問題３　次のヒープに関する記述を読み，各設問に答えよ。

［ヒープについて］

二分木構造において，親の値が子の値より常に大きいか等しい，または小さいか等しいという制約を満たすものをヒープと呼ぶ。この問題では，親の値が子の値より常に大きいか等しいものを扱う。

図１にヒープの例を示す。図は「○」で節を表しており，「○」の中の数値が節の値である。また，子を持つ節を親と呼び，親を持たない節を根と呼ぶ。ヒープの制約に従えば，根の値は節の中で一番大きな値となる。なお，子の節が１つの場合は左の子とし，子の節が２つある場合の並びは値と関係ない。

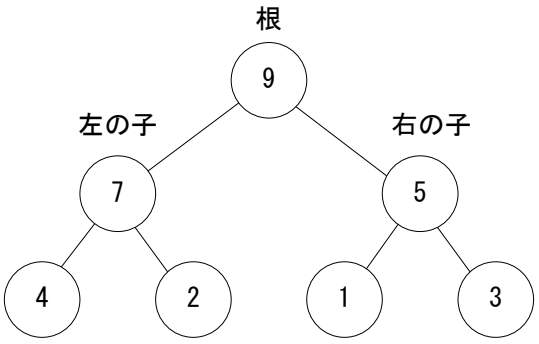


図１　ヒープの例

［ヒープの実装について］

・ヒープ構造を一次元配列 h に構築する。
・配列の添え字は 0 から始まるものとする。
・根は h[0]に格納する。
・親の節の位置を i とすれば，左の子が格納される位置は i×2＋1，右の子が格納される位置は i×2＋2 となる。

次の図２は，図１を配列 h に格納したものである。

位置	0	1	2	3	4	5	6
h	9	7	5	4	2	1	3

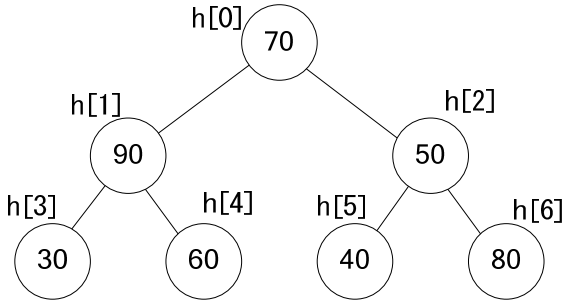
図２　ヒープを一次元配列に格納した例

・根である h[0]の左の子は h[1]，右の子は h[2]に格納される。
・h[1]の左の子は h[3]，右の子は h[4]に格納される。
・h[2]の左の子は h[5]，右の子は h[6]に格納される。

＜設問１＞　次のヒープの構築に関する記述中［　　］に入れるべき適切な字句を解答群から選べ。

ランダムな値が格納された配列でヒープを構築するには，「親と子要素の中での最大値が親の位置に格納されるようにデータを入れ替える」という構築操作を繰り返す。この時，データの入れ替えが発生した場合は，入れ替えた子の位置を新たな親の位置として構築操作を続け，入れ替えが発生しなければ構築操作を終える。

ここで，ランダムな値を格納した二分木が図３のような場合において，ヒープを構築することを考える。



位置	0	1	2	3	4	5	6
h	70	90	50	30	60	40	80

図３　ヒープが構築されていない二分木と配列の状態

ヒープの構築を開始する位置は，末端の親(位置が一番大きな場所にある親)から始め，配列の前方(位置の小さい方)の親へと進める。図３では，末端の親はh[2]である。これは，「(配列の要素数－2)÷2」で計算できる。なお，除算の結果は小数点以下を切り捨てた整数値にする。

最初に，h[2]と h[2]を親とした子要素の中での最大値を h[2]に格納する。この場合は，h[2]と［　(1)　］を入れ替える。さらに［　(1)　］を新たな親の位置として構築操作を進めようとすれば，新しい子の位置は配列の要素を超えた位置になるため，構築操作を終える。

次に，h[1]と h[1]を親とした子要素の中での最大値を h[1]に格納する。ここで，h[1]に格納されている値は子要素の値と比較しても一番大きな値である。よって，入れ替えは行わないため，構築操作を終える。

最後に，h[0]を親とした子要素の中での最大値をh[0]に格納する。この場合は，h[0]と［　(2)　］を入れ替える。さらに［　(2)　］を新しい親の位置として構築操作を続ける。新しい親の位置に格納されている値が子要素の値と比較して一番大きな値となる。よって，入れ替えは行わないため，構築操作を終了する。

以上の操作により構築したヒープは図４になる。

位置	0	1	2	3	4	5	6
h	90	70	80	30	60	40	50

図４　ヒープを構築した状態

(1)，(2) の解答群

ア．h[1]　　イ．h[2]　　ウ．h[3]
エ．h[4]　　オ．h[5]　　カ．h[6]

＜設問２＞　配列に構築されたヒープとして正しいものを(3)の解答群から選べ。

(3) の解答群

ア．	700	200	300	400	500	600	100
イ．	600	500	700	400	300	200	100
ウ．	700	500	600	300	400	100	200
エ．	700	500	400	600	100	200	300

