問8　ヒープの性質を利用したデータの整列（データ構造及びアルゴリズム）　(H30春-FE 午後問8)

【解答】

［設問1］　a－イ，b－エ

［設問2］　c－エ，d－イ，e－イ

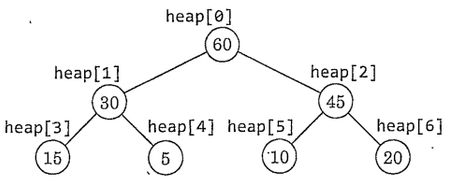
【解説】

ヒープの性質を利用して，データを昇順に整列するアルゴリズムの問題である。ヒープは，データ構造の中の木構造の一つで，2分木で親子の間に決まった大小関係のある構造をもつ。木構造は，ある要素と他の要素に親子の関係を与え，一つの要素が複数の要素に枝分かれした（木を逆さまにした）ような構造である。

用語を確認しておくと，図1に示されているように，○は「節」，最も上にある節を「根」，上位にある節を「親」，下位にある節を「子」と呼ぶ。また，子をもたない要素を「葉」という。そして，節のもつ子の要素が多くても2個までの木構造を2分木という。更に，どの節でも決まった親子の大小関係が成立する2分木をヒープという。

この問題では，ヒープの性質の理解と，1次元配列でヒープの構造を考える点，左側の子・右側の子・親の配列要素を求めるそれぞれの関数を利用して，要素番号を扱っているのか，要素の値を扱っているのか，を意識して考えていく必要がある。また，ヒープの性質は，「親の値は子の値よりも常に大きいか等しい」という大小関係が成立することである。

図1と図2を見比べて，図1に対応する配列heapの要素を記入すると図Aとなる。



図A　ヒープの例と配列heapの対応

親の要素番号と子の要素番号を関係付ける三つの関数は，heap[1]を例にすると，次のようになる。

lchild(1)	要素の値が30の左側の子の値15の要素番号3を返却する。
rchild(1)	要素の値が30の右側の子の値5の要素番号4を返却する。
parent(1)	要素の値が30の親の値60の要素番号0を返却する。

［設問1］

プログラムの空欄を埋める問題である。〔プログラム1〕を確認していく。

(行番号)

```
1  ○副プログラム: makeHeap( 整数型: data[], 整数型: heap[], 整数型: hnum )
2  ○整数型 : i, k
3  ■ i: 0, i < hnum, 1
4  |  ・heap[i] ← data[i]            /* heapにデータを追加 */
5  |  ・k ← i
6  |  ■ k > 0
7  |  |    a
8  |  |    ・swap(heap, k, [ b ] )
9  |  |    ・k ← parent(k)
10 |  |    ・break                  /* 内側の繰返し処理から抜ける */
11 |  |
12 |  ■
13 ■
14 ○副プログラム: swap( 整数型 : heap[], 整数型 : i, 整数型 : j )
15 ○整数型 : tmp
16 ・tmp ← heap[i]
17 ・heap[i] ← heap[j]
18 ・heap[j] ← tmp
```

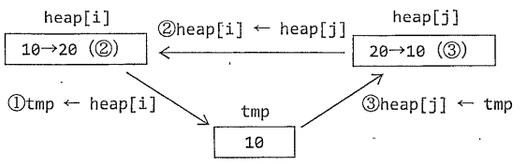
行番号3～13を外ループ，行番号6～12を内ループとする。

・行番号3～5：外ループでは，iを0からhnumより小さい間（hnum－1まで），1ずつ増やしながら配列dataから配列heapに一つずつ要素を格納する（hnumはデータの個数）。このとき，heap[]に追加した要素をヒープの性質を満たすよう交換していくが，交換対象を示す要素番号をkで表す。行番号5でiをkに設定し，初期値とする。

・行番号6～12：kが0より大きい間，すなわち根以外の節を処理する場合にループを繰り返す。なお，ループ途中で［ a ］の条件が成立せず，行番号10のbreakを実行したときも内ループを抜ける。行番号7～9は追加した要素と親の要素を比較し，ヒープの性質を満たさないとき（子の値が親の値より大きいとき），値を交換する処理を行っていると考えられる。

