基本情報技術者試験　アルゴリズム

データ構造

リスト

データとデータを数珠繋ぎにして管理する構造。リストが扱うデータにはポインタと呼ばれる番号が付属される。
ポインタはメモリ上の位置を表す番号。ポインタを書き換えることでデータの追加や挿入・削除が行える。新しいデータへのポインタを書くと挿入、一つ後ろのポインタを書き込むと削除になる。
ただし、リストはポインタ順に辿らないといけないため、配列のように添字を使用して任意のデータ

に直接アクセスすることはできない。
ポインタの持ち方には以下の3種類がある。

単方向リスト

次のデータへのポインタを持つリスト。ただし、一方通行なので先頭のデータから辿っていく必要がある。

双方向リスト

次のデータへのポインタと、前データへのポインタを持つリスト。前後どちらにもデータを辿っていくことができる。

循環リスト

次のデータへのポインタを持つリスト。ただし、最後尾のデータは先頭データのポインタをもつ。よって循環して辿っていくことになる。

キュー

待ち行列。FIFO（First In First Out）先入先出。実例としてGUIプログラムの動作やプリンタの印刷キュー。

スタック

積み上げ行列。LIFO(Last In First Out)後入先出。実例としてプログラムが呼び出したサブルーチンから元の処理に戻れる仕組み。

木構造

階層構造を持つデータで広く用いられるデータ構造。HDDなどのファイルシステムやドメイン名などの管理に使用されている。
データを分岐した節点を辿ることで目的のデータに到達できる。
最上位の節点が「根（ルート）」、そこから「枝（ブランチ）」が伸びて各節点を繋ぎながら末端の「葉（リーフ）」に辿り着く。
葉以外の全ての節が二つの子を持ち、根から葉までの深さが等しい二分木を完全二分木という。

二分木探索

親に対する左分木と右分木の関係が 左の子 ＜ 親 ＜ 右の子 となる探索法のこと。二分木探索法ではデータの探索が容易に行うことができる。

探索アルゴリズム

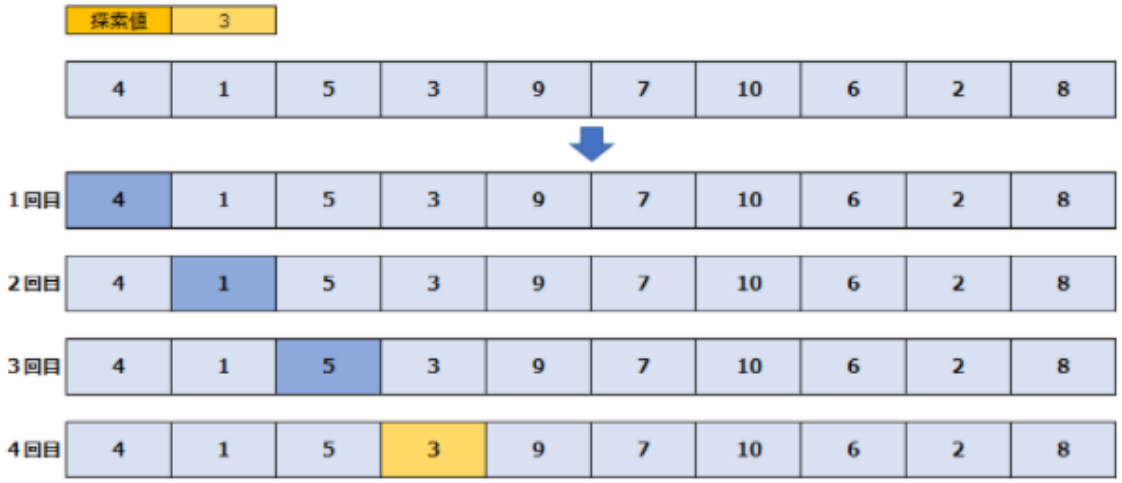
探索アルゴリズムとは複数あるデータ群から目的のデータを探し出すアルゴリズムのこと。

参考ページ

アルゴリズムの計算量（実行時間）を表すものをオーダ記法という。あくまで大まかな処理効率を測るための指標。式では 0 を使用する。オーダが同じであっても処理時間が同じとは同義ではない。

