FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 Introduction to Data Structures

YEAR II, Second Semester, Midterm Examination, May 11, 2016, Time 13:00 – 16:00

ชื่อ-นามสกุล_____เลขประจำตัว | | | | | | | | | 2 | 1 | CR58_____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 11 ข้อในกระดาษคำถามคำตอบจำนวน 11 แผ่น 11 หน้า คะแนนเต็ม 110 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ควรเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
6. ผู้เข้าสอบสามารถออกจากห้องสองได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. ผู้ที่ปฏิบัติเข้าข่ายทุจริตในการสอบ ตามประกาศคณะวิศวกรรมศาสตร์
   มีโทษ คือ ได้รับ สัญลักษณ์ F ในรายวิชาที่ทุจริต และพักการศึกษาอย่างน้อย 1 ภาคการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

หมายเหตุเพิ่มเติม: ในข้อที่มีการเขียนโปรแกรมนั้น จะมีการพิจารณาประสิทธิภาพในการทำงานของโปรแกรมที่เขียนมาด้วย ให้พยายามเขียนโปรแกรมที่ทำงานได้เร็วที่สุดและใช้หน่วยความจำน้อยที่สุดเท่าที่เป็นไปได้

1.1 (5 คะแนน) จงวิเคราะห์อัตราการเติบโตของส่วนของโปรแกรมต่อไปนี้ **ให้ตรงกับความเป็นจริงมากที่สุด** โดยให้เขียนอยู่ในรูปสัญกรณ์เชิงเส้นกำกับในตัวแปร n

`// x is a vector of n elements`
`int a1(vector<int> &d, int e) {`
`  int k = -1;`
`  for (int i=d.size()-1; i>=0; --i)`
`  if (d[i] == e) k = i;`
`  return k;`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a vector of n elements`
`void a2(vector<int> &x) {`
`  for(int i=0; i<10; i++)`
`  x.insert(x.begin(),x[0]);`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a list of n elements`
`//  (circular-doubly-linked with header)`
`void a3(list<int> &x) {`
`  x.insert(x.begin(), x.back());`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a vector of n elements`
`priority_queue<int> a4(vector<int> d) {`
`  priority_queue<int> q;`
`  for (int i=0; i<d.size(); i++) q.push(d[i]);`
`  return q;`
`}`
	มีอัตราการเติบโต เป็น $O($ _____ $)$

`// x is a map of n elements (binary search tree)`
`vector<pair<int,int>> a5(map_bst<int,int> &x) {`
`  vector<pair<int,int>> d(x.size());`
`  map_bst<int,int>::iterator i=x.begin();`
`  while (i!=x.end())`
`  d.push_back(*(i++));`
`  return d;`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

1.2 (5 คะแนน) จงเติมคำตอบลงในช่องว่าง เพื่อให้ฟังก์ชันต่อไปนี้ มีอัตราการเติบโตของเวลาการทำงานเป็นไปตามที่กำหนดให้

`int b1(int n) {`
`  int s = 0;`
`  for (int i = 0; i < _____; i++)`
 	
`  for (int j = 0; j < _____; j++)`
`  for(int k = 0; k < n; k++)`
`  s += i + j + k;`
`  return s;`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n)$

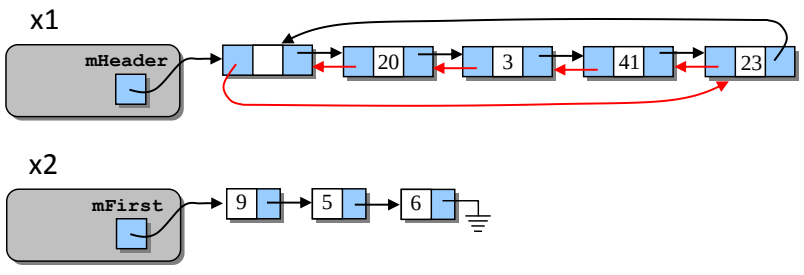
`// x is an empty vector`
`void b2(vector<int> &x, int n) {`
`  for (int i=0; i<n; i++) x.push_back(i);`
`  int s = 0;`
`  for (int i=0; i<n; i++) {`
 	
