問３　【解答ウ】

この流れ図では，最後に変数“合計”を出力していることから，配列Ｔの要素T(1)～T(10)に格納されたデータの合計を変数“合計”に求める。また，要素T(1)～T(10)を加算（合計）していくためには，変数Ｉを添字として配列Ｔの要素を順番に参照する。ここで，注意しなければいけないのは，変数“合計”の初期値としてT(1)を代入している（“T(1) → 合計”）ことである。つまり，合計に加算するのはT(2)からとなるので，変数Ｉは2，3，4，…，10と変えていくことになる。したがって，変数Ｉの初期値は２となるので空欄ａは「2 → I」，変数Ｉが10より大きくなったら繰返しを終了するので空欄ｂの終了条件は「I ＞ 10」，合計に加算するのは変数Ｉを添字とした配列要素となるので空欄ｃは「合計 ＋ T(I) → 合計」となる。

３．１　アルゴリズムとプログラミング(4)　探索アルゴリズム

問１　【解答ア】

この流れ図は，順次探索の流れ図である。順次探索は，配列中の先頭要素から順番に，目的のデータと比較していく探索である。また，問題の流れ図は，条件”A(I) ≠ x”がNo（A(I) ＝ x）になるとＩを出力して処理を終了している。つまり，配列中の先頭要素（A(1)）から目的のデータｘと比較を開始し，すぐに条件“A(I) ≠ x”がNo（A(1)＝x）になるので，その時点で繰返し処理を抜けて変数Ｉの値である「１」を出力して処理を終了する。

問２　【解答エ】

ア：２分探索は，１回の比較で探索範囲を半分にせばめることができるので，順次探索よりも効率が良い探索アルゴリズムである。

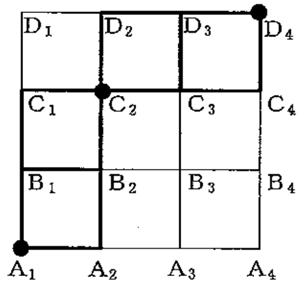
イ：番兵法は，順次探索を効率良く行うために考えられた方法である。

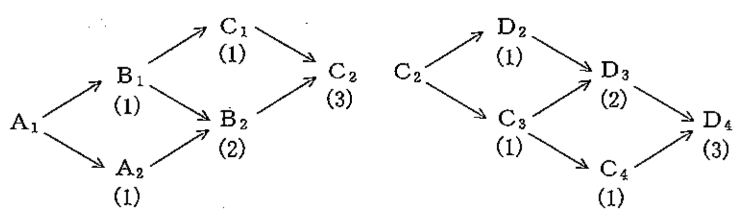
ウ：２分探索は，配列に格納されているデータの中央の要素と，目的のデータを比較していく探索アルゴリズムである。

エ：２分探索は，配列中に昇順（小さい順）または降順（大きい順）に格納されているデータに対して利用できる探索アルゴリズムである。探索対象範囲の中央にある要素（データ）と目的のデータの大小関係によって探索範囲を半分にせばめるため，探索対象となるデータが一定の順序に並んでいないと，利用することができない。（正解）

問３　【解答イ】

A_1地点からC_2地点への最短経路と同様に考えると，C_2地点からD_4地点へ行く最短経路も図のように数えることによって３通りあることがわかる。したがって，A_1地点から，C_2地点を経由してD_4地点へ行く最短経路は，3×3＝「9」通りある。





問4 【解答ア】

この流れ図は，番兵法を利用した順次探索の流れ図である。番兵法は，探索する配列のデータの末尾要素の後に，番兵として目的のデータを格納する方法である。これにより，目的のデータが配列中になくても最後の番兵と必ず一致するため，繰返しの判定条件を一つ減らすことができる。ただし，繰返しを抜けた後には，目的のデータが配列中のデータと一致したのか，番兵と一致したのかを判定する必要がある。番兵（問題の場合はT(N+1)）と一致した場合は，目的のデータが配列中に存在しなかったことになるので“見つからない”を出力する。逆に，配列中のデータ（T(1)～T(N)）と一致して繰返しを抜けた場合は，目的のデータが配列中に存在したことになるので変数Iを探索結果として出力する。したがって，成立する（Yes）と変数Iを出力する条件は，繰返しを抜けるときに目的のデータと一致したT(I)がT(1)～T(N)かを判定するので「I ≦ N」となる。