線形探索法（リニアサーチ）

探索対象のデータ群の先頭から順に値を比較していく方法。データ群が整列していなくても探索できるが、使用頻度が高い順に整列されていると少ない回数で目的のデータを探索することが可能。
平均比較回数は目的のデータが先頭にあれば探索回数（最小の回数）は1回、逆に末尾なら探索回数（最大の回数）はN回になるので、その上下回数の和の半分で平均回数になる。



引用元：線形探索、二分探索、ハッシュ探索についてご紹介 | BREEZE

目的の数値がリスト上に存在するか不明確な場合、探索のループ条件
に 配列[i]の値と目的の値Xが等しい ということ以外に 変数iが添字の最大値nよりも大きい という条件が必要になる。1番目の条件はともかく、2番目の条件は探索するにあたって、あらかじめ目的のデータが配

列の末尾に付けておくことで、配列の中に目的のデータが必ず存在するので条件が必要なくなる。この探索ループの終了判定を簡単にするために末尾に付加したデータのことを番兵という。

2分探索法（バイナリサーチ）

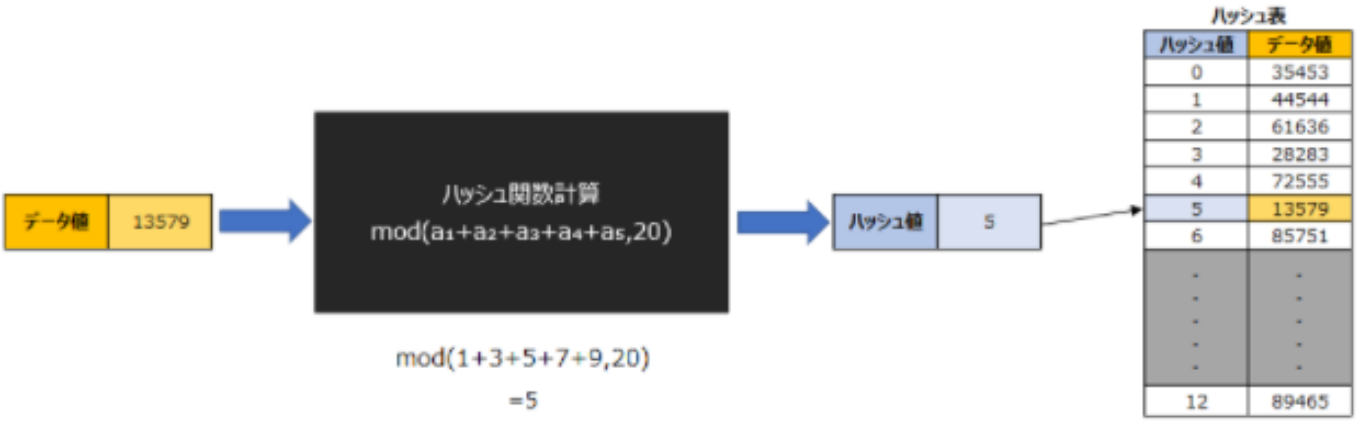
探索データが昇順または降順に整列されている時に用いる方法。データ中央値と探索対象のデータを比較し、その代償により探索範囲を1/2ずつ狭めていくことで線形探索と比べて効率よく探索できることが可能。
平均比較回数は1/2ずつに探索を行うので、Nが2倍になって初めて探索回数が1回増えることになる。つまり以下のように2の対数の関係になる。

引用元：線形探索、二分探索、ハッシュ探索についてご紹介 | BREEZE

ハッシュ探索法

ハッシュ関数を用いてデータの格納位置を参照する探索方法。ただしデータ格納時に同じハッシュ関数を用いた計算式で求める位置へと格納されている必要がある。

例：5桁の数a1,a2,a3,a4,a5を `mod(a1+a2+a3+a4+a5,20)` というハッシュ関数を用いて位置が決められ格納されているとする。※modは余りを求める関数



