1章

(1)アルゴリズムの定義→　与えられた問題の正しい答えを求めるための“上手いやり方”

(2)アルゴリズムを比較するための基準→　アルゴリズムの評価基準

(3)3 の倍数かどうかの判定について、数学的に判定する観点 VS コンピュータで判定する際の観点で問題の質の違い

→数学・・・「桁和が 3 で割り切れるなら整数は 3 の倍数」という性質そのもの/コンピュータ・・・表現・オーバーフロー・I/O・基数・堅牢性といった資源制約の中でどう実装するかが本質.

(4)コンピュータの計算資源（メモリ，CPU）の観点からアルゴリズムの性能を比較する際の評価指標 2 点.

→時間計算量(実行時間の速さ)，空間計算量（メモリの増加の程度）

(5)コンピュータの計算資源の観点からアルゴリズムの性能を比較する際の評価指標を 3 点あげよ。

→最悪時間計算量，平均時間計算量，最良時間計算量

(6)時間計算量の性能を比較する際に用いられる評価指標→オーダー記法(漸近的（∞に近い）な時間計算量を表記する）

(7)オーダ記法が使われる理由→アルゴリズムの性能を装置や実装に依存せず，理論的に比較できるようにする

(8)不良品のボールを見つけるまでの時間の回数を求めよ。 →$\left(\frac{1}{2}\right)^2 \times n = 1 \therefore 2^k = \mathrm{n} \therefore \log_2 2^k = \log_2 n \therefore k = \log_2 n$回

(9)テニスボールが 10 から 100000 まで増加しても、最大の実行時間が約 170 秒で収まる理由を説明せよ。

→一回の操作で探索範囲を半分に絞れる．数が増えても探索回数は対数的で，計算時間量はわずか.

(10)オーダ記法は主要部（n=∞の時の最も影響の大きい項）に係数を省いたもの

(11)計算時間量の大小関係→$logn < \sqrt{n} < \mathrm{n} < nlogn < n^2 < n^3 < \cdots < 2^n < \mathrm{n!}$

(12)入力サイズ n が大きくなると実行時間はどんな観点で比較できるか→漸近的な時間計算量

(13)最大値の計算の最悪時間計算量→O(n)　(14)等しい整数の入力の整数の出力→外側の for 文は n-1 回内側の for 文は外側に依存するので，n-i 回であるため，n-i を n-1 回足し合わせると，$\frac{n(n-1)}{2}$となり，$O(n^2)$となる.

2章

(1)配列からデータを消去：最悪時間計算量→O(n)　（2）追加：最良時間計算量→O(1)/最悪時間計算量→O(n)

(3)連結リストの考え方→1 つのデータをレコードという格納場所で管理，各レコードをポインタで指し，データの列を作っている.

(4)連結リストを実現するためのデータ構造の考え方→各要素をノードとして持ち、ポインタで順につなぐことで構成されるため、配列のように要素を移動させる必要がない。新しいノードを作成し、そのポインタ部を旧先頭ノードに向け、head を新しいノードに更新するだけで追加できる.

(5)連結リストのデータの追加・削除の特徴→先頭：O(1)/途中・末尾：O(n)．よって探索は遅いが，先頭での追加や消去は高速である

(6)スタックのデータ構造としての特徴→スタックは処理要求の遅いものからの順序でデータの格納，取り出しをする．一番上からしか出し入れできないため，計算量は定数時間 O(1)．(7)スタック実装の考え方→格納 push/取り出し pop

(8)キューの特徴→キューは処理要求がはやいものから処理し，要素の追加は末尾から行い、削除は先頭から行う。

(9)キュー実装の考え方→格納「enqueue」，取り出し「dequeue」．配列の場合実行時間はどちらも O(1).

(10)スタックの実装注意点→配列を用いた空のスタックの準備には，十分なサイズの配列の準備，変数 top の値の初期化が必要.

(11)キューの実装注意点→配列を用いた空のキューの準備には，変数 left，right の初期化が必要．変数 right はキューに格納されている右端のデータから一つ右の格納場所を表す.

