Ans 1 :-

a) myfunc.

b) address of the function.

c) typedef int ftype(string a, string b);

Here ftype is the newly created type name.

d) ftype*[] var
=alloc_array(ftype*, 10);

Explanation :-
Syntax of allocating an array of type T is:

T[] var = alloc_array(T, size)

Our function type is ftype
Its pointer type is ftype*
∴ we need T = ftype*

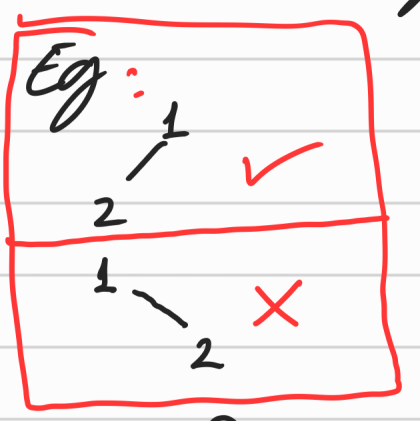
Ans 2 outline:-

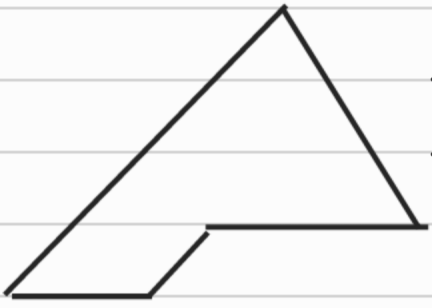
sort (A, n) {

1) create a min-heap H.
2) insert all elements of A in H one by one.
3) Extract elements from H. Store them in order in A, overwriting the old entries.

}

Ans 3: 1) Shape invariant





check the lectures for details.

2) Ordering invariant.

for any node (except the root) its parent has equal or higher priority.

Ans 4:

1 has to be the top element.

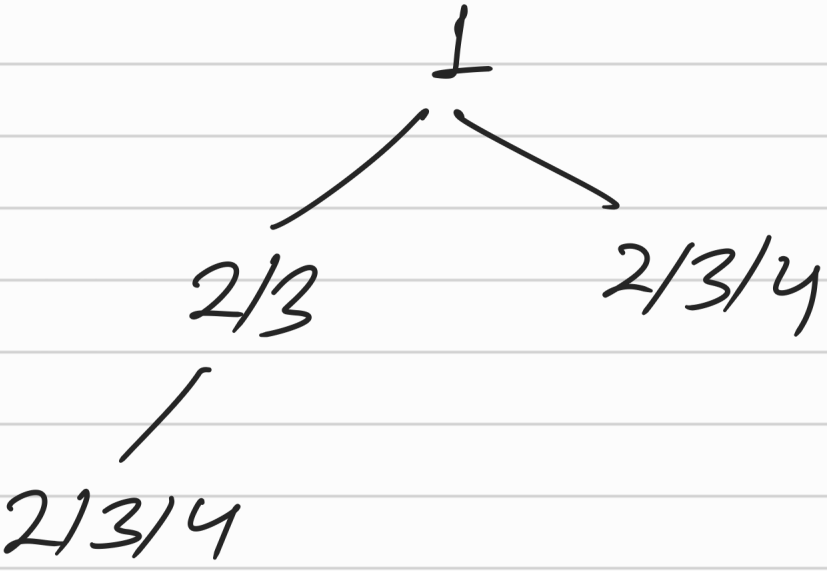
and the shape is 

We solve this in two different ways.