引用元：線形探索、二分探索、ハッシュ探索についてご紹介 | BREEZE

データ値「13579」のハッシュ値を探索するハッシュ関数

ハッシュ値がデータ値「13579」の格納位置になる。

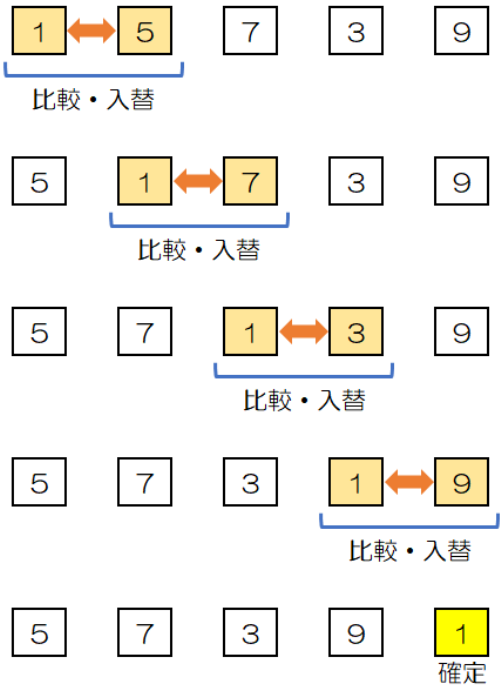
探索対象の格納位置をピンポイントで参照する方法なので圧倒的に小さい計算量で探索を行える(オーダが1なのもそのため)。
ただし前提として格納時にハッシュ関数を用いた計算式で格納されている必要がある。また格納位置が同じになってしまう計算式が他にあると、衝突（シノニム）が発生してしまうので結果が重複しないようなハッシュ関数で格納されている必要がある。

データ整列

引用元：アルゴリズム | ITの基礎知識｜ITパスポート・基本情報
とてもわかりやすい画像ありがとうございます。

バブルソート

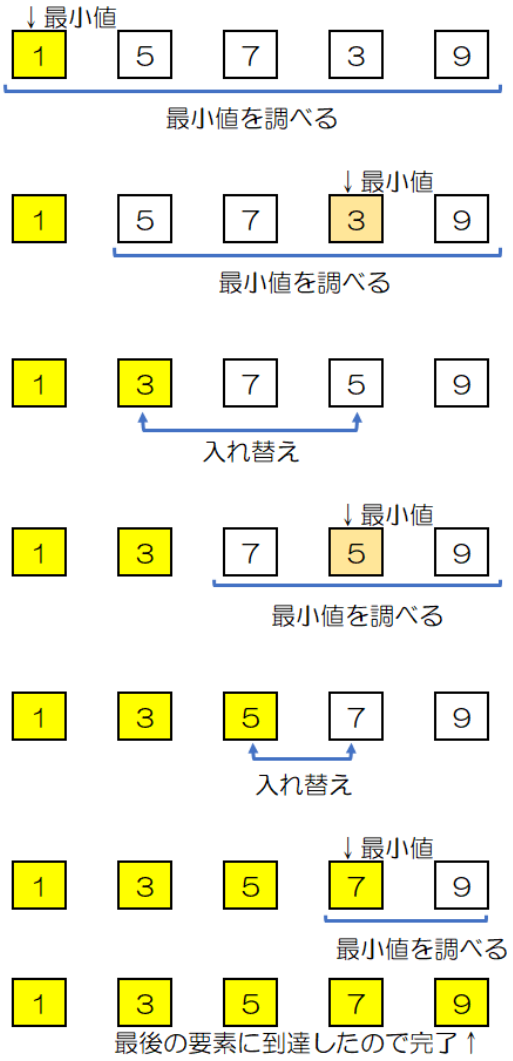
隣接するデータの大小を必要に応じていれかえることで全体を整列させる方法。下図は降順に並べようとしているので「1」が終端で確定している。



n個の要素をバブルソートで整列する回数は以下のようになるためnの2乗に比例する。

選択ソート

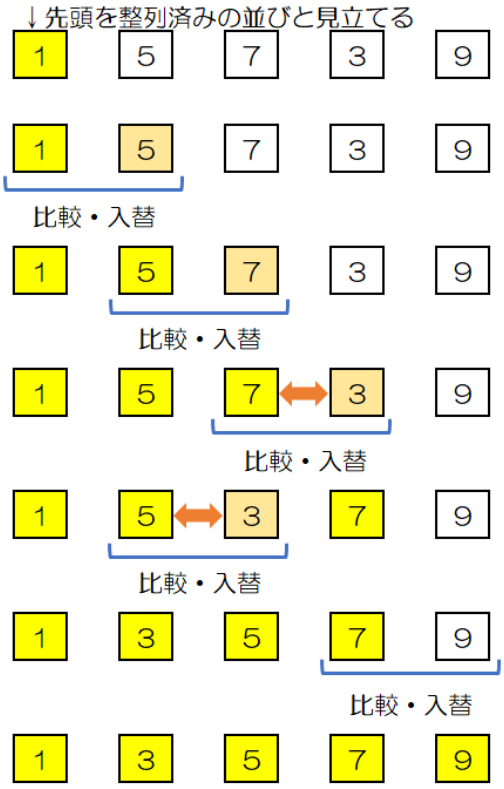
対象となるデータの中から最小値（もしくは最大値）のデータを取り出して先頭のデータと交換を繰り返していくことで全体を整列させる方法。