3章

(1)二分木の特徴→すべての節点が 2 個以下の個しか持たない．探索や分類に使いやすい.

(2)完全 2 分木の葉の数は高さを h とすると，$2^{h-1} = O(2^h)$となる理由→完全 2 分木において，レベル 0 の節点数は 1．レベルが 1 上がるごとに節点数は 2 倍になる．よってレベルが k の時節点数は2^k個．ここで，木の高さは，木に含まれる葉のレベルの最大値に 1 加えたものだから2^{h-1}．計算量は，オーダ記法より，主要項2^hを取り，係数を削除すると$O(2^h)$となる.

(3)完全 2 分木の高さは，葉の数を m とすると，$1 + log_2 m = O(logm)$である理由→　葉の数が m 個の時，高さを h とすると$m = 2^{h-1}$．よって$h = 1 + log_2 m$．計算量はオーダ記法より，主要項$log_2 m$を取り，係数を削除すると$O(logm)$となる

(4)(5)完全 2 分木の節点数は，高さを h とすると$2^h-1=O(2^h)$となる理由→レベルが k の節点数は2^k．高さを h とすると，節点数は各レベルの節点数の和より，

$$1+2+4+8\cdot\cdot+2^{h-1}=\sum_{k=0}^{h-1}2^k$$

この式は初項が 1，公比 2 の等比数列の和である．これを変形して2^h-1．オーダ記法より主要項2^hを取り，係数を削除すると$O(2^h)$となる．

(6)完全 2 分木の高さは，節点の数を n とすると，$log_2(n+1)=O(logn)$となる理由→(4)(5)より，n=2^h-1 ∴　$h=log_2(n+1)=O(logn)$

(7)二分木を配列で表現できる説明→まず，木の根の節点を 1 とする．番号 i をもつ節点の子を左が 2i，右が 2i+1 とする．次に，i の節点のデータを T[i]に格納する．よって i の節点の子は，T[2i]と，T[2i+1]に格納される．また，i の節点の親データは[i/2]で T[k]に格納される

(8)細胞 10 個が毎分 2 倍になり 1 個死滅するアルゴリズムの n 分後の細胞数→c(0)=10/$c(n)=2\cdot c(n-1)-1=9\cdot 2^n+1$

(9)(8)のアルゴリズム→初期値を 10 とし，1 分ごとに『2 倍して 1 引く』操作を n 回繰り返す」ことで n 分後の細胞数を求める

(10)和の計算が再帰木を用いて求められる説明→再帰木の高さは繰り返し関数が呼び出される回数を表している．また，再帰を終了するのに必要な時間計算量を表している．よって，再帰アルゴリズムの時間計算量は，再帰木のすべての節点の和に等しくなる．

(11)(10)の計算量→n 個の和を求めるアルゴリズム全体の時間計算量を T(n)とおく．このとき，定数個の演算と時間計算量が T(n-1)となる関数の再帰的な呼び出しから構成されているため T(n)= T(n-1)+c(n＞2 のとき)，T(n)=c(n=1 のとき)

(12)和の計算を 2 項の数列に分けて計算する方法の漸化式→T(n)=2T(n/2)+c(n＞2 のとき)/T(n)=c(n=1 のとき)

(13)(12)の時間計算量→$\sum_{i=0}^{log_2 n}c\times 2^i$=c(2n-1)=O(n)(難しく書いてあるが，結局は再帰によって n 個分割され，最後に n 買い足されている)

4 章

(1)探索の定義→入力として n 個のデータと値 x が与えられたときにデータの中から x=di となる di を見つける操作.

