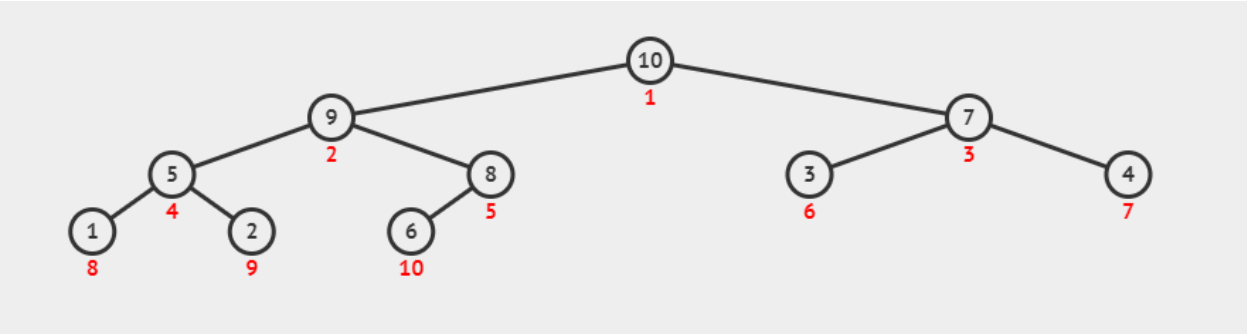
2.

a) **Max heap (MaxPQ)**

- Thêm lần lượt vào cây: 1, 3, 10, 9, 6, 7, 4, 5, 2, 8



- Lấy và xóa đi giá trị lớn nhất

- Hàm void swim (nếu nút con lớn hơn nút cha thì đổi chỗ)

```
private void swim(int k)
{
   while (k > 1 && less(k/2, k))
   {
      exch(k, k/2);
      k = k/2;
   }
}
```

parent of node at k is at k/2

- Hàm void sink (nếu nút cha nhỏ hơn 1 hoặc cả 2 nút con thì đổi chỗ nút cha với nút con lớn nhất)

```
private void sink(int k)
{
   while (2*k <= n)
   {
      int j = 2*k;
      if (j < n && less(j, j+1)) j++;
      if (!less(k, j)) break;
      exch(k, j);
      k = j;
   }
}
```

children of node at k are 2*k and 2*k+1

Tìm nút con lớn nhất

Nếu nút cha k nhỏ hơn nút con -> break

Đổi chỗ nút cha với nút con lớn nhất

Tiếp tục xét với nhánh vừa đổi chỗ

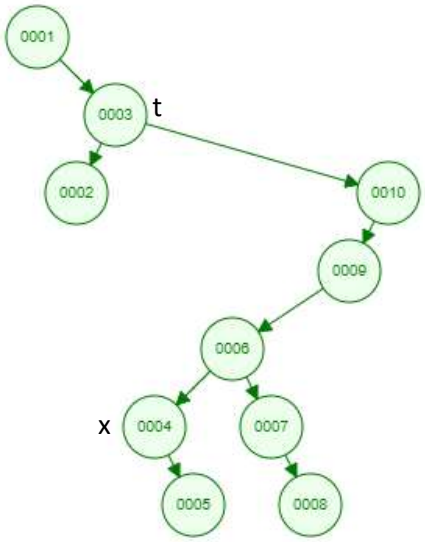
```
public Key delMax()
{
   Key max = pq[1];
   exch(1, n--);
   sink(1);
   pq[n+1] = null;
   return max;
}
```