処理効率はバブルソートと同じく遅い。

挿入ソート（基本挿入法）

対象となるデータ列を「整列済みのもの」と「未整列のもの」に分ける。未整列側からデータを一つずつ整列済みの列の適切な位置に挿入して全体を整列させる方法。



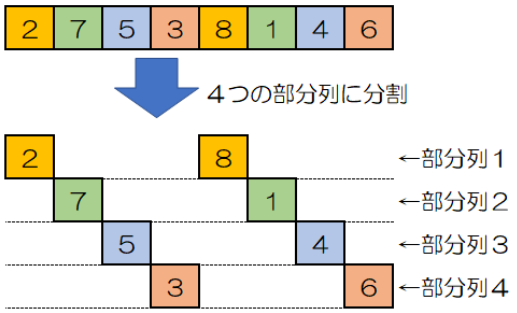
処理効率はバブルソート、選択ソートと同じ遅い。

シェルソート

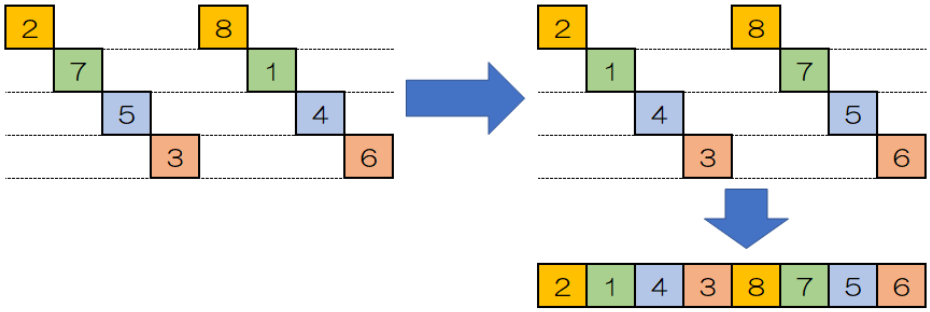
一定間隔おきに取り出した要素で部分列を作り、それぞれ整列して戻す。今度はさらに間隔を詰めて要素を取り出して再度整列させる。取り出す間隔が1になるまでこれを繰り返すことで整列を行う方法。
シェルソートは、元々の並びが整列状態に近いほど高速になる性質があるため、間隔を空けて部分的な並びを作って徐々に整列させることで速度の向上を図っている。

元の配列（要素数：８）

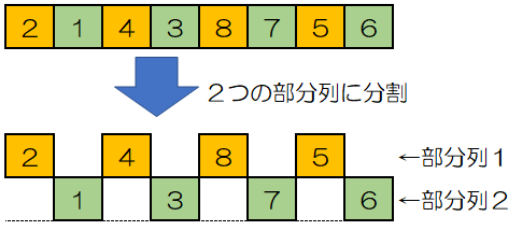
①配列の要素数８の半分、４を最初の間隔として、４つずつ離れた要素２つで４つの部分列を作