・行番号14～18：副プログラムswapは配列要素heap[i]とheap[j]を交換するプログラムである。そのままheap[j]をheap[i]に格納すると，heap[i]の値が失われるため，いったん，heap[i]をtmpに退避させた後，tmpからheap[j]に格納するという典型的な交換処理である。

heap[i]の要素を10，heap[j]の要素を20とした場合の副プログラムswapの処理を図Bに示す。



図B　副プログラムswapの処理

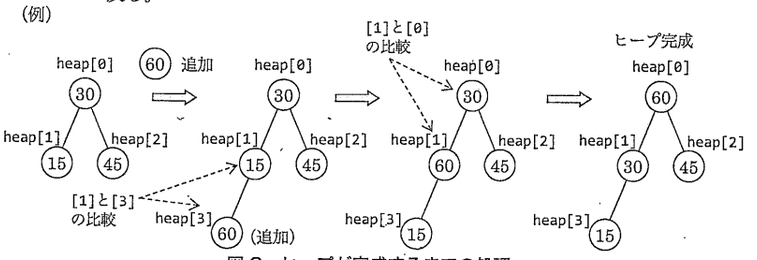
・空欄a：問題文の冒頭に「親の値は子の値よりも常に大きいか等しいという性質をもつ」とあるため，最後に追加した要素の値（子の値）とその親の要素の値を比較し，子の方が大きければ入れ替える処理を行えばよい。追加した要素の値はheap[k]，要素番号kの親の要素番号は関数parent(k)で求めることができるので，親の要素の値はheap[parent(k)]である。したがって，空欄aには（イ）の「heap[k] ＞ heap[parent(k)]」が入る。

・空欄b：副プログラムswapに渡す引数の設定である。配列heapに追加した要素番号kと，kの親の要素番号を渡せばよいので，空欄bには（エ）の「parent(k)」が入る。

・行番号7～10：heap[k] ＞ heap[parent(k)]の場合は，副プログラムswapでheap[k]と親の要素の値heap[parent(k)]を入れ替え，追加したデータは要素番号parent(k)の位置に移されたので，次は，このparent(k)の値をkに設定し，更に上位の親の要素と比較するため内ループの先頭の行番号6に戻る。追加したデータが親の要素より大きい場合は入れ替え，根の一つ下の階層に至るまでこの処理を繰り返す。

heap[k]がheap[parent(k)]より小さいか等しい場合は，ヒープの性質を満たしているため，行番号10のbreakで内ループを抜け，外ループに戻る。

(例)



図C　ヒープが完成するまでの処理

参考までに，hnumが7，配列dataが次の値のときのmakeHeapの処理をトレースしておく（網掛け部分は行番号7で比較する要素heap[k]とparent(k)を示す）。結果は図Aのヒープになる。

要素番号	0	1	2	3	4	5	6
配列 data	15	30	10	45	5	20	60

i	k	parent(k)	配列 heap [0]	[1]	[2]	[3]	[4]	[5]	[6]	説明
0	0		15							k ＞ 0を満足しない。内ループを抜ける。i ← i+1
1	1	0	15	30						heap[1] ＞ heap[0] 入替え。k ← parent(1)=0
↓	0		30	15						k ＞ 0を満足しない。内ループを抜ける。i ← i+1
2	2	0	30	15	10					heap[2] ≦ heap[0] で入替えなし。内ループを抜ける。i ← i+1
3	3	1	30	15	10	45				heap[3] ＞ heap[1] 入替え。k ← parent(3)=1
↓	1	0	30	45	10	15				heap[1] ＞ heap[0] 入替え。k ← parent(1)=0
↓	0		45	30	10	15				k ＞ 0を満足しない。内ループを抜ける。i ← i+1
4	4	1	45	30	10	15	5			heap[4] ≦ heap[1] で入替えなし。内ループを抜ける。i ← i+1
5	5	2	45	30	10	15	5	20		heap[5] ＞ heap[2] 入替え。k ← parent(5)=2
↓	2	0	45	30	20	15	5	10		heap[2] ≦ heap[0] で入替えなし。内ループを抜ける。i ← i+1
6	6	2	45	30	20	15	5	10	60	heap[6] ＞ heap[2] 入替え。k ← parent(6)=2
↓	2	0	45	30	60	15	5	10	20	heap[2] ＞ heap[0] 入替え。k ← parent(2)=0
↓	0		60	30	45	15	5	10	20	k ＞ 0を満足しない。内ループを抜ける。i ← i+1
7			60	30	45	15	5	10	20	i ＜ hnum=7を満足しない。外ループを抜けて終了。（図Aのヒープを実現）

