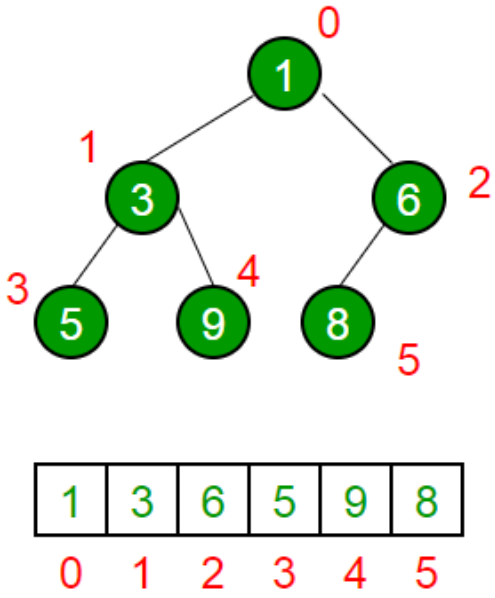
Elementi core dell'implementazione di uno heap binario (`binary_heap`)

La figura mostra come gli elementi di un array A possono essere visti come i nodi di un albero orientato, ordinato e binario.
-*Orientato* in quanto:

1. esiste un nodo *radice* (in figura, il nodo 0, di valore $A[0] = 1$),
2. ogni altro nodo ha un unico *padre* di cui è *figlio*,
3. partendo da un qualsiasi nodo terminiamo nella radice se ci muoviamo di padre in padre.

-*Binario* in quanto ciascun nodo ha al più due figli.
-*Ordinato* in quanto ogni padre ha un *figlio sinistro* (sempre) e un *figlio destro* (non sempre) che comunque distinguiamo. Ad esempio, del nodo 0, il nodo 1 è il figlio sinistro mentre il 2 è quello destro.

Si noti come, in generale, il padre del nodo i sia il nodo $\frac{i-1}{2}$, dove la divisione è arrotondata per difetto.



Un array A di n elementi è considerato un *binary heap* se soddisfa la seguente proprietà (*buon ordinamento dello heap*):

> Per nessuna coppia di nodi in relazione padre/figlio il valore del padre supera quello del figlio.
> In formula: $A\left[\frac{i-1}{2}\right] \leq A[i]$ per ogni $i \in \mathbb{N}$ con $0 < i < n$.

L'array $A = [1, 3, 6, 5, 9, 8]$ in figura è un binary heap benchè non sia totalmente ordinato ($A[2] > A[3]$ e $A[4] > A[5]$). Ciò nonostante, il buon ordinamento dello heap basta a garantire che $A[0]$ sia minimo tra tutti i valori contenuti in A. La domanda è quindi quanto ci costa mantenere il buon ordinamento quando i valori degli elementi in A vengono modificati.

Task

Purtroppo, un binary heap A potrebbe cessare di essere tale quando modifichiamo il valore anche di uno solo dei suoi elementi: il nuovo valore potrebbe infatti essere minore del valore del padre oppure superare quello di un figlio. Ti viene comunicato n e l'indice i dell'elemento modificato. Senza sapere null'altro di A devi riordinarne gli elementi in modo da ripristinare il buon ordinamento dello heap. Per fare questo puoi avvalerti di due primitive, da applicarsi ad un qualsiasi $j \in \mathbb{N}$ con $0 < j < n$:

query puoi chiedere se $A[j] < A\left[\frac{j-1}{2}\right]$, ossia indagare se l'elemento attualmente in posizione j non sia superato in valore dal proprio padre, in **violazione** della proprietà di buon ordinamento dello heap.

update puoi richiedere che l'elemento j e suo padre vengano scambiati in A.

Non sprecare queries e percorri soluzioni semplici. Non appena ti sarai assicurato di aver ripristinato un buon ordinamento chiudi con questo array A per eventualmente passare al prossimo. Potrai invece fare (e disfare) quante updates vorrai (pur di rimanere entro il time limit).

Protocollo di comunicazione

Il tuo programma interagirà col server leggendo dal proprio canale `stdin` e scrivendo sul proprio canale `stdout`. La prima riga di `stdin` contiene T, il numero di testcase da affrontare. All'inizio di ciascun testcase, trovi sulla prossima riga di `stdin` i tre numeri c, n e i separati da spazio, dove c specifica la categoria di subtask del testcase. Inizia ora un loop per uscire dal quale devi stampare su `stdout` la riga "`!0`" con la quale comunichi di voler chiudere col testcase corrente (se a quel punto A sarà uno heap ti verrà assegnato il punto del testcase). Ogni riga della forma "`!j`" con $j \in \mathbb{N}, 0 < j < n$ chiede al server di scambiare col padre in A l'elemento che attualmente si trova in posizione j. Ogni riga della forma "`?j`" con $j \in \mathbb{N}, 0 < j < n$ pone al server la domanda se $A[j] < A\left[\frac{j-1}{2}\right]$. (Dovrai a immediato seguire leggere la risposta da `stdin`: la riga "`y`" significa sì mentre "`n`" sta per no.)

Nota 1: Ogni tua comunicazione verso il server deve essere collocata su una diversa riga di `stdout` e ricordati di forzarne l'invio immediato al server effettuando un flush del tuo output!