`  s += *find(x.begin(), x.end(), _____);`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n^2)$

`// x is an empty priority queue (max binary heap)`
`void b3(priority_queue<int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
 	
`  x.push( _____ );`
`  }`
`  for (int i = 0; i < n; i++) {`
`  x.pop();`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n)$

`// x is an empty map_bst (binary search tree)`
`void b4(map_bst<int,int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
 	
`  x[ _____ ] = i;`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n^2)$

`// x is an empty unordered_map`
`//  uses separate chaining hash table`
`//  with the table size of 11,`
`//  h(k) = k % 11 as the hash function and`
`//  no rehashing)`
`void b5(unordered_map<int,int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
`  x[ 4 + i*11 ] = i;`
`  }`
`  int s = 0;`
`  for (int i = 0; i < n; i++) {`
 	
`  s += x[_____];`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n^2)$

2. (10 คะแนน) `what1` เป็นเมท็อดอยู่ใน class `list` (circular doubly linked list with header) และ `what2` เป็นเมท็อดอยู่ใน class `slist` (singly linked list without header) ที่นิสิตได้เรียนโครงสร้างและการทำงานในวิชานี้ ถ้าให้ `x1` และ `x2` เป็น `list` และ `slist` ซึ่งมีโครงสร้างภายในดังรูปด้านข้างนี้ (ในรูปไม่ได้แสดง `mSize` เพราะเราไม่สนใจในโจทย์ข้อนี้)

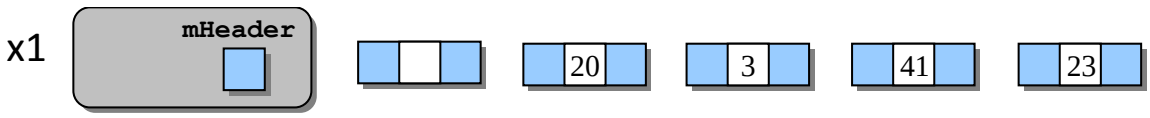


2.1 **what1** ที่ใช้กับ **x1** และ **what2** ที่ใช้กับ **x2** ข้างล่างนี้ทำอะไร จงตั้งชื่อใหม่ให้กับฟังก์ชันทั้งสองที่สื่อความหมาย และเขียนอธิบายสั้น ๆ ว่าทำอะไร (เขียนตอบในด้านขวาของตาราง)

`void what1() {`
`  node *p = mHeader->prev;`
`  if (mSize % 2 == 1)`
`  p = mHeader->prev->prev;`
`  while(p != mHeader) {`
`  p->prev->next = p->next;`
`  p->next->prev = p->prev;`
`  p = p->prev;`
`  if (p != mHeader) p = p->prev;`
`  // ยังต้องทำอะไรบางอย่าง แต่ไม่เกี่ยวกับคำถามนี้`
`  }`
`}`
	ตอบข้อ 2.1

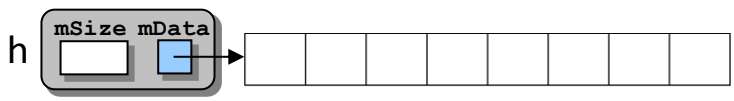
`void what2() {`
`  mFirst = what2(mFirst);`
`}`
`node *what2(node *p) {`
`  if (p == NULL) return p;`
`  p->next = what2(p->next);`
`  // ยังต้องทำอะไรบางอย่าง แต่ไม่เกี่ยวกับคำถามนี้`
`  return new node(p->data, p);`
`}`
	ตอบข้อ 2.1