［設問2］

この問題におけるヒープの性質である最大値が根(heap[0])であることを利用して，配列要素を整列するヒープソートの問題である。

〔プログラム2〕の行番号1～7は〔プログラム2の説明〕(2)に記述されている処理手順に対応する部分である。

(処理手順)

① 全データでヒープを構成

② ヒープで最大値であるheap[0]の値とheap[hnum－1]の値を交換

③ 要素数を1減らしたheap[0]からheap[hnum－2]でヒープを再構成

④ ②のhnum－1をhnum－2，③のhnum－2をhnum－3と1ずつ減じながら要素数が1になるまで②，③を繰り返す。

整列対象領域の最後の要素番号をlastで示し，hnum－1から1まで1ずつ減じながら処理を繰り返す。この処理で，整列済みデータ領域には要素番号hnum－1の最大値から順に要素番号0に向かって降順に要素の値が格納されることになる。そして，左側の整列対象領域に対して，副プログラムdownHeapでヒープの性質を満たすように再構成する。

設問2は，プログラムの穴埋めでなく，プログラムの動作を問う内容なので，まず〔プログラム2の動作〕を読み，具体的なデータを当てはめた配列heapを見ながら

問8　ヒープの性質を利用したデータの整列（データ構造及びアルゴリズム）　(H30春-FE 午後問8)

【解答】

［設問1］　a－イ，b－エ

［設問2］　c－エ，d－イ，e－イ

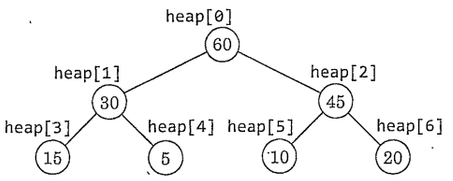
【解説】

ヒープの性質を利用して，データを昇順に整列するアルゴリズムの問題である。ヒープは，データ構造の中の木構造の一つで，2分木で親子の間に決まった大小関係のある構造をもつ。木構造は，ある要素と他の要素に親子の関係を与え，一つの要素が複数の要素に枝分かれした（木を逆さまにした）ような構造である。

用語を確認しておくと，図1に示されているように，○は「節」，最も上にある節を「根」，上位にある節を「親」，下位にある節を「子」と呼ぶ。また，子をもたない要素を「葉」という。そして，節のもつ子の要素が多くても2個までの木構造を2分木という。更に，どの節でも決まった親子の大小関係が成立する2分木をヒープという。

この問題では，ヒープの性質の理解と，1次元配列でヒープの構造を考える点，左側の子・右側の子・親の配列要素を求めるそれぞれの関数を利用して，要素番号を扱っているのか，要素の値を扱っているのか，を意識して考えていく必要がある。また，ヒープの性質は，「親の値は子の値よりも常に大きいか等しい」という大小関係が成立することである。

図1と図2を見比べて，図1に対応する配列heapの要素を記入すると図Aとなる。



図A　ヒープの例と配列heapの対応

親の要素番号と子の要素番号を関係付ける三つの関数は，heap[1]を例にすると，次のようになる。

lchild(1)	要素の値が30の左側の子の値15の要素番号3を返却する。
rchild(1)	要素の値が30の右側の子の値5の要素番号4を返却する。
parent(1)	要素の値が30の親の値60の要素番号0を返却する。