(2)線形探索の最良時間計算量，最悪時間計算量の考え方→最良時間計算量：D[0]に値がある場合で O(1)/最悪時間計算量：配列 D に値が含まれてない時で O(n)

(3) 2 分探索のアルゴリズム→探索範囲の右端と左端を変数 left，right．中央の場所を変数 mid．While 文で 2 分探索の繰り返しを表し，探索範囲が 1 になった時(left,right が等しくなる)繰り返し処理を終了する．

(4)最悪時間計算量→O(logn)　(5)ハッシュ法と線形探索・2 分探索との違い

→ハッシュ：平均探索時間が O(1)と非常に高速/線形探索・2 分探索：平均探索時間が O(n)，O(logn)とハッシュ法に比べて遅い．

(6)ハッシュ法の探索例→データをハッシュ関数で配列の位置に割り当て，衝突が起きると次の空き場所を探して格納する．配列の最後の H[1.5n-1]で格納できなかったら，H[0]に戻る．

(7)ハッシュ関数に求められる特性→同じデータが入力されたとき，必ず同じハッシュ値を返す必要がある．

(9)探索した結果，データが格納されている場合の回避方法→オープンアドレス法：衝突が起きたときに，次の空きを探して格納する．

チェイン法：各配列の要素をリストにし，同じハッシュ値のデータをつなげて格納する．

(10)配列を用いたハッシュ法の性質→索時間量の平均アルゴリズムは，O($\frac{m}{m-n}$).配列のサイズが大きければ大きいほど時間計算量は小さくなる．ハッシュ法の時間計算量は入力データサイズに依存せず，定数時間で実行できる．

(11)探索の実行時間の比較→線形探索：データのサイズに比例して増加する．

2 分探索・ハッシュ法：非常に高速．ハッシュ法はデータ数によらずほぼ一定時間で実行でき，2 分探索よりも高速である．

5 章

(1)アルゴリズムにおけるソートとはどのような操作か→与えられたデータを順に並べるとういう操作

(2)選択ソートの考え方→外側のループで整列済み部分と未整列部分を区別し，内側のループで未整列部分の中の最大値を探す.

(3)選択ソートの最悪計算時間→for 文の 2 重ループより$O(n^2)$(最良も$O(n^2)$と同じ)

(4)挿入ソートの考え方→挿入ソート：すでに整列されている部分列に対して，要素を適切な位置に挿入していくことで全体を順に整列させる．説明：まず，配列の整列とみなす．次の要素 D[i]を取り出し，一時変数 x に保存する．整列済みの部分の中で，x より大き要素を右にずらす．空いた位置に x を挿入する．これを末尾まで繰り返す.

(5)挿入ソートの最悪時間計算→for と while が実行されるので$O(n^2)$(最良はすでにソートされており，while 文が実行されないため O(n)

(6)　ヒープソートを構成する 2 つの操作→ヒープ：大量のデータを特定の順序で記憶するためのデータ構造

2 つの操作：プッシュヒープ：ヒープ H に対してデータ x を格納する/デリートマキシマム：ヒープ H から最大の値をもつデータを削除し，取り出したデータを入力する

(7)ヒープソートはどのようなデータ構造の特徴を用いたソート方法なのか→優先順位つきの処理を実現するためのデータ構造

(8)ヒープソートに用いられるヒープに求められるデータ構造としての性質を説明せよ

→2 分木の最大レベルを L とすると，0 <k<L-1 を満たす各レベルには 2^k 個の節点が存在し，レベル L に存在する葉は，そのレベルに左詰めにされる/各節点に保存されるデータは，その子に保存されるデータより大きい．

(9)ヒープのデータの追加の手順→末尾に新しいデータを追加する（完全二分木の形を保つため，空いている一番下の右側に入れる）

→親ノードと比較する（最大ヒープなら「親より大きいか，最小ヒープなら「親より小さいか」）

→条件を満たさない場合は親と交換する（ヒープ条件が崩れていれば，親子を入れ替える）

→根に到達するか，条件を満たすまで繰り返す

(11)ヒープから最大値取り出し時の手順→根から最大値を取り出し，右端の葉のデータを格納．移動したデータの節点と，その子の節点の比較．子節点が存在しなければ終了．子節点が 1 つの場合は，このデータが小さい場合は終了．大きい場合はデータを交換し比較を繰り返す．子節点が 2 つの場合，両方とも小さければ終了．片方が大きかったら，大きいほうのデータと交換し比較を続ける．