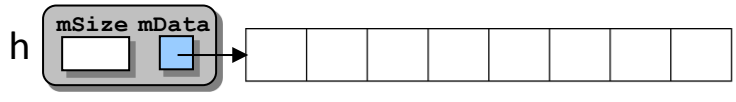
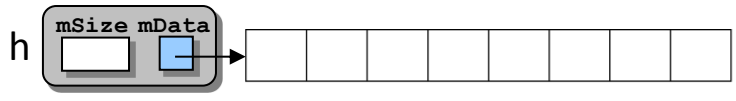
2.2 จากรูปของ **x1** ที่แสดงในหน้าที่แล้ว หลังทำจากคำสั่ง **x1.what1()** จะเปลี่ยนโครงสร้างภายใน **x1** อย่างไร (วาดเส้นเชื่อมต่างๆ ให้ครบในรูปข้างล่างนี้ ไม่ต้องสนใจ **mSize** ที่ไม่ได้เขียนในรูป)



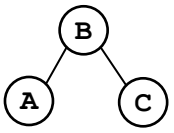
2.3 จากรูปของ **x2** ที่แสดงในหน้าที่แล้ว หลังทำคำสั่ง **x2.what2()** จงวาดโครงสร้างภายในใหม่ของ **x2** (ไม่ต้องสนใจ **mSize** ที่ไม่ได้เขียนในรูป)

3 (10 คะแนน) จงวาดโครงสร้างภายในของ priority_heap ที่สร้างด้วย binary heap (แบบ max heap) หลังทำคำสั่งหรือกลุ่มคำสั่งทางด้านซ้ายของแต่ละแถว

`priority_queue<int> h;`
`int a[] = {90,50,60,30,40,10,20};`
`for (int i=0; i<7; i++)`
`  h.push(a[i]);`
	

`h.push(52);`	
`h.pop();`	

4 (10 คะแนน) จงวาด binary tree ในช่องด้านขวา ที่เมื่อทำ inorder, preorder หรือ postorder traversal แล้วจะได้ลำดับของ nodes ตามที่เขียนไว้ในช่องด้านซ้าย

ตัวอย่าง inorder : **A, B, C** postorder : **A, C, B**	
inorder : **X, Z, Y, U, V, W** postorder : **V, U, W, Y, Z, X**	
preorder : **E, C, A, B, F, D, G, H** postorder : **A, B, E, G, D, H, F, E**	
inorder : **G, F, B, A, H, C, D** preorder : **A, B, F, G, D, C, H**	

5 (10 คะแนน) ในข้อนี้ให้นิสิตเพิ่มบริการ `node* operator[] (int k)` ให้กับ circular doubly linked list with header เพื่อเข้าถึงข้อมูล ตัวที่ k ภายใน list นี้ **โดยห้ามนิสิตใช้ STL ใดๆ และ ต้องเขียนโค้ด *ที่พิจารณา nodes เป็นจำนวนน้อยสุดเท่าที่เป็นไปได้***

```
template <typename T>
class list {  // คลาส list และ node ทำงานได้ตามปรกติทุกอย่าง ให้นิสิตเขียนบริการเพิ่มเติม ตามข้อกำหนดด้านบน
  protected:
  class node {
     friend class list;
     protected:
       T     data;  node *prev; node *next;
  };
   node   *mHeader; // pointer to a header node
   size_t  mSize;
    T operator [](int k) {          // เพิ่ม code ของนิสิตตรงนี้
   }
};
```

6 (10 คะแนน) หากเรานำเมท็อด **rotate_left_child, rotate_right_child** ของ **map_avl** และอื่น ๆ ที่เกี่ยวข้องมาไว้ใน **map_bst** จากนั้นเขียนบริการ **find_min** และ **find_max** ให้กับ **map_bst** ดังแสดงในช่องซ้ายมือข้างล่างนี้

```
class map_bst {
  class node {
    ...
    void set_left(node *n) {
      this->left = n;
      if (n != NULL)
        this->left->parent = this;
    }
    void set_right(node *n) {
      this->right = n;
      if (n != NULL)
        this->right->parent = this;
    }
  };
protected:
  node* rotate_left_child(node *r) {
    node * new_root = r->left;
    r->set_left(new_root->right);
    new_root->set_right(r);
    return new_root;
  }
  node* rotate_right_child(node *r) {
    node * new_root = r->right;
    r->set_right(new_root->left);
    new_root->set_left(r);
    return new_root;
  }
  //---------------------------------
  node* find_min(node *r) {
    if (r->left == NULL) return r;
    r->set_left(find_min(r->left));
    return rotate_left_child(r);
  }
  node* find_max(node *r) {
    if (r->left == NULL) return r;
    r->set_right(find_max(r->right));
    return rotate_right_child(r);
  }
  //---------------------------------
  ...
public:
  iterator find_min() {
    mRoot = find_min(mRoot);
    mRoot->parent = NULL;
    return iterator(mRoot);
  }
  iterator find_max() {
    mRoot = find_max(mRoot);
    mRoot->parent = NULL;
    return iterator(mRoot);
  }
  ...
};
```