＜設問３＞　次の流れ図の説明を読み，流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。

［流れ図の説明］

要素数 n の一次元配列 h[i](i＝0,1,…,n－1)に格納されたランダムな値に対してヒープを構築する。流れ図に示したものは，ヒープの構築を制御する makeHeap，ヒープを構築する処理を行う downHeap，配列内の値を入れ替える swap である。

・makeHeap は，ヒープを構築するための操作を行う donwHeap を呼び出す。この時，親の要素位置と配列内の最後の位置を引数で与える。
・downHeap は，配列内の値を入れ替える swap を呼び出す。この時，入れ替える配列内の２つの位置を引数で与える。

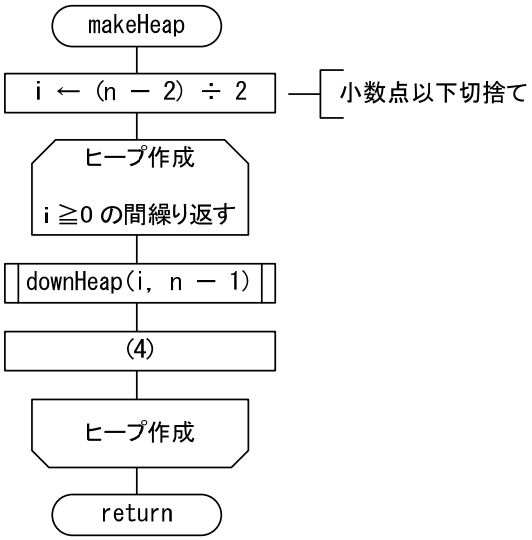


図５　makeHeap の流れ図

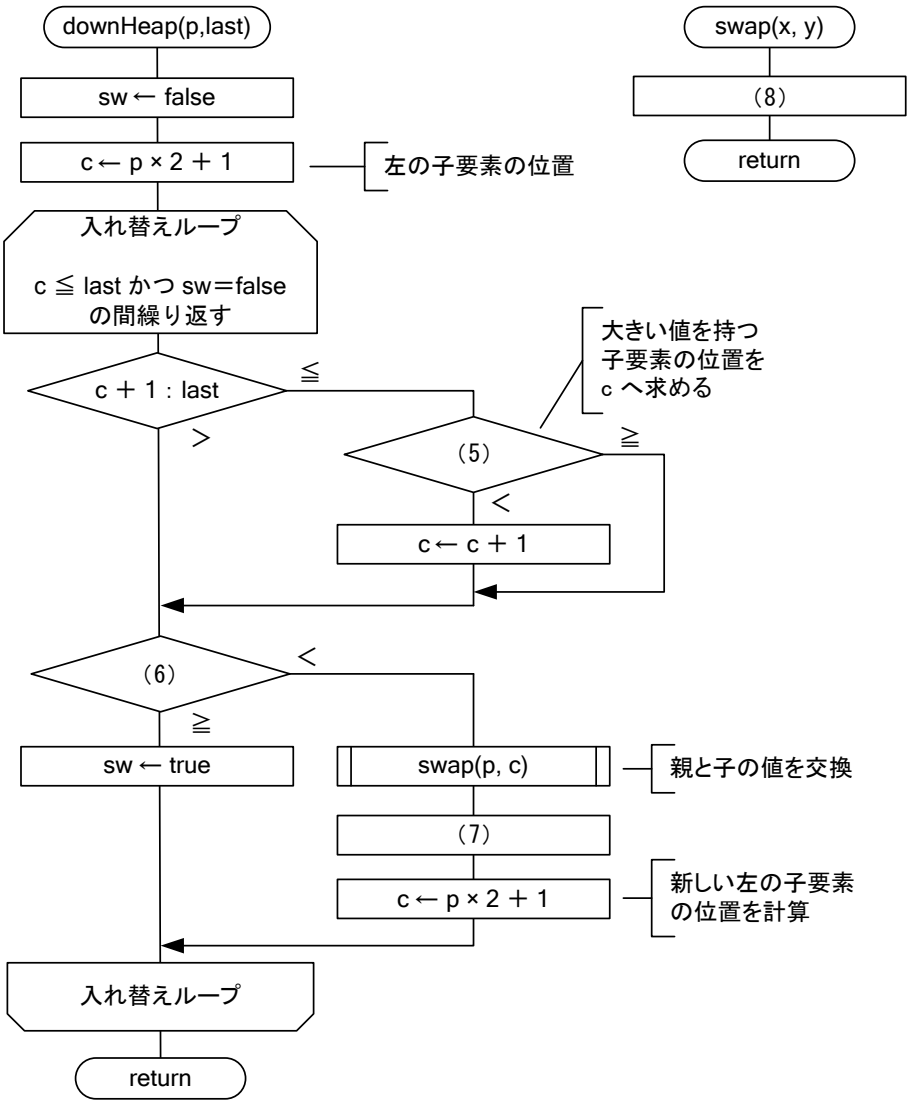


図６　downHeap と swap の流れ図

(4) の解答群

ア．i ← i － 1　　イ．i ← i ＋ 1
ウ．i ← i × 2　　エ．i ← i ＋ 2

(5)，(6) の解答群

ア．h[c]：h[c＋1]　　イ．h[c＋1]：h[c]
ウ．h[c]：h[p]　　エ．h[c＋1]：h[p]
オ．h[p]：h[c]　　カ．h[p＋1]：h[c]

