アルゴリズムとデータ構造 中間試験問題

2024 年 11 月 27 日 10:35〜11:35，担当：井上

注意事項：持ち込み禁止です．机の上には筆記用具と時計，ちり紙等以外は置かないこと．携帯電話は電源を切って鞄等の中に入れてください．第 2 問以降については，手順を計算用紙などで図示しながら考えるなどして，凡ミスを避けること．また，全て解答を終わったとしても，十分納得できるまでチェックする時間を惜しまないこと（エンジニアの心得）．

1. 以下の文章中の空欄 **(A)**〜**(V)** に入るものを選択肢から選び，記せ．(44 点)

- 特定の問題をコンピュータで解く場合，コンピュータには人間のような常識や経験的直感がないので，問題を解く手順である **(A)** とその過程で扱うデータをどう管理するかを定める **(B)** とを詳細に記述する必要がある．
- 2 数の最大公約数を求めるのに使える (A) として，**(C)** がある．また，(C) による処理の過程で効率を高めるには，何度も同じ数の引き算をする代わりに **(D)** の計算を用いることができる．
- 解析的に解けない数理的計算をコンピュータを用いて行うことを **(E)** という．ただし，コンピュータでの計算は常に近似的で，誤差を含む．例えば，データ型の桁数制限のために四捨五入を行うことによる誤差を **(F)** という．また，無限級数の計算を有限時間で打ち切ることで生じる誤差は **(G)** である．
- 解析的に解けない積分計算を (E) により解くことを **(H)** という．その初歩的な方法として，積分区間を細かく区分し，各区分の積分値を台形の面積として近似する方法を **(I)** という．さらに精度を上げる方法としては，区間積分値を放物線とx軸で囲まれた領域の面積として近似する方法があり，これは **(J)** と呼ばれる．
- (A) の良し悪しを定量的に評価する方法として，特定の (A) で問題を解く際の繰り返し回数等を問題のサイズnで関数化したものを **(K)** という．また，(K) の値自体は用いるコンピュータ等によって変動するので，例えば多項式なら最大次数以外の項を無視したり，定数係数を無視したりして (K) がどのような関数であるかだけを表した表記とする．これを (K) の **(L)** という．
- 多数のデータを小さい順などに並べ直す作業を **(M)** といい，それに対して様々な (A) が提案されている．例えば，データ中の最小のものをデータ列の先頭と交換する操作を繰り返す方法は **(N)** であり，途中まで (M) できているデータ列に次のデータを挿入することを繰り返す方法を **(O)** という．これら 2 つの (A) の (K) の (L) は **(P)** である．
- 多数のデータから目標データを探し出す作業を **(Q)** という．単純にデータを一つずつ調べる逐次探索に対して，**(R)** という方法では，探索の手がかり（キー）を有限の数値に写像し，その数値に対応した配列要素にデータを格納することで (Q) を高速化している．
- 解くべき問題を「同じ解き方で解ける，サイズの小さな副問題」に分解して副問題を先に解くという考え方で関数を作る場合，問題を解く関数はその処理の中で，自分と同じ関数をより小さな引数を使って呼び出すような構造となる．このような手法を **(S)** という．
- (S) を用いた (M) の (A) に **(T)** がある．(T) では，データ列を「軸」と呼ばれる基準値に基づいて 2 つのグループに分割するという操作を (S) を用いて繰り返す．その (K) の (L) は **(U)** である．
- データの一時保管場所として広く用いられる (B) の一つに **(V)** がある．(V) では，最後に格納したデータが最初に取り出される LIFO (last in first out) 方式でデータが扱われ，(S) における関数呼び出しにおいて関数の内部データを格納しておくなどの用途に使用される．