［設問1］

プログラムの空欄を埋める問題である。〔プログラム1〕を確認していく。

(行番号)

```
1  ○副プログラム: makeHeap( 整数型: data[], 整数型: heap[], 整数型: hnum )
2  ○整数型 : i, k
3  ■ i: 0, i < hnum, 1
4  |  ・heap[i] ← data[i]            /* heapにデータを追加 */
5  |  ・k ← i
6  |  ■ k > 0
7  |  |    a
8  |  |    ・swap(heap, k, [ b ] )
9  |  |    ・k ← parent(k)
10 |  |    ・break                  /* 内側の繰返し処理から抜ける */
11 |  |
12 |  ■
13 ■
14 ○副プログラム: swap( 整数型 : heap[], 整数型 : i, 整数型 : j )
15 ○整数型 : tmp
16 ・tmp ← heap[i]
17 ・heap[i] ← heap[j]
18 ・heap[j] ← tmp
```

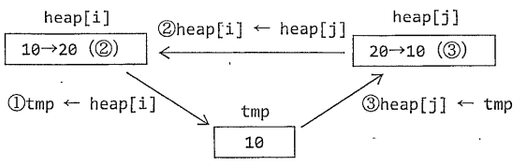
行番号3～13を外ループ，行番号6～12を内ループとする。

・行番号3～5：外ループでは，iを0からhnumより小さい間（hnum－1まで），1ずつ増やしながら配列dataから配列heapに一つずつ要素を格納する（hnumはデータの個数）。このとき，heap[]に追加した要素をヒープの性質を満たすよう交換していくが，交換対象を示す要素番号をkで表す。行番号5でiをkに設定し，初期値とする。

・行番号6～12：kが0より大きい間，すなわち根以外の節を処理する場合にループを繰り返す。なお，ループ途中で［ a ］の条件が成立せず，行番号10のbreakを実行したときも内ループを抜ける。行番号7～9は追加した要素と親の要素を比較し，ヒープの性質を満たさないとき（子の値が親の値より大きいとき），値を交換する処理を行っていると考えられる。

・行番号14～18：副プログラムswapは配列要素heap[i]とheap[j]を交換するプログラムである。そのままheap[j]をheap[i]に格納すると，heap[i]の値が失われるため，いったん，heap[i]をtmpに退避させた後，tmpからheap[j]に格納するという典型的な交換処理である。

heap[i]の要素を10，heap[j]の要素を20とした場合の副プログラムswapの処理を図Bに示す。



図B　副プログラムswapの処理

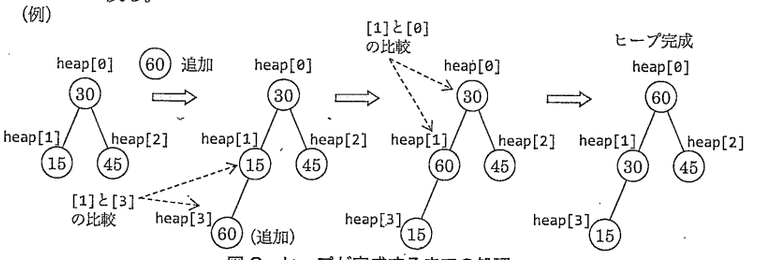
・空欄a：問題文の冒頭に「親の値は子の値よりも常に大きいか等しいという性質をもつ」とあるため，最後に追加した要素の値（子の値）とその親の要素の値を比較し，子の方が大きければ入れ替える処理を行えばよい。追加した要素の値はheap[k]，要素番号kの親の要素番号は関数parent(k)で求めることができるので，親の要素の値はheap[parent(k)]である。したがって，空欄aには（イ）の「heap[k] ＞ heap[parent(k)]」が入る。

・空欄b：副プログラムswapに渡す引数の設定である。配列heapに追加した要素番号kと，kの親の要素番号を渡せばよいので，空欄bには（エ）の「parent(k)」が入る。