(7) の解答群

ア．p ← c　　イ．p ← c ＋ 1
ウ．p ← p ＋ 1　　エ．p ← p × 2

(8) の解答群

ア．
```
h[x] ← h[y]
h[y] ← h[x]
```

イ．
```
h[x] ← h[x＋1]
h[y] ← h[y＋1]
```

ウ．
```
work ← h[x]
h[y] ← work
h[x] ← h[y]
```

エ．
```
work ← h[x]
h[x] ← h[y]
h[y] ← work
```

＜設問４＞　次のヒープソートに関する記述を読み，流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。

ヒープが構築された配列では，配列の先頭に格納されるのは一番大きな値である。これを利用して配列内を並び替える方法がヒープソートである。

例えば，図７のヒープが構築されている配列で考える。

	0	1	2	3	4	5	6
h	90	70	80	30	60	40	50

図７　ヒープが構築されている配列

h[0]に格納されている値が最大値であり，配列の最後である h[6]と入れ替える。

	0	1	2	3	4	5	6
h	50	70	80	30	60	40	90

図８　h[0]と h[6]を入れ替える

要素を入れ替えたことによりヒープ構造が崩れるため，未整列の領域である h[0]～h[5]でヒープを再構築する。再構築では，h[0]からの構築操作を行う。

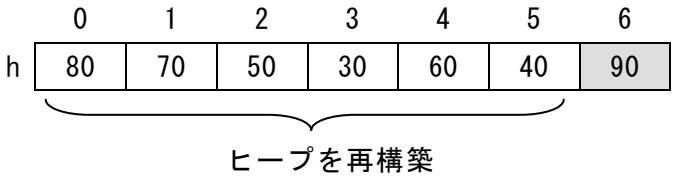


図９　h[0]～h[5]でヒープを再構築

再構築を終えると，h[0]～h[5]の最大値が h[0]に格納されるため，h[0]と h[5]を入れ替える。

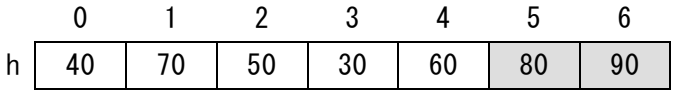


図 10　h[0]と h[5]でヒープを入れ替える

再びヒープ構造が崩れるため，未整列の領域である h[0]～h[4]でヒープを再構築する。再構築では，h[0]からの構築操作を行う。

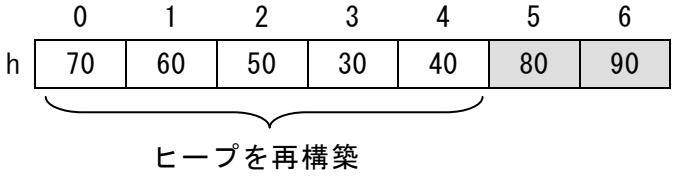


図 11　h[0]～h[4]でヒープを再構築

このように，配列の先頭と並べ替える対象範囲の末尾を入れ替え，対象範囲を１つ縮めてヒープを再構築するという操作を繰り返すことで，配列内のデータを昇順に並べ替えることができる。

次の流れ図は，ランダムに格納された配列 h[i](i＝0，1，…，n－1)をヒープソートにより並べ替えるものである。流れ図中で呼び出す makeHeap，downHeap，swap は，設問３の流れ図で用いたものである。

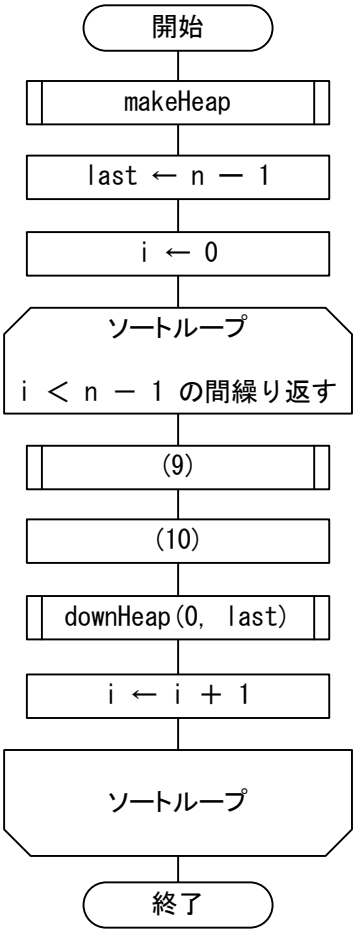


図 12　ヒープソートの流れ図

(9) の解答群

ア. swap(0, last)　　イ. swap(0, i)

ウ. swap(i, n)　　エ. swap(i, last)

(10) の解答群

ア. last ← last − 1　　イ. last ← last ＋ 1

ウ. last ← last − i　　エ. last ← last ＋ i