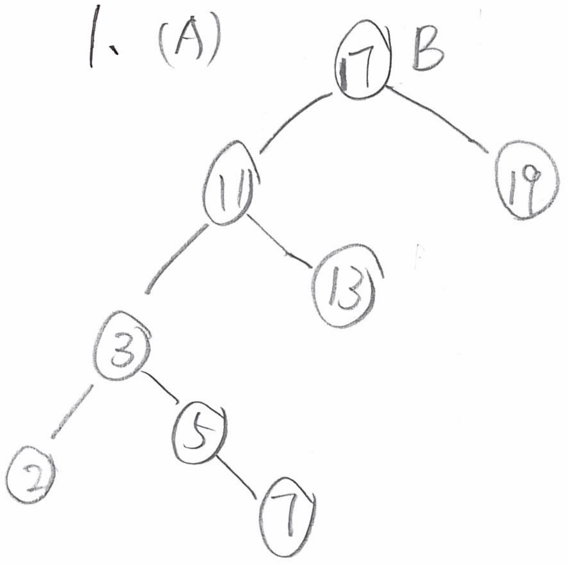
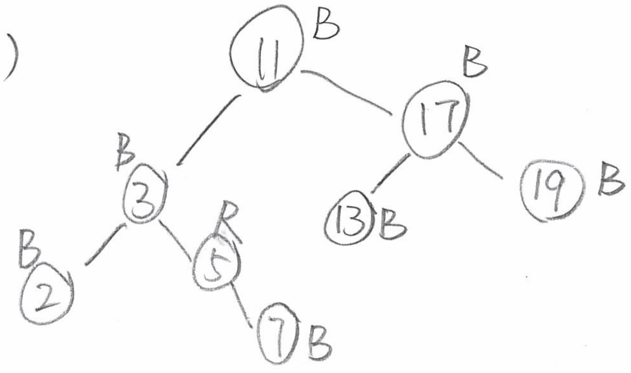
1、(A)



(B) No, start from root, all simple paths should have the same number of black nodes. the black height of the right subtree of the root is at most 2. The root's left subtree's black height should be the same as the left subtree, but this is impossible One property of R-B is to avoid the consecutive red nodes.

(C)



2、

```
parent (i):
    if i==1:
        return Nil
    return ⌈(i-1)d⌉
```

```
child (i, num):
    return (dxi)+num;
```

```
Detelemin( A ):
    if A.heap-size <1
        error "heap underflow"
    min = A[1]
    A[1] = A[A.heap-size]
    A.heap-size = A.heap-size -1
    MIN-Heapify(A, 1)
    return min
```

```
MIN-Heapify(A, i):
    for num in range(1, d):
        c = dxi + num
        if c ≤ A.heap-size and A[c] <
                                A[i
            min = c
        else
            min = i
    if  min != i
        exchange A[i] with
                  A[ min ]
        MIN-Heapify(A, min)
```

3 a) Chaining has lower collision. Chaining only forms the linked list for each slot in the hash table. Chaining does not occupy any slots. However, probing always take up spaces in the hash table.

b) Linear probing is more effective than chaining in collision handling. Because chaining needs us to do the linear traversal, it is time consuming. Access performed in linear probing tends to be closer to the memory than the chaining. In addition, insertion to the hash table by using linear probing does not require the new allocation.

c). Quadratic probing is more effective than chaining. Because quadratic probing is more effective than linear probing in dealing with collisions, because quadratic probing can avoid the clustering problem, so quadratic probing is more effective than chaining.