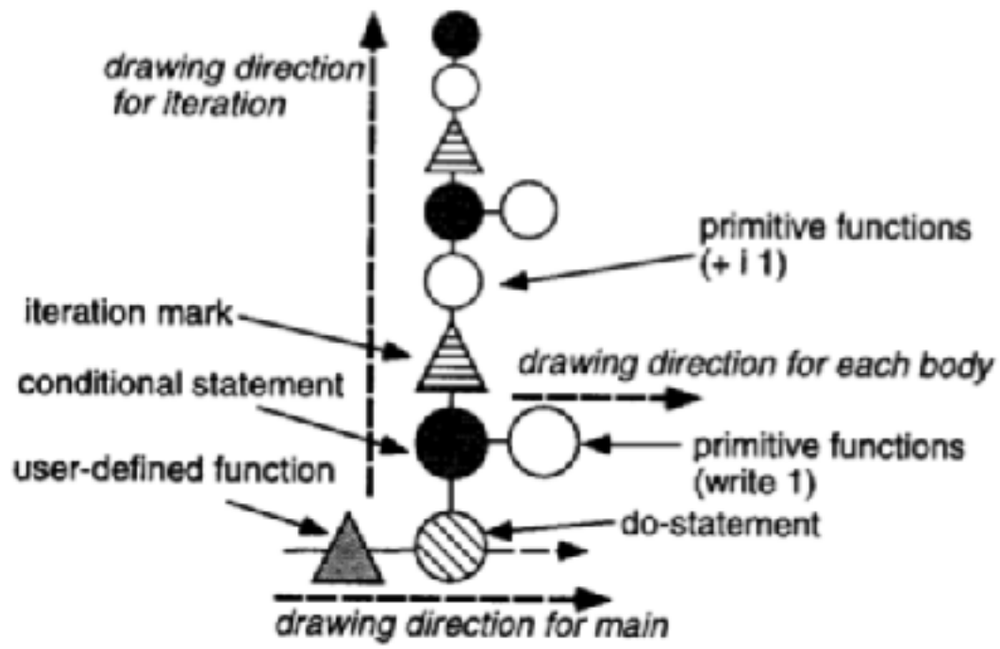
If the function is called as `(if-test 1)`, the picture shown in Figure 3 is generated.



反復

```
(define (do-test)
  (do ((i 1 (+ i 1)))
      ((> i 2))
      (write 1))
```

A picture for `(do-test)` will be drawn as in Figure 4.