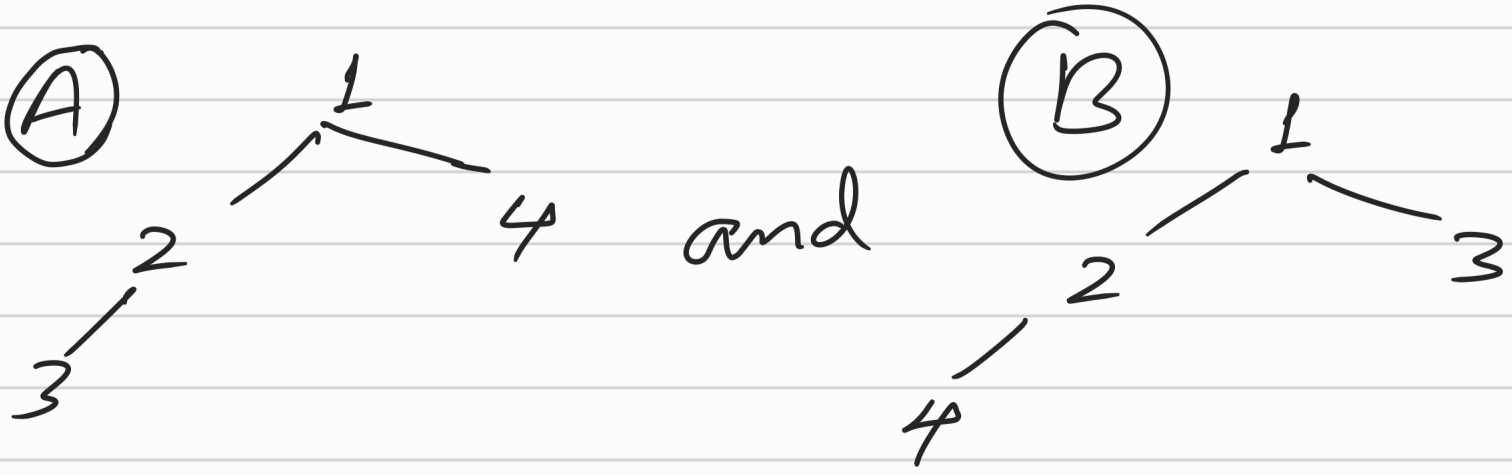
Soln 1

1) 1 is the top element
2) 4 cannot be the left child of 1, otherwise ordering will be violated.