選択肢

- 直接選択法
- 数値計算
- 中点則
- 漸化式
- サーチ
- 逐次探索
- コンパイル
- ニュートン法
- エラトステネスのふるい
- 剰余
- フィールド
- オイラー法
- スタック
- オーバーフロー
- $\sqrt{n}$
- バブル・ソート
- $n^{1.25}$
- 台形則
- クイック・ソート
- 丸め誤差
- シンプソン則
- キュー
- 打ち切り誤差
- 浮動小数点
- ソート
- プログラム
- パターンマッチング
- アルゴリズム
- 構造体
- メモリ空間
- 2 分法
- n^3
- データ構造
- 数値積分
- ポインタ
- n
- オーダ
- マージ・ソート
- 再帰
- $n \log n$
- n^2
- レコード
- 計算量
- 桁落ち
- 基本挿入法
- $n\sqrt{n}$
- チェイン法
- 番兵
- 漸化式
- ユークリッドの互除法
- シェル・ソート
- ヒープ・ソート
- ハッシュ
- $\log n$
- 2 分探索

(裏面に続く)

2. 以下は, 教科書に書かれた基本挿入法のソース・コードを改変したものである. 変更部分 (太字・下線) は, データ列の初期化, データ列の表示の関数化, およびソートの途中経過の表示に関するものである. これについて, 以下の問に答えよ. (24 点)

```
#include <stdio.h>
#include <stdlib.h>
#define N 6
void disp(int *);

void main(void)
{
  int a[]={80,41,35,90,40,20};
  int i,j,t;

  disp(a);

  for (i=1; (A); i++){
    for (j=i-1; (B); j--){
      if ((C)){
        t=a[j];
        a[j]=a[j+1];
        a[j+1]=t;
      }
      else (D);
    }
    disp(a);
  }
}

void disp(int a[])
{
  int i;

  for (i=0; i<N; i++)
  printf("%d ", a[i]);
  printf("¥n");
}
```

(1) 空欄 **(A)**～**(D)** に入るものを記せ.
(2) プログラム実行時の表示内容を記せ (disp(a)の呼び出し位置に注意すること).

3. 以下は, 教科書に記載された組み合わせの数を再帰的に求める関数 combi() である. ただし, 関数が呼び出されるたびに画面表示をするコードを追加した. このとき, combi(4,2) を呼び出したときの画面表示がどうなるかを示せ. (再帰呼び出しの連鎖がどのように起こるかを図に描いてみるなどして考えるのがよい) (16 点)

```
long combi(int n,int r)
{
  printf("%dC%d¥n", n, r);

  if (r==0 || r==n)
    return 1L;
  else
    return combi(n-1,r)+combi(n-1,r-1);
}
```

4. 以下は, 教科書に書かれたスタックのソース・コードである. ただし, 配列 **stack[]** のサイズは **2** とした. このプログラムを実行し, 以下の順序でキー入力をしたときにスタックから取り出されるデータを順序も含めて示せ. (キー操作においてリターンキーは省略した) (16 点)

キー操作：**i,1,i,2,i,3,o,i,4,i,5,o,o,o**

```
#include <stdio.h>
#define MaxSize 2

int stack[MaxSize];
int sp=0;

int push(int);
int pop(int*);

void main(void)
{
    int c,n;

    while (printf("]"),(c=getchar())!=EOF){
        rewind(stdin);
        if (c=='i' || c=='I'){
            printf("data--> ");
            scanf("%d",&n);rewind(stdin);
            if (push(n)==-1){
                printf("Full.¥n");
            }
        }
        if (c=='o' || c=='O'){
            if (pop(&n)==-1)
                printf("Empty.¥n");
            else
                printf("%d¥n",n);
        }
    }
}

int push(int n)
{
    if (sp<MaxSize){
        stack[sp]=n;
        sp++;
        return 0;
    }
    else
        return -1;
}

int pop(int *n)
{
    if (sp>0){
        sp--;
        *n=stack[sp];
        return 0;
    }
    else
        return -1;
}
```

※ 問題 2 以降については, 正しい最終結果のみが記載されていれば満点とするが, 最終結果のみが記載されており, それが間違っている場合は, 部分点をつけにくい. 一方で,「なぜこの結果になると思われるか」についての考え方が説明されていれば, 最終結果が間違っている場合でも, どの部分までは理解できているかを評価できるので, **できるだけ部分点を与えることができるようになる**.

以上