3.1 アルゴリズムとプログラミング(5) 整列アルゴリズム(1)

問1 【解答ウ】

基本選択法は，「未整列のデータの中から最小値（または最大値）を見つけて（選択して）いく整列アルゴリズムである。」未整列のデータの中から，昇順に整列する場合は最小値，降順に整列する場合は最大値を選択していくことで，先頭要素から順番に確定していく方式である。

ア：基本交換法に関する説明である。
イ：クイックソートに関する説明である。
エ：基本挿入法に関する説明である。

問2 【解答ウ】

配列要素の変化を，次のように区切ってみると，1周するごとに整列済みの部分が増えていることがわかる。

初期状態	4	3	1	2
1周目	3	4	1	2
2周目	1	3	4	2
3周目	1	2	3	4

この処理は，1周目で“3”を，2周目で“1”を，3周目で“2”を，整列済みの部分の正しい位置に挿入することで実現されている。このような整列方法を「基本挿入法」という。

問3 【解答エ】

基本選択法は，未整列のデータの中から最小値（または最大値）を見つけて（選択して）いく整列アルゴリズムである。降順に整列する場合は，T(1)～T(n)の最大値をT(1)に，T(2)～T(n)の最大値をT(2)に，・・・，T(n−1)～T(n)の最大値をT(n−1)に求めていく。したがって，1～n−1と変化する変数Iを指標とするT(I)が最大値を求める要素，I+1～nと変化する変数Jを指標とするT(J)が最大値を探すために比較する要素となる。最大値は，すべての要素と比較して最も大きい要素のことなので，条件「T(I) ＜ T(J)」がYes（真）のとき，比較する要素T(J)を新しい最大値の候補としてT(I)と交換すればよい。

問４　【解答イ】

問題の整列手順（アルゴリズム）は，基本交換法である。手順に従ってトレースをすると，次のようになる。

１回目	｛$\underline{5}$，$\underline{4}$，１，３，６，２｝	：	５＞４なので，要素を交換する。
	｛４，$\underline{5}$，$\underline{1}$，３，６，２｝	：	５＞１なので，要素を交換する。
	｛４，１，$\underline{5}$，$\underline{3}$，６，２｝	：	５＞３なので，要素を交換する。
	｛４，１，３，$\underline{5}$，$\underline{6}$，２｝	：	５＜６なので，要素は交換しない。
	｛４，１，３，５，$\underline{6}$，$\underline{2}$｝	：	６＞２なので，要素を交換する。
	｛４，１，３，５，２，６｝	：	１回目のパス終了
２回目	｛$\underline{4}$，$\underline{1}$，３，５，２，６｝	：	４＞１なので，要素を交換する。
	｛１，$\underline{4}$，$\underline{3}$，５，２，６｝	：	４＞３なので，要素を交換する。
	｛１，３，$\underline{4}$，$\underline{5}$，２，６｝	：	４＜５なので，要素は交換しない。
	｛１，３，４，$\underline{5}$，$\underline{2}$，６｝	：	５＞２なので，要素を交換する。
	｛１，３，４，２，$\underline{5}$，$\underline{6}$｝	：	５＜６なので，要素は交換しない。
	「｛１，３，４，２，５，６｝」	：	２回目のパス終了

３．１　アルゴリズムとプログラミング(6)　　整列アルゴリズム(2)

問１　【解答ア】

クイックソートは，未整列のデータ列の中から，基準となるデータ（基準値）を一つ選び，残りのデータ列を基準値より小さい値のグループと大きい値のグループに分割する。この処理を，グループのデータが一つになるまで，小さい値のグループと大きい値のグループで繰り返す整列方法である。