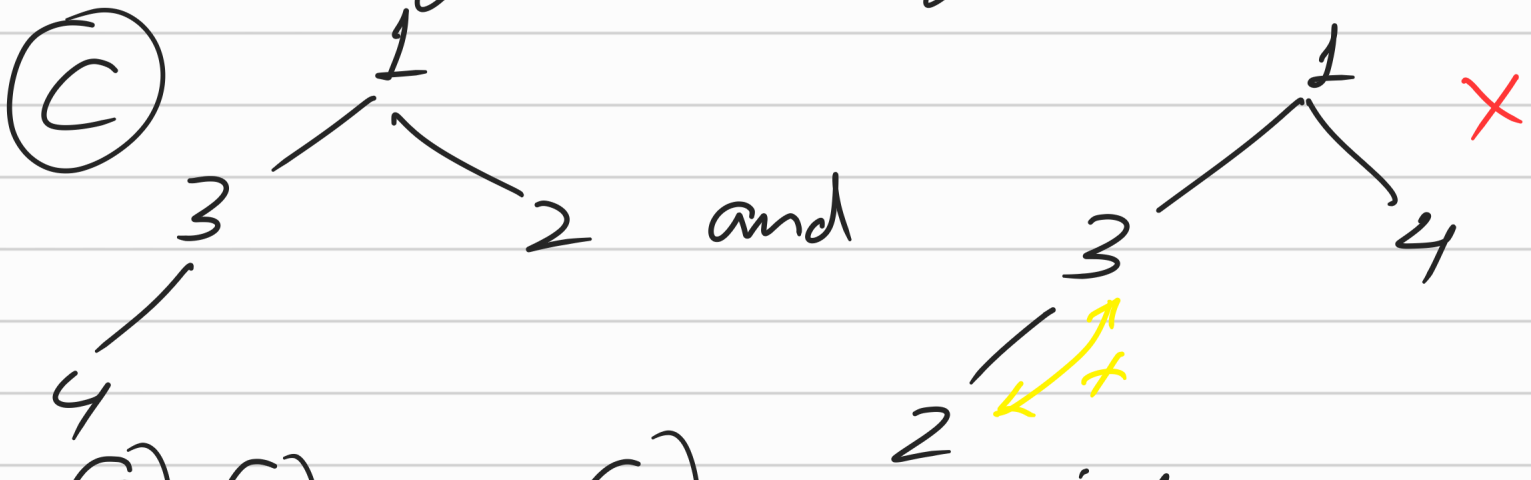
The choices for the tree entries are:-



2 left-child of 1, gives us two possibilities :-



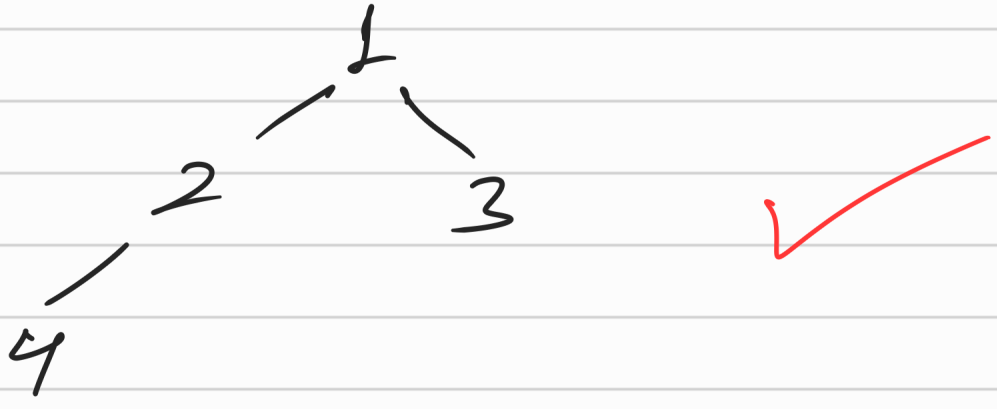
3 left-child of 1, also gives us two possibilities, but one of them does not satisfy the ordering property.



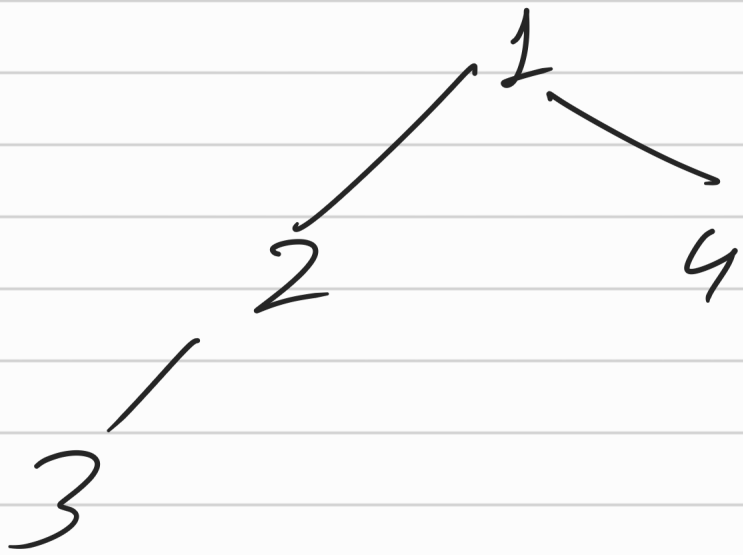
(A), (B) and (C) are min-heaps.

John 2 A heap can be implemented as an array. The first element has to be 1.
Rest of the elements 2, 3, 4 can take any permutation, we will just select the ones satisfying the ordering invariant.

