DATE: PAGE:

```
class BinomialHeap:
    def __init__(self):
        self.trees = []

    def extract_min(self):
        if self.trees == []:
            return None
        smallest_node = self.trees[0]
        for tree in self.trees:
            if tree.key < smallest_node.key:
                smallest_node = tree
        self.trees.remove(smallest_node)
        h = BinomialHeap()
        h.trees = smallest_node.children
        self.merge(h)

    def get_min(self):
        if self.trees == []:
            return None
        least = self.trees[0].key
        for trees in self.trees:
            if tree.key < least:
                least = tree.key
        return least

    def combine_roots(self, h):
        self.trees.extend(h.trees)
        self.trees.sort(key=lambda tree: tree.order)

    def merge(self, h):
        self.combine_roots(h)
```

DATE: PAGE:

```
if self.trees == []:
    return
i = 0
while i < len(self.trees) - 1
    current = self.trees[i]
    after = self.trees[i+1]
    if current.order == after.order:
        if (i+1) < len(self.trees) - 1
            and self.trees[i+2].order ==
                after.order):
            after after = self.trees[i+2]
            if after.key < after_after.key:
                after.add_at_end(after after)
                del self.trees[i+2]
            else
                after.after.add_at_end(after)
                del self.trees[i+2]
        else:
            if cur.key < after.key:
                cur.add_at_end(after)
                del self.trees[i+1]
            else
                after.add_at_end(cur)
                del self.trees[i]
    i = i+1

def insert(self, key):
    g = BinomialHeap()
    g.trees.append(BinomialTree(key))
    self.merge(g)
```