イ：基本選択法に関する説明である。

ウ：基本挿入法に関する説明である。

エ：基本交換法に関する説明である。

問２　【解答エ】

図では，隣り合った二つのデータまたは二組のデータ列を一つにまとめる処理を繰り返している。これは，二つの整列されたデータ列を，そのデータ列と同じ整列順で一つにまとめる併合（マージ）を繰り返す「マージソート」の整列手順である。なお，図では，マージソート前半の分割操作は省略されている。

問３　【解答ア】

・再帰

：自分自身を呼び出して同じ処理を繰り返すことである。クイックソートやマージソートは，再帰を利用した高速な整列アルゴリズムである。（正解）

・整列

：配列などに格納されているデータを，昇順（小さい順）または降順（大きい順）に並べ替える処理である。

・探索

：配列などに格納されているデータから，目的のデータを探す処理である。

・併合

：二つの整列されたデータ列を，そのデータ列と同じ整列順で一つにまとめる処理である。

問4　【解答ウ】

ア：クイックソートは，整列前のデータの状態によって，すべての未整列データが一方のグループに偏る分割が繰り返されると処理効率が少し悪くなる。ただし，少なくとも1回の分割で基準値として選んだデータの位置は決定するので，最小値（または最大値）を一つずつ決めていく基本選択法よりも処理効率が大幅に悪くなることはない。

イ：クイックソートは，昇順・降順のどちらにも対応できる整列アルゴリズムである。

ウ：マージソートは，データを分割して併合する処理を繰り返す整列アルゴリズムである。このとき，併合を行う回数はデータの状態ではなくデータ数によって決まる（例えば，元のデータ数が8件なら，併合は4+2+1=7回行われる）。したがって，マージソートによる整列の処理時間はデータ数によってほぼ決まることとなる。（正解）

エ：マージソートは，昇順・降順のどちらにも対応できる整列アルゴリズムである。

問5　【解答イ】

流れ図をトレースしていくと，次のようになる。

(1) F(775, 527)【x=775, y=527】
- ①　条件“y = 0”を判定する。
  - →　y=527なので条件は成立しない。(Noに分岐する)
- ②　処理“x % y → W”　：x % y = 775 % 527 = 248を変数Wに代入する。
- ③　処理“F(y, W) → R”　：y=527, W=248で関数Fを呼び出す（再帰)。

(2) F(527, 248)【x=527, y=248】
- ①　条件“y = 0”を判定する。
  - →　y=248なので条件は成立しない。(Noに分岐する)
- ②　処理“x % y → W”　：x % y = 527 % 248 = 31を変数Wに代入する。
- ③　処理“F(y, W) → R”　：y=248, W=31で関数Fを呼び出す（再帰)。

(3) F(248, 31)【x=248, y=31】
- ①　条件“y = 0”を判定する。
  - →　y=31なので条件は成立しない。(Noに分岐する)
- ②　処理“x % y → W”　：x % y = 248 % 31 = 0を変数Wに代入する。
- ③　処理“F(y, W) → R”　：y=31, W=0で関数Fを呼び出す（再帰)。

(4) F(31, 0)【x=31, y=0】
- ①　条件“y = 0”を判定する。
  - →　y=0なので条件は成立する。(Yesに分岐する)
- ②　処理“xを返却する”　：x=31を(3)③に返却する。

(3) 続き
- ③　処理“F(y, W) → R”　：返却された31を変数Rに代入する。
- ④　処理“Rを返却する”　：R=31を(2)③に返却する。

(2) 続き
- ③　処理“F(y, W) → R”　：返却された31を変数Rに代入する。
- ④　処理“Rを返却する”　：R=31を(1)③に返却する。

(1) 続き
- ③　処理“F(y, W) → R”　：返却された31を変数Rに代入する。
- ④　処理“Rを返却する”　：R=31を返却する。

したがって，最終的に返却される値は「31」である。