Nota 2: Il tuo dialogo diretto col server (ossia ancora prima di aver scritto una sola riga di codice) può aiutarti ad acquisire il corretto protocollo di comunicazione tra il server e il tuo programma. Verrai così anche a meglio conoscere il problema e sarai più pronto a progettare come condurre la fase di codifica. Verrai anche a meglio conoscere come gioca il server, il quale adotta una strategia adattiva per metterti alla prova secondo il criterio del caso peggiore. Il file `results.txt` verrà comunque scaricato nel folder `output` alla fine dell'interazione col server se la termini con `Ctrl-D`.

Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 1 secondo.

I testcase sono raggruppati nei seguenti subtask.

1. **[6 pts← 6 istanze] `small0`:** $n < 16, c = 0$ (puoi assumere che il valore dell'elemento i è stato diminuito, puoi fare al più $\lfloor \log_2 n \rfloor = 3$ domande)
2. **[6 pts← 6 istanze] `small1`:** $n < 16, c = 1$ (puoi assumere che il valore dell'elemento i è stato aumentato, puoi fare al più $2\lfloor \log_2 n \rfloor = 6$ domande)
3. **[6 pts← 6 istanze] `small2`:** $n < 16, c = 2$ (a priori non sai se $A[i]$ è stato aumentato o diminuito, puoi fare al più $2\lfloor \log_2 n \rfloor = 6$ domande)
4. **[6 pts← 6 istanze] `medium0`:** $n < 1024, c = 0$ (puoi assumere che il valore dell'elemento i è stato diminuito, puoi fare al più $\lfloor \log_2 n \rfloor = 9$ domande)
5. **[6 pts← 6 istanze] `medium1`:** $n < 1024, c = 1$ (puoi assumere che il valore dell'elemento i è stato aumentato, puoi fare al più $2\lfloor \log_2 n \rfloor = 18$ domande)
6. **[6 pts← 6 istanze] `medium2`:** $n < 1024, c = 2$ (a priori non sai se $A[i]$ è stato aumentato o diminuito, puoi fare al più $2\lfloor \log_2 n \rfloor = 18$ domande)
7. **[6 pts← 6 istanze] `big0`:** $n < 2^{16}, c = 0$ (puoi assumere che il valore dell'elemento i è stato diminuito, puoi fare al più $\lfloor \log_2 n \rfloor = 15$ domande)
8. **[6 pts← 6 istanze] `big1`:** $n < 2^{16}, c = 1$ (puoi assumere che il valore dell'elemento i è stato aumentato, puoi fare al più $2\lfloor \log_2 n \rfloor = 30$ domande)
9. **[6 pts← 6 istanze] `big2`:** $n < 2^{16}, c = 2$ (a priori non sai se $A[i]$ è stato aumentato o diminuito, puoi fare al più $2\lfloor \log_2 n \rfloor = 30$ domande)

In generale, quando si richiede la valutazione di un subtask vengono valutati anche i subtask che li precedono, ma si evita di avventurarsi in subtask successivi fuori dalla portata del tuo programma che potrebbe andare in crash o comportare tempi lunghi per ottenere la valutazione completa della sottomissione. Ad esempio, chiamando[1, 2] :

[1] <URL> server esame: `wss://ta.di.univr.it/esame`

[2] <URL> server esercitazioni e simula-prove: `wss://ta.di.univr.it/algo`

```
rtal -s <URL>  connect -x <token> -a size=medium0
      binary_heap  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`small0`, `small1`, `small2`, `medium0`.

Il valore di default per l'argomento `size` è `big2` che include tutti i testcase.

Esempio

Le righe che iniziano con '?' o '!' sono quelle inviate dallo studente o da suo programma, le altre sono quelle inviate dal server. Inoltre: di ciascuna riga è di mero commento quella parte che comincia col primo carattere di cancelletto '#'.

```
5      # numero di testcase/istanze/partite
0 7 0  #   il server comunica il setting del primo testcase (c=0, n=7, i=0)
!0     # il problem solver chiude il testcase 1 (fà punto).
0 7 4  #   avvio testcase 2: (c=0, n=7, i=4)
!0     # testcase 2 chiuso ma nessun punto: nulla garantisce che A[4] >= A[1]
0 7 4  #   avvio testcase 3: (c=0, n=7, i=4)
?4     # il problem solver pone la query "A[4] < A[1]?"
n      # il server risponde di no
!0     # testcase 3 chiuso con punto: A è necessariamente uno heap.
0 7 4  #   avvio testcase 4: (c=0, n=7, i=4)
?4     # il problem solver pone la query "A[4] < A[1]?"
y      # il server risponde di sì
!4     # il problem solver richiede lo scambio tra A[4] e il padre A[1]
?1     # il problem solver pone la query "A[1] < A[0]?"
n      # il server risponde di no
!0     # testcase 4 chiuso con punto: di sicuro A è ora uno heap.
1 7 0  #   avvio testcase 5: (c=1, n=7, i=0)
?1     # il problem solver pone la query "A[1] < A[0]?"
n      # il server risponde di no
?2     # il problem solver pone la query "A[2] < A[0]?"
y      # il server risponde di sì
!2     # il problem solver richiede lo scambio tra A[2] e il padre A[0]
?5     # il problem solver pone la query "A[5] < A[2]?"
n      # il server risponde di no
?6     # il problem solver pone la query "A[6] < A[2]?"
n      # il server risponde di no
!0     # si chiude la comunicazione. Anche il testcase 5 ha fatto punto.
```