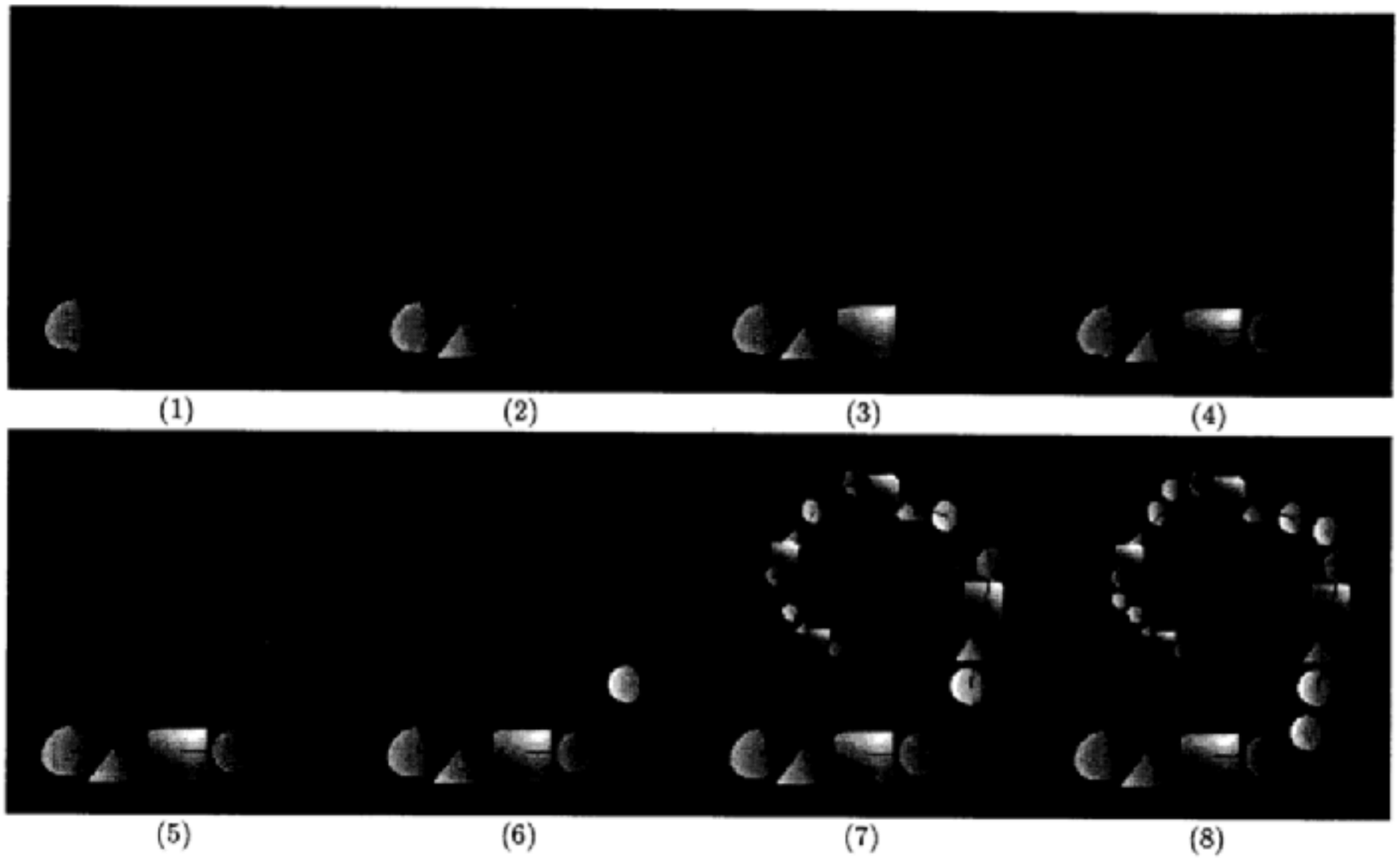


Figure 5: A visualization obtained by evaluating `(fact 5)`. (1) After an initialization process, the system draws a yellow sphere and waits for user's input. (2) When the `(fact 5)` is given to the system, the system draws a pyramid. (3) The system evaluates `(if (= n 1) 1 (* (fact (- n 1)) n))`, and draws a cube. (4) A conditional expression `(= n 1)` is evaluated by the system and a blue sphere is drawn. (5) Since `n` equals to 5, the result of evaluation is false. Therefore, the drawing direction is changed by 30 degree. (6) The system evaluates `(* (fact (- n 1)) n)`. Since the system cannot complete the evaluation before `(fact (- n 1)` is evaluated, it draws nothing at this time. Then, a white sphere for `(- n 1)` is drawn. (7) In this way, the system evaluates `fact` recursively. This figure is just after `(fact 1)` is evaluated. (8) The final image. Each recursive call to `fact` was completed. The system draws white spheres corresponding to the evaluation of primitive function `*`.