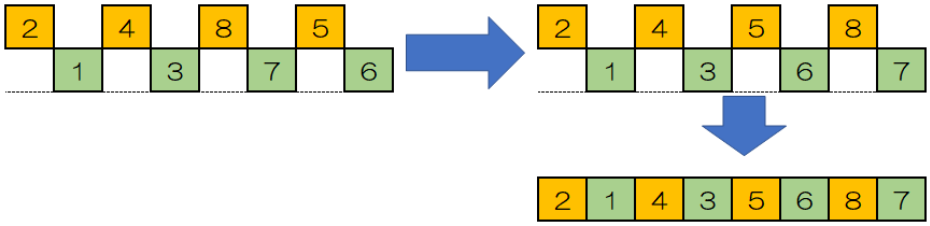
②それぞれの部分列に対して、挿入ソートを行う



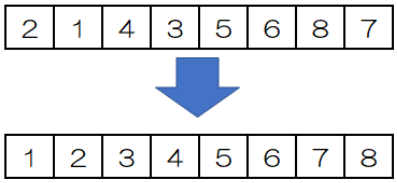
③間隔を半分の２として、２つずつ離れた要素４つで２つの部分列を作る



④それぞれの部分列に対して、挿入ソートを行う



⑤間隔を半分にすると１なので、全ての要素に対して挿入ソートを行なって終了。



クイックソート

中間的な基準値を決めてそれより小さい値グループとそれより大きい値グループに振り分ける。その後それぞれのグループ内でまた基準値を決めて振り分けて...と繰りかすことで全体を整列させる。
反復する毎に、分割されたグループの数は2倍、4倍、8倍というペースで増えていくので、各グループに含まれる要素の数は急激に減っていく。
基準値と要素の値を比較する回数が急速に減っていくためクイックソートが特に高速である。

元の配列（要素数：９）

9	2	7	5	3	8	1	4	6

①並びの中央の要素を基準値とする。

9	2	7	5	**3**	8	1	4	6

↓基準値

②並びの左側から順に基準値と比較し、基準値より大きな要素を見つける。

9	2	7	5	**3**	8	1	4	6

↓基準値

↑基準値より大きい

③並びの右側から順に基準値と比較し、基準値より小さな要素を見つける。

9	2	7	5	**3**	8	**1**	4	6

↓基準値

↑基準値より大きい　　↑基準値より小さい

④2で見つけた要素と、３で見つけた要素を交換する。

1	2	7	5	**3**	8	**9**	4	6

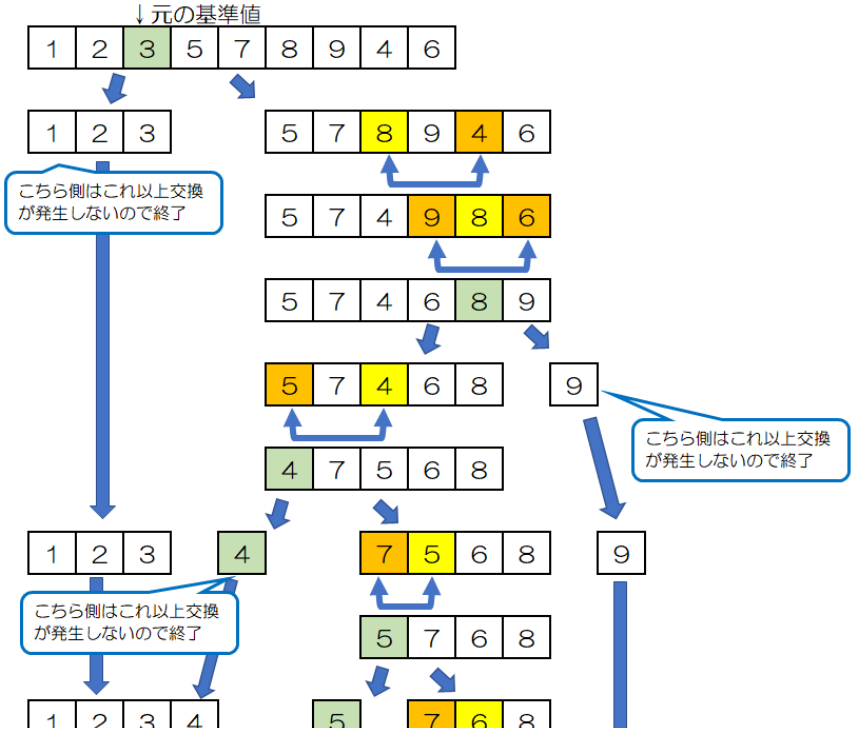
↓基準値

⑤ ２～４を繰り返し、基準値より小さい要素は全て基準値より左に、大きい要素は右になる。
この時、２か３で交換相手が見つからなかった場合は、基準値を交換相手にする。