(13)ヒープを表す配列に対するデータの取り出し手順

→まず，根（最大値）を取り出し，配列の末尾のデータを先頭に移す．子ノードの 23.24 のうち大きいほうを選び 5 と 24 を交換する．

(14)ヒープソートの手順の構成→配列に格納された n 個のデータについて，push_heap を n 回繰り返し，ヒープを表す 2 分木をつくる．これに対し，delete_maximum を n 回繰り返し，データを取り出した順に並べる．

(15)ヒープソート最悪時間計算量の考え方→最良，最悪ともに O(nlogn)．N 個のデータが格納されたヒープに対する時間計算量は O(logn) だが，データサイズが 1 から n までの場合は，$2\times\sum_{i=1}^{n} logi < 2\times n\times logn = O(nlogn)$

6 章

(1)クイックソート，ソートの考え方→まず，分割の基準となる値（基準値）を入力データから適当に選び，次に，入力データを基準値よりも大きいか小さいかで 2 分割する，このとき基準値よりも小さいデータは基準値よりも左に，大きいデータは右に置く．その後また同じ手順を繰り返して分割した，集合のデータが 1 つになるまで繰り返すとすべてのデータがソートされる．

(5)クイックソートの再帰説明→Partition で 1 回並び替えの区切りを作り，大きいほう小さいほうをそれぞれ独立して整列させる．この処理を再帰的に繰り返すことで全体が整列されるから．(6)partition に求められる処理→基準値を決めて配列を大きい順，小さい順に分けること (7)partition の実行例→基準値 11 とし，右端の 6 と交換する．と i と j の位置を決めそれらを交換する．この操作を i>j となるまで繰り返し最後に i と右端のデータを交換する．

(8)分割操作の考え方→基準値を中心に小さい値を左に，大きい値を右に整理して基準値を正しい位置に置く処理．

(9)クイックソートの最悪時間計算量が O(n^2)になるときの入力例→基準値を先頭や末尾から選ぶとき

(10)クイックソート，最悪時間計算量求め方考え方→再帰木の高さが n，節点の時間計算量が cn，c(n-1),・・c のとき再帰アルゴリズムの時間計算量は，再帰木のすべての節点が表す計算量に等しいので，その和を求めると，$\sum_{i=0}^{n-1} c(n-i) = \sum_{i=1}^{n} ci = c\frac{n(n+1)}{2} = O(n^2)$

(12)クイックソートが最良時間計算量で動作するときの再帰木の構成→再帰木の各レベルの節点に含まれる時間計算量の和が等しい時．

(13)クイックソート，最良時間計算量→O(logn×cn)=O(nlogn)

(14)ソートアルゴリズム性能比較→選択ソートと，挿入ソートアルゴリズムは，時間計算量の$O(n^2)$に比例する実行時間になっていて，時間計算量が O(nlogn)のソートアルゴリズムがすべて 1 秒以下で実行できることと比較すると実行にとても時間がかかるとわかる．また，同じ O(nlogn)という時間計算量を持つ 3 つのソートアルゴリズムの中で，クイックソートがもっとも高速．

(15)ソートに求められる安定性→与えられたデータを決められた順番に並べる．・同じ値のデータは，入力の順番通りに並べる．この二つの条件を満たすこと．

まとめ

1章

・アルゴリズムの計算時間量は(アルゴリズムの評価基準/入力サイズの関数を用いて評価される)

・常に最良時間計算量T_B(n)>最悪時間計算量を$T_w(n)$の関係

2章

・配列　　→格納するデータをあらかじめ決めておく/任意の格納場所に O(1)でデータの読み出しと書き込みが可能

・連結リスト→データサイズの変更自由/先頭データ消去は O(1)/1 つのデータをレコードで管理

・スタック→LIFO/処理要求の順番が遅いものから処理する/時間計算量は O(1)

　　空の初期化は配列の添え字が 0 からなので top=-1

・キュー　→FIFO/処理要求の順番が早いものから処理する/時間計算量は O(1)

　　からの初期化は配列を環状キューで使う場合は right = 0, left = 0

木構造

木構造(レベル:k, 高さ:h, 葉:m, 節:n)