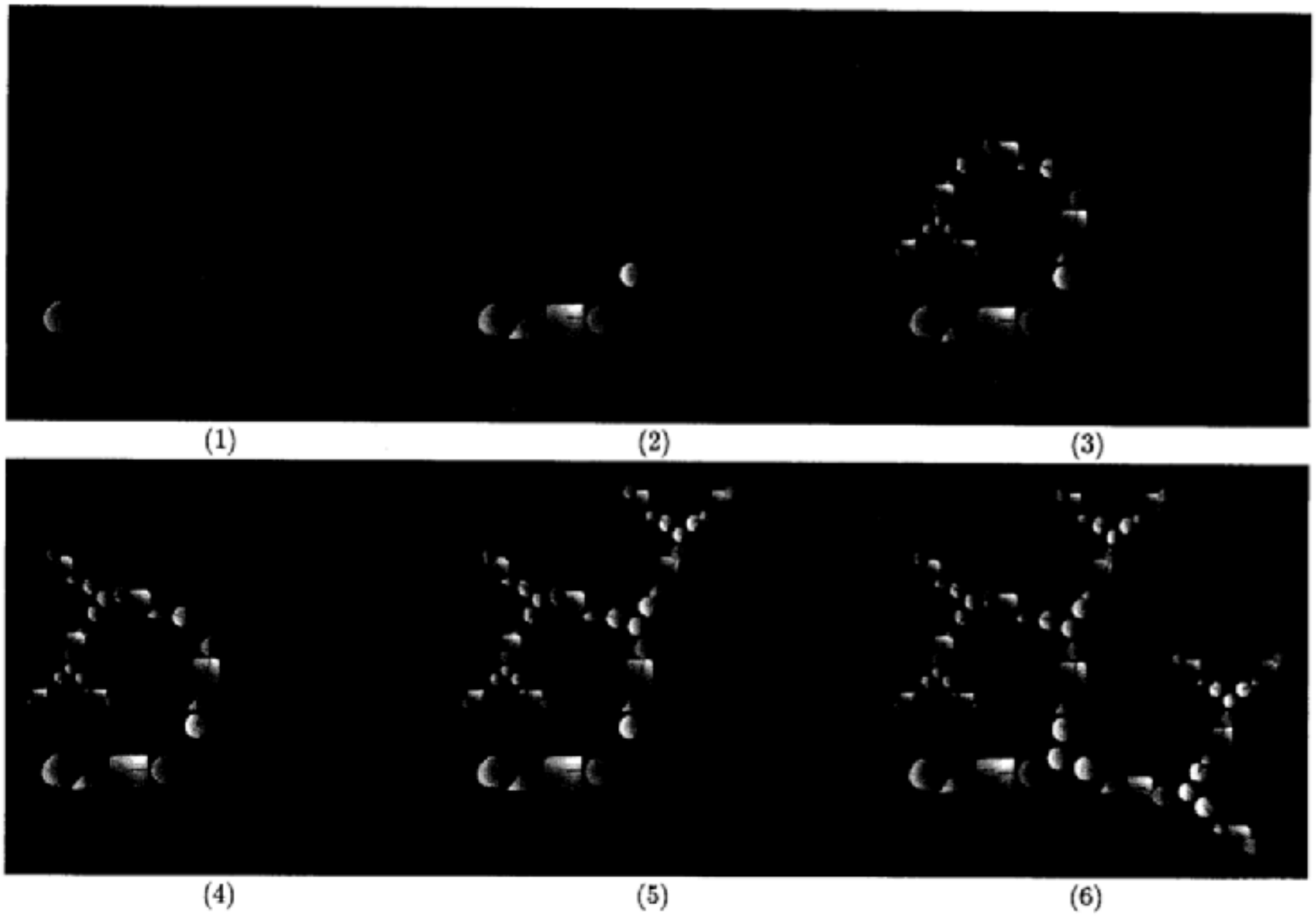


Figure 7: A visualization obtained by evaluating `(fib 5)`. (1) Initial state. (2) The system is going to evaluate `(fib 4)`. (3) The evaluation of `(fib 2)` which is called in `(fib 3)` is finished. (4) The evaluation of `(fib 3)` is finished. (5) The evaluation of `(fib 4)` is finished. (6) `(fib 5)` is completed. We can understand that this program is written using double recursion. We can also notice that the left branch contains exactly the same figure as the right branch. It indicates that this program is doing the same calculation twice.