Sorted Maps

→ traditional map ADT allows a user to lookup val associated w/ a given key, the search for that key is known as an exact search

Sorted Search Tables

→ storing map's items in array-based sequence A so they are in increasing order of their keys

→ primary adv. of this representation is it allows usage of binary search

```
class SortedTableMap(MapBase):          //excluded get/set/del/iter
  def _find_index(self, k, low, high):
    if high < low:
      return high + 1
    else:
      mid = (low + high) // 2
      if k == self._table[mid]._key:
        return mid
      elif k < self._table[mid]._key:
        return self._find_index(k, low, mid-1)
      else
        return self._find_index(k, mid+1, high)

  def find_min(self):
    return (self._table[0]._key, self._table[0].value)

  def find_max(self):
    return (self._table[-1]._key, self._table[-1].value)

  def find_ge(self, k)
    j = self._find_index(k, 0, len(self.table)-1)
    if j < len(self._table)
      return (self._table[j]._key, self._table[j].value)

  def find_lt(self, k):
    j = self._find_index(k, 0, len(self.table) -1)
    if j > 0:
      return (self.table[j-1].key, self.table[j-1].value)

  def find_gt(self, k):
    j = self._find_index(k, 0, len(self.table)-1)
    if j < len(self.table) and self.table[j].key == k:
      j += 1
    if j < len(self.table)
      return (self.table[j].key, self.table[j].value)

  def find_range(self, start, stop):
    if start is None
      j = 0
    else
      j = self._find_index(start, 0, len(self.table)-1)
    while j < len(self.table)
      yield (self.table[j].key, self.table[j].value)
      j += 1
```