จงวาด binary search tree ของ m หลังสั่งทำงานกลุ่มคำสั่งในช่องทางขวาข้างล่างนี้ (แสดงเฉพาะ key กำกับแต่ละ node ก็พอ)

```
map_bst<int,int> m;
int a[] = {5,4,3,2,1};
for (int i=0; i<5; i++) m[ a[i] ] = 1;
map_bst<int,int>::iterator it = m.find_min();
// วาดต้นไม้ในช่องว่างข้างล่างนี้
```

```
map_bst<int,int> m;
int a[] = {1,3,2,4,5};
for (int i=0; i<5; i++) m[ a[i] ] = 1;
map_bst<int,int>::iterator it = m.find_max();
// วาดต้นไม้ในช่องว่างข้างล่างนี้
```

7 (10 คะแนน) ในข้อนี้ให้นิสิตเขียนฟังก์ชั่น `node* succ(KeyT v)` ของ `map_bst` เพื่อคืน `node*` ที่มีค่า `KeyT` น้อยที่สุดที่มากกว่า `v` โดยเติม code ลงไปในที่ว่างข้างล่างนี้ **ในข้อนี้ห้ามนิสิตใช้ STL ใดๆนอกจาก pair และ เวลาการทำงานของโปรแกรมจะต้องไม่มากกว่า O(h) โดยที่ h คือความสูงของต้นไม้**

```
template <typename KeyT, typename MappedT, typename CompareT = std::less<KeyT> >
class map_bst {  // คลาส map_bst และ node ทำงานได้ตามปรกติทุกอย่าง ให้นิสิตเขียนบริการเพิ่มเติม ตามข้อกำหนดด้านบน
protected:
  typedef pair<KeyT, MappedT> ValueT;
  class node {
     ValueT data; node  *left; node  *right; node  *parent;
  };
  node     *mRoot; CompareT  mLess; size_t    mSize;
  int compare(const KeyT& k1, const KeyT& k2) {
    if (mLess(k1, k2)) return -1;
    if (mLess(k2, k1)) return +1;
    return 0;
  }
  node* succ(KeyT v) {       // เพิ่ม code ของนิสิตตรงนี้









  }
};
```

8 (10 คะแนน) ในข้อนี้ให้นิสิตเขียนตัวตรวจสอบความถูกต้องของการเปิดปิดของ HTML tag โดยให้เขียนฟังก์ชัน `bool isWellFormed( const vector<HtmlElement>& v )` โดยนิสิตสามารถเรียกใช้บริการ `string getElementType(const HtmlElement& e)` ได้ โดย ค่าที่ `getElementType` คืนมาจะเป็น string ต่อไปนี้ “<body>”, “</body>”, “ ”, “ ”, “
”, “</br>”, “ ”, “ ”, “<h1>”, “</h1>”, “<a>”, “”, “
”,””,”
”,””,”<img>” เท่านั้น (เพื่อความง่าย, เราให้นิสิตดู HTML tag แค่บางตัวเท่านั้น) โดย `isWellFormed` จะคืนค่าจริงถ้าทุกๆ tag มีการเปิดและปิดถูกต้องเช่น <ul> ก็ต้องปิดด้วย เป็นต้นนอกจาก tag <img> ซึ่งไม่มี tag ปิด ตัวอย่างเช่น

<body>   </body> นั้น `isWellFormed` เป็น true
       <img> นั้น `isWellFormed` เป็น true
        นั้น `isWellFormed` เป็น false

