1. B
2. E (in terms of general complexity class) or G (empirically)
3. A
4. A
5. A
6a. reals
b. reals
c. positive, non-zero reals
d. cannot map to negatives, therefore not onto; x = 2 and x = -2 map to the same value, therefore not one-to-one (thus neither)

9.

```
void merge( int a[], int n1, int b[], int n2, int c[], int n3) {

   int j,k = 0;
   for( int i=0; i<n3; i++ ) {
       if( j<n1 && k<n2 ) {
           if( a[j] < b[k] ) {
              c[i] = a[j];
              j++;
           } else {
              c[i] = b[k];
              k++;
           }
      }
       else if( j>n1 && k<n2 ) { c[i] = b[k]; k++; }
       else if( j<n1 && k>n2 ) { c[i] = a[j]; k++; }
   }
}
```

10.

I(n) : s = i(i+1)/2, n=i

BC: i = 0
s = 0 by code
s = 0 by LI

IH: assume for some iteration k

IS: s_new = s_old + (k+1)
s_new = k(k+1)/2 + (k+1) (by IH)
s_new = (k^2 + k)/2 + (k+1)
s_new = (k^2 + k)/2 + 2(k+1)/2
s_new = (k^2 + 3k + 2)/2

s_new = (k+1)(k+2)/2
s_new = (k+1)(k+1+1)/2 as predicted by the LI

11.

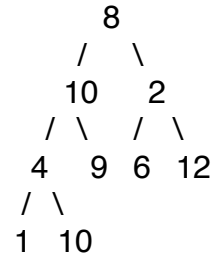
7 8 2 -3 **10** 12 21 15 11

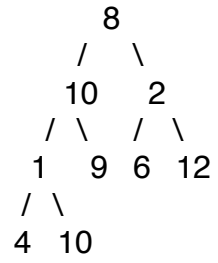
2 -3 **7** 8 11 **12** 21 15

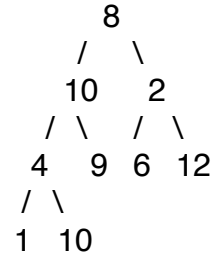
-3 **2** **8** **11** 15 **21**

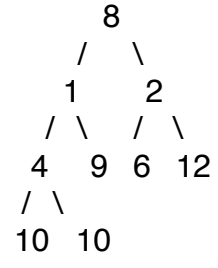
-3 **15**

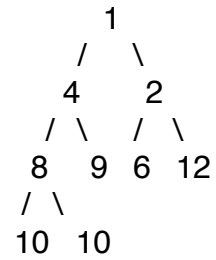
2.



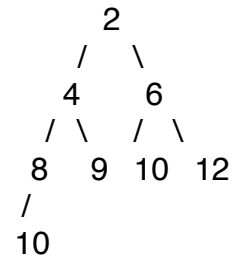




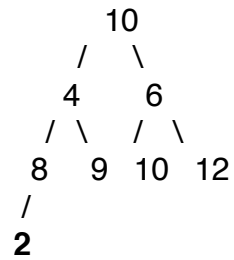


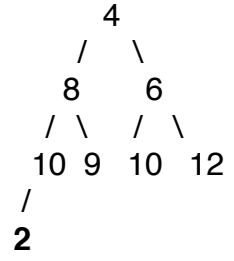

1
4 2
8 9 6 12
10 10

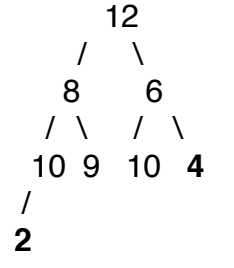
b.

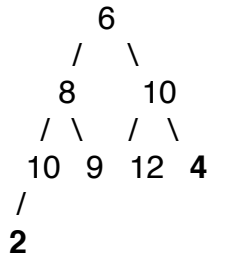

2
4 6
8 9 10 12
10

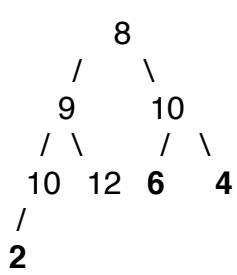
c.

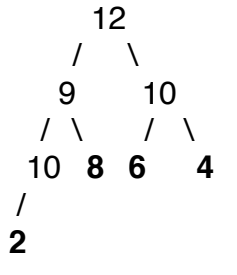

10
4 6
8 9 10 12
2

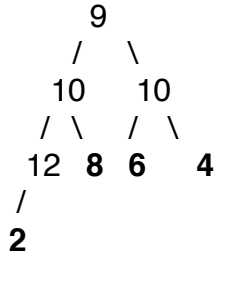

4
8 6
10 9 10 12
2

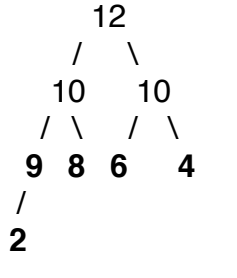

12
8 6
10 9 10 4
2

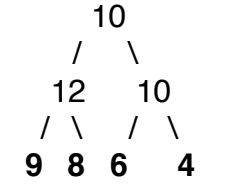

6
8 10
10 9 12 4
2

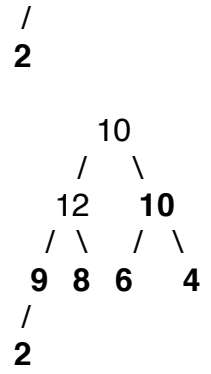

8
9 10
10 12 6 4
2

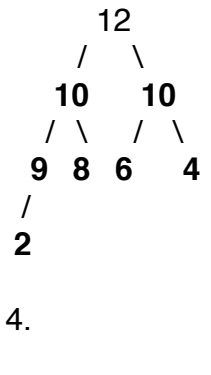

12
9 10
10 8 6 4
2


9
10 10
12 8 6 4
2

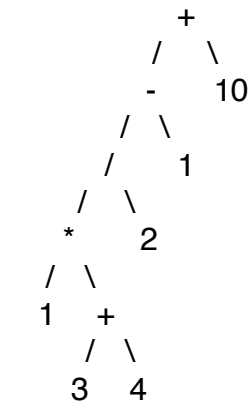

12
10 10
9 8 6 4
2


10
12 10
9 8 6 4





4.



b. 1 3 4 + * 2 / 1 - 10 +

5.

_3/4__Building a heap (heapify)
___2__Searching a BST in the worst case.
__1___Searching a binary tree in the best case.
_3/4__Searching a heap in the worst case.
__5___Heapsort

6. Not relevant for this class.

7.

```
if( !root ) return false
if( root->item == item ) return true;
if( root->item > item ) return false;
if( !heapSearch( root->left, item ) return heapSearch( root->right, item );
```