葉: $n = 2^k = 2^{h-1} = O(2^h)$/高さ: レベル+1: $h = 1 + \log_2 m = O(\log m)$

節: $\sum_{k=0}^{h-1} 2^k = 2^h - 1 = O(2^h)$/高さ: $\log_2(n+1) = O(\log n)$

完全3分木

$n = \sum_{k=0}^{h-1} 3^k = 3^0 + 3^1 + 3^2 + \cdots + 3^{h-1}$

$-)\ 3n = 3^1 + 3^2 + \cdots + 3^{h-1} + 3^h$

$n - 3n = 3^0 - 3^h = 1 - 3^h$

$-2n = 1 - 3^h \therefore n = \frac{3^h - 1}{2}$

3章

・木とは→データ間の順序，依存関係を表す/レベル 0(根)から始まる

・節→ノード全て/葉→子を持たない/高さ→レベル+1

・フィボナッチ数列の時間計算量→O(2^n)

4章

・線形探索の時間計算量→O(n)

・２分探索の時間計算量→O(logn)(再帰ありでも同じ)

・ハッシュは配列が大きくなればなるほど計算量は小さくなる

・ハッシュの最悪時間計算量→データ：n 個，サイズ：m の配列だと$O\left(\frac{m}{m-n}\right)$(*平均時間計算量*) $= O\left(\frac{1.5n}{1.5n-n}\right) = o(1)$

5，6章

・ソート→全順序関係が成り立っていなければならない

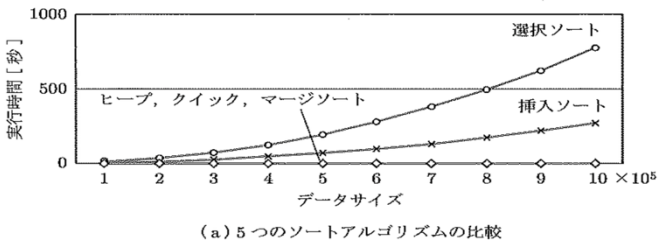
・　選択ソートの時間計算量→最良：O(n^2)，最悪：O(n^2)

・　挿入ソートの時間計算量→最良：O(n)，最悪：O(n^2)

・ヒープソートの時間計算量→最良：O(nlogn)，最悪 O(nlogn)

　　データ追加，最大データ削除：O(logn)（木の高さに等しい）

　　配列において親 T[n/2]，子 T[n]の関係



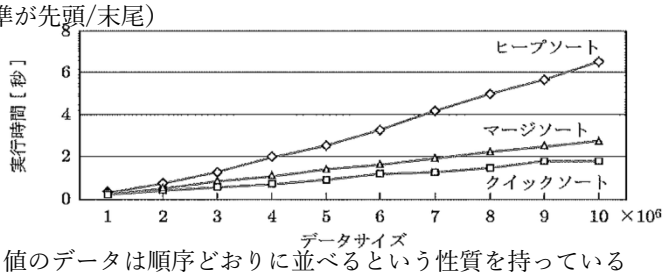
（a）5 つのソートアルゴリズムの比較

・クイックソート→基準値より小さいと大きいに分割する関数 partition を再起的に行う

・クイックソート時間計算量→最良：O(nlogn)，最悪：O(n^2)（基準が先頭/末尾）

　　Partition の時間計算量→O(n)

　　木の高さ→O(logn)



・クイックソート→基準値によって時間計算量が変わる

　　同じ値はバグが発生しやすい

　　クイック/ヒープソートは安定しない，安定なソート→同じ値のデータは順序どおりに並べるという性質を持っている

Partition

入力 (35 21 4 49 55 19 12 32 24 42)

0 1 2 3 4 5 6 7 8 9

i, j, k

① swap(D[i], D[right])

(42 21 4 49 55 19 12 32 24 35)

D[i]が基より小さい間 i++

D[j]が基より大きい間 j--

② swap(D[i], D[j])

(24 21 4 49 55 19 12 32 42 35)

③ (24 21 4 32 55 19 12 49 42 35)

④ (24 21 4 32 12 19 55 49 42 35)

j<iより swap(D[i], D[right])

⑤ (24 21 4 32 12 19 35 49 42 55)