・行番号7～10：heap[k] ＞ heap[parent(k)]の場合は，副プログラムswapでheap[k]と親の要素の値heap[parent(k)]を入れ替え，追加したデータは要素番号parent(k)の位置に移されたので，次は，このparent(k)の値をkに設定し，更に上位の親の要素と比較するため内ループの先頭の行番号6に戻る。追加したデータが親の要素より大きい場合は入れ替え，根の一つ下の階層に至るまでこの処理を繰り返す。

heap[k]がheap[parent(k)]より小さいか等しい場合は，ヒープの性質を満たしているため，行番号10のbreakで内ループを抜け，外ループに戻る。

(例)



図C　ヒープが完成するまでの処理

参考までに，hnumが7，配列dataが次の値のときのmakeHeapの処理をトレースしておく（網掛け部分は行番号7で比較する要素heap[k]とparent(k)を示す）。結果は図Aのヒープになる。

要素番号	0	1	2	3	4	5	6
配列 data	15	30	10	45	5	20	60

i	k	parent(k)	配列 heap [0]	[1]	[2]	[3]	[4]	[5]	[6]	説明
0	0		15							k ＞ 0を満足しない。内ループを抜ける。i ← i+1
1	1	0	15	30						heap[1] ＞ heap[0] 入替え。k ← parent(1)=0
↓	0		30	15						k ＞ 0を満足しない。内ループを抜ける。i ← i+1
2	2	0	30	15	10					heap[2] ≦ heap[0] で入替えなし。内ループを抜ける。i ← i+1
3	3	1	30	15	10	45				heap[3] ＞ heap[1] 入替え。k ← parent(3)=1
↓	1	0	30	45	10	15				heap[1] ＞ heap[0] 入替え。k ← parent(1)=0
↓	0		45	30	10	15				k ＞ 0を満足しない。内ループを抜ける。i ← i+1
4	4	1	45	30	10	15	5			heap[4] ≦ heap[1] で入替えなし。内ループを抜ける。i ← i+1
5	5	2	45	30	10	15	5	20		heap[5] ＞ heap[2] 入替え。k ← parent(5)=2
↓	2	0	45	30	20	15	5	10		heap[2] ≦ heap[0] で入替えなし。内ループを抜ける。i ← i+1
6	6	2	45	30	20	15	5	10	60	heap[6] ＞ heap[2] 入替え。k ← parent(6)=2
↓	2	0	45	30	60	15	5	10	20	heap[2] ＞ heap[0] 入替え。k ← parent(2)=0
↓	0		60	30	45	15	5	10	20	k ＞ 0を満足しない。内ループを抜ける。i ← i+1
7			60	30	45	15	5	10	20	i ＜ hnum=7を満足しない。外ループを抜けて終了。（図Aのヒープを実現）

［設問2］

この問題におけるヒープの性質である最大値が根(heap[0])であることを利用して，配列要素を整列するヒープソートの問題である。

〔プログラム2〕の行番号1～7は〔プログラム2の説明〕(2)に記述されている処理手順に対応する部分である。

(処理手順)

① 全データでヒープを構成

② ヒープで最大値であるheap[0]の値とheap[hnum－1]の値を交換

③ 要素数を1減らしたheap[0]からheap[hnum－2]でヒープを再構成

④ ②のhnum－1をhnum－2，③のhnum－2をhnum－3と1ずつ減じながら要素数が1になるまで②，③を繰り返す。

整列対象領域の最後の要素番号をlastで示し，hnum－1から1まで1ずつ減じながら処理を繰り返す。この処理で，整列済みデータ領域には要素番号hnum－1の最大値から順に要素番号0に向かって降順に要素の値が格納されることになる。そして，左側の整列対象領域に対して，副プログラムdownHeapでヒープの性質を満たすように再構成する。

設問2は，プログラムの穴埋めでなく，プログラムの動作を問う内容なので，まず〔プログラム2の動作〕を読み，具体的なデータを当てはめた配列heapを見ながら