Đổi chỗ phtu lớn nhất với phần tử cuối cùng trong mảng

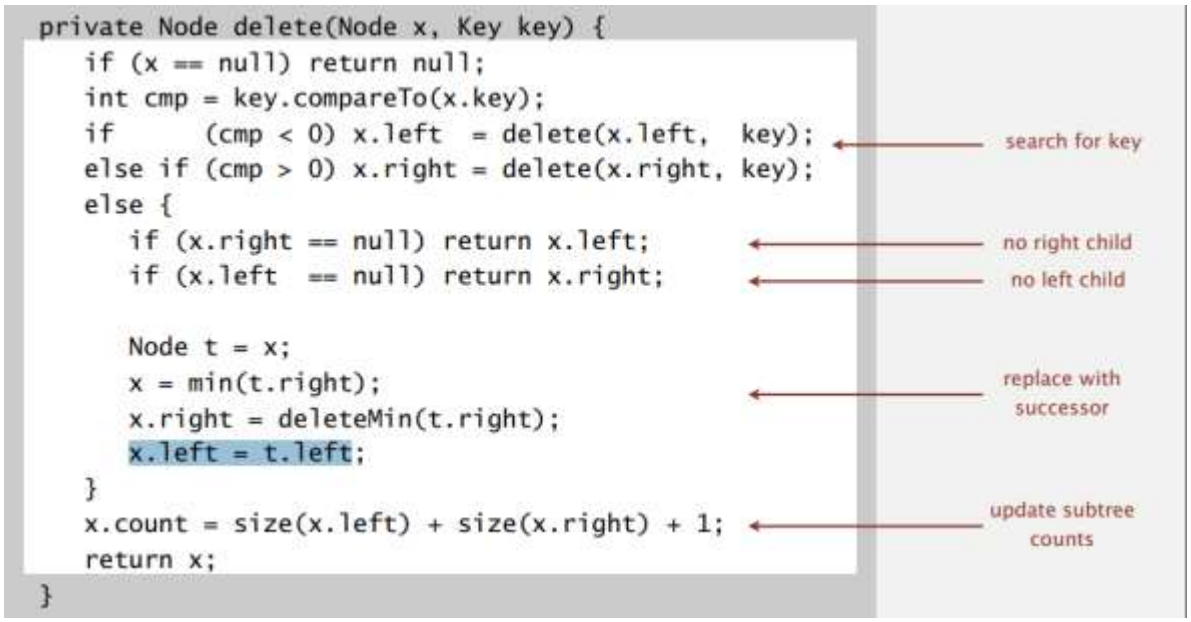
prevent loitering

b) **BST**

- Thêm lần lượt vào cây: 1, 3, 10, 9, 6, 7, 4, 5, 2, 8



- Xóa khỏi cây: 3
  - Tìm nút nhỏ nhất ở nhánh phải của 3: số 4
  - Thay thế nút 3 = nút 9
    - X.right = deleteMin(t.right) (nút số 6 nối với nút số 5, cập nhật và return nút số 10)
    - X.left = t.left
  - Cập nhật BST sau khi xóa nút số 3

- 

- Hàm deleteMin (xóa nút Min trong cây hoặc cây con)
- ```
  public void deleteMin()
  {  root = deleteMin(root);  }

  private Node deleteMin(Node x)
  {
     if (x.left == null) return x.right;
     x.left = deleteMin(x.left);
     x.count = 1 + size(x.left) + size(x.right);
     return x;
  }
  ```
  Gặp nút nhỏ nhất, return nút con phải

  Đi đến nút trái cuối cùng

3. **Hash table**
  ```

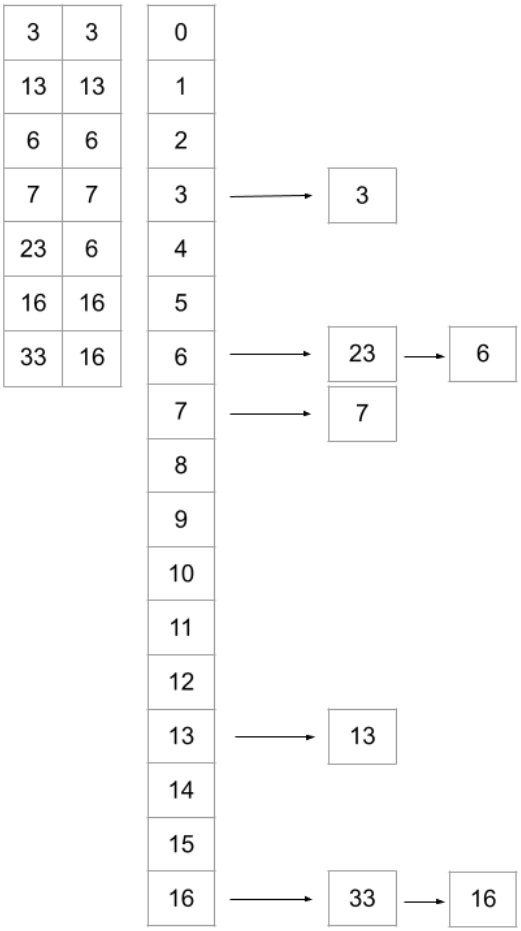
a) **Linear probing open addressing**

Index/Key	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LP	33			3			6	7	23					13			16
3				3													
13														13			
6							6										
7								7									
23							23	23	23								
16																	16
33	33																33

- Search: 6, 33, 40
  - Hash 6 = 6 -> tìm thấy 6 tại ô 6
  - Hash 33 = 16: ô 16 khác 33 -> quay ngược về tìm từ ô 0 -> tìm thấy 33
  - Hash 40 = 6 -> ô 6 khác 40, ô 7 khác 40, ô 8 khác 40, ô 9 null -> không có 40 trong hash table

b) **Separate chaining:**

- Chèn các khoá vào lần lượt là 3, 13, 6, 7, 23, 16, 33



- Tìm kiếm các khoá 6, 33, 40
  - Khóa 6: Hash = 6 -> duyệt linked list ở ô 6, tìm thấy khóa 6
  - Khóa 33: Hash = 16 -> duyệt linked list ở ô 16, tìm thấy khóa 33
  - Khóa 40: Hash = 6 -> duyệt linked list ở ô 6, không tìm thấy khóa trả về null