Perm 1: 1 2 3 4

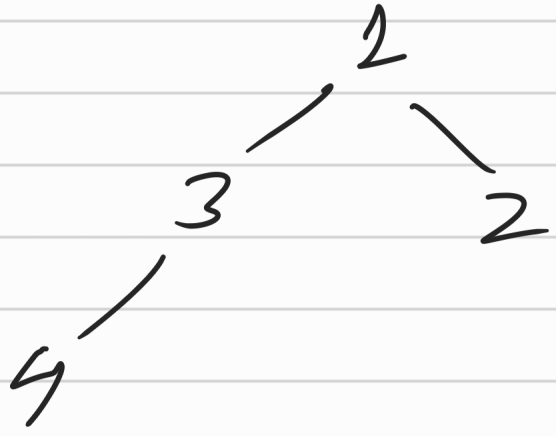


Perm 2: 1 2 4 3



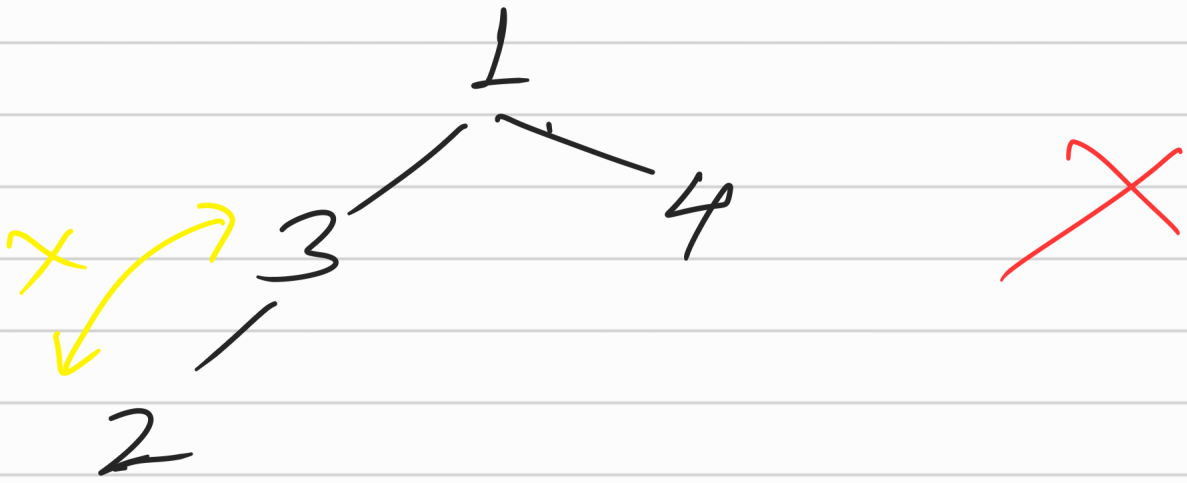
✓

Perm 3: 1 3 2 4

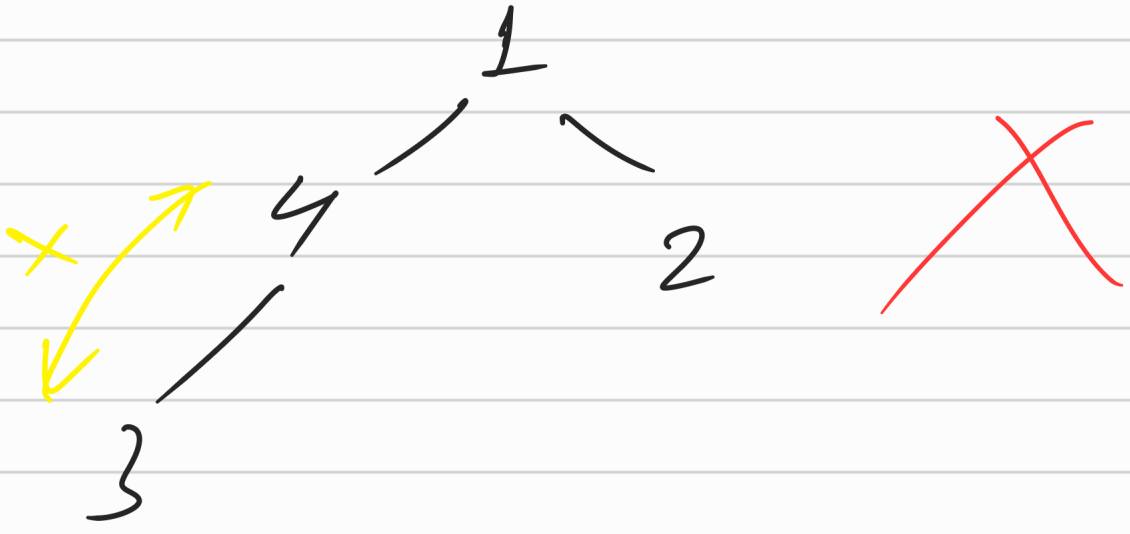


✓

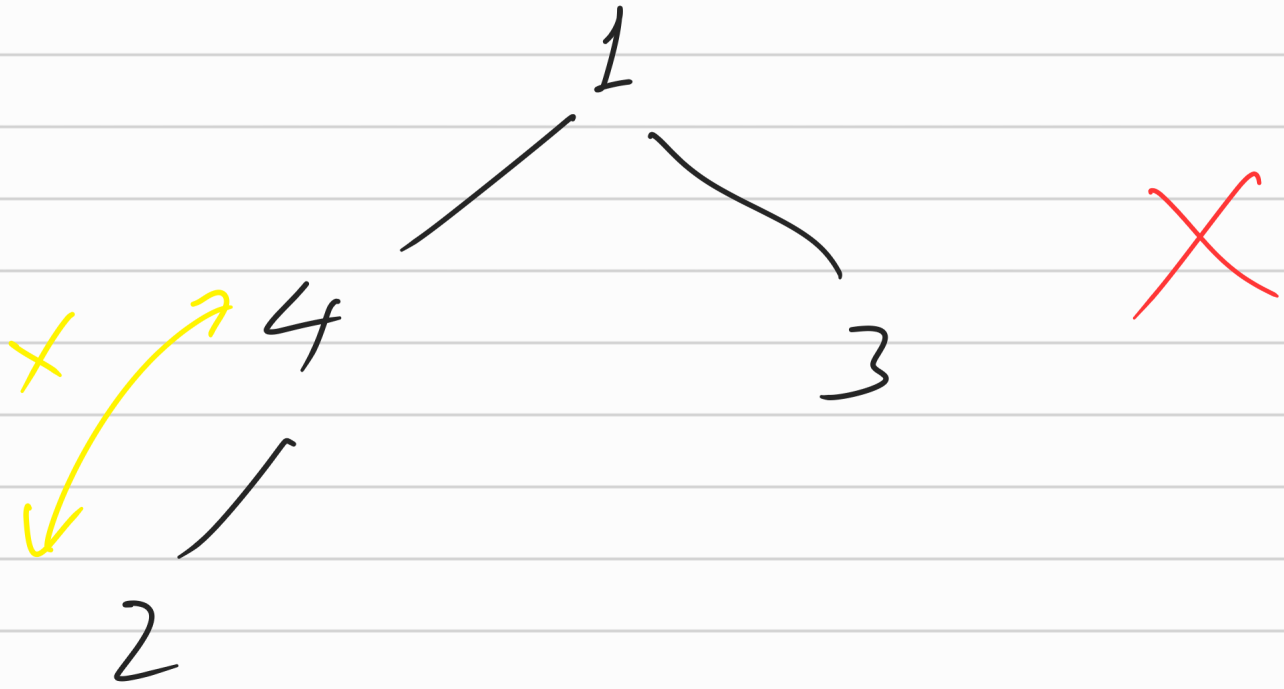
Perm 4: 1342



Perm 5: 1423



Perm 6: 1 4 3 2



Perm 1, 2 & 3 are all the ways a min-heap can be formed using 1, 2, 3, 4.

Ans 5) [outline]

Suppose a non-empty min-heap contains k elements, its height is $\leq 1 + \log_2 k$

For some constant c:

The cost of insertion is $\leq c(1 + \log_2 k)$

this is given in the question → or $O(\log k)$

As we are inserting n elements

$k \leq n$.

$\therefore$ cost of inserting any element is $\leq c(1 + \log_2 n)$

or $O(\log n)$.

Cost of inserting n elements is $\leq n \cdot c(1 + \log_2 n)$

or $O(n \log n)$.

The analysis above may seem to involve a lot of overcounting as we replaced $\log k$ with $\log n$.

Let's do a slightly better analysis.

Given: cost of inserting an element in a min-heap having k elements is $O(\log k)$.

The cost of creating a heap with one element is constant & not $c \log 0$ ($\equiv$ undefined.)

Similarly cost of inserting 2nd element is constant and not $c \log 1 = 0$

With this in mind:-

cost of inserting 3rd element $\leq c \log 2$

" " " 4th element $\leq c \log 3$

⋮ ⋮ ⋮

Cost of inserting n^{th} element $\leq c \log(n-1)$.

$\therefore$ total cost $\leq$

(cost of inserting 1st & 2nd elem) $+ c \sum_{i=2}^{n-1} \log i$

for some const c_1 :

$$\text{total cost} \leq c_1 + c_1 \sum_{i=2}^{n} \log i$$

$$= c_1 + c_1 \log \prod_{i=2}^{n} i$$

$$= c_1 + c_1 \log(n!)$$

Using Stirling's Approx

$\log n!$ is $O(n \log n)$.

$\therefore$ total cost is $O(n \log n)$.

* Remember we used it to sort an array and sorting takes $O(n \log n)$ time.