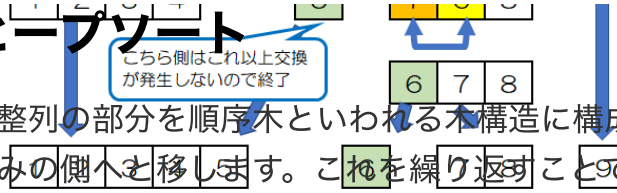
1	2	**3**	5	**7**	8	9	4	6

↓基準値

⑥基準値で元の並びを分割し、１～４を繰り返す。交換が発生しなければ、その並びの整列は完了。



ヒープソート

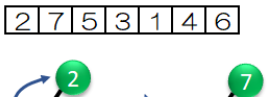


未整列の部分を順序木といわれる木構造に構成して、そこから最大値もしくは最小値を取り出し整列済みの側へと移します。これを繰り返すことで、未整列部分を縮めて整列を行う方法。
二分木は木の階層ごとに、部分木の数が2倍、4倍、8倍と増えていく。値の大小比較の回数は2つの部分木の根を比較するだけなので、ヒープソートは特に高速とされている。

元の配列（要素数：7）

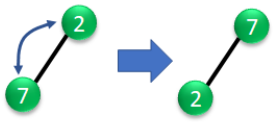
2	7	5	3	1	4	6

① １番目の要素を根とする。



② 次の要素を葉とする。

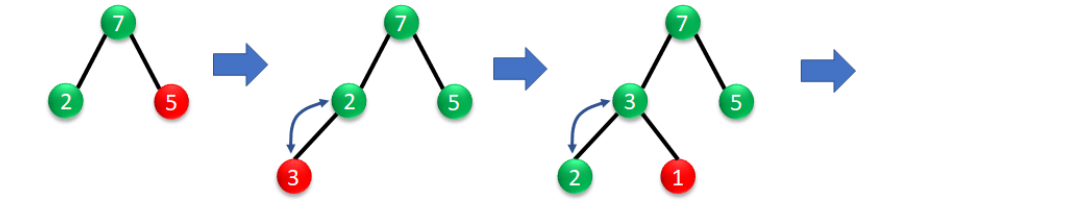
③ 葉が親の値より大きければ、
親と交換する。

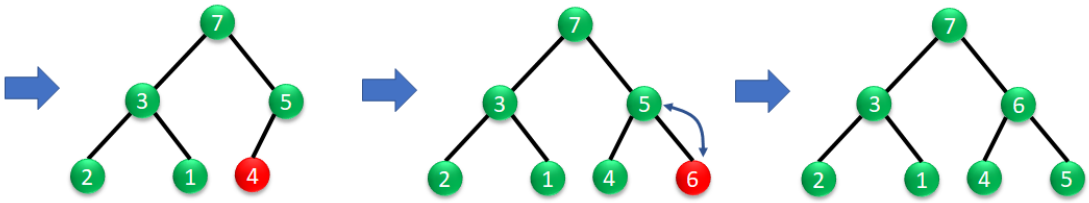


④ 交換した結果が、その親の値より大きければ、
親と交換する。

⑤ ④を交換が不要になるか、根と交換するまで繰り返す。

⑥ ②～⑤を繰り返し、ヒープを構築する。





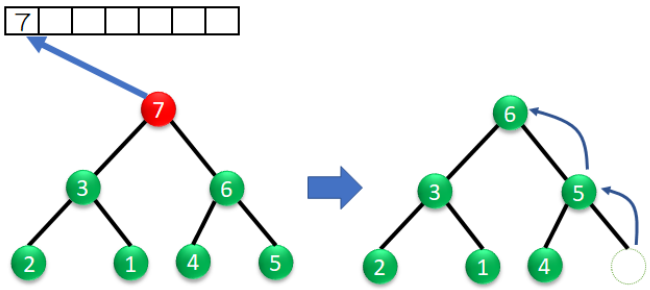
これでヒープが構築されたので、今度はヒープから要素を取り出す。

⑦ ヒープの根を取り出す。

⑧ 空いた位置には、その子のうち
大きい方を移す。

⑨ ⑧を葉に到達するまで繰り返す。

⑩ ⑦～⑨をヒープが空になるまで
繰り返す。



数値解析

モンテカルロ法

確率を近似的に求めるために使われる手法。乱数によるn回のシミュレーションを行い、ある事象がm回起これば、その事象の起こる確率は m／nで近似できる。
試行回数nが大きくなるほどより良い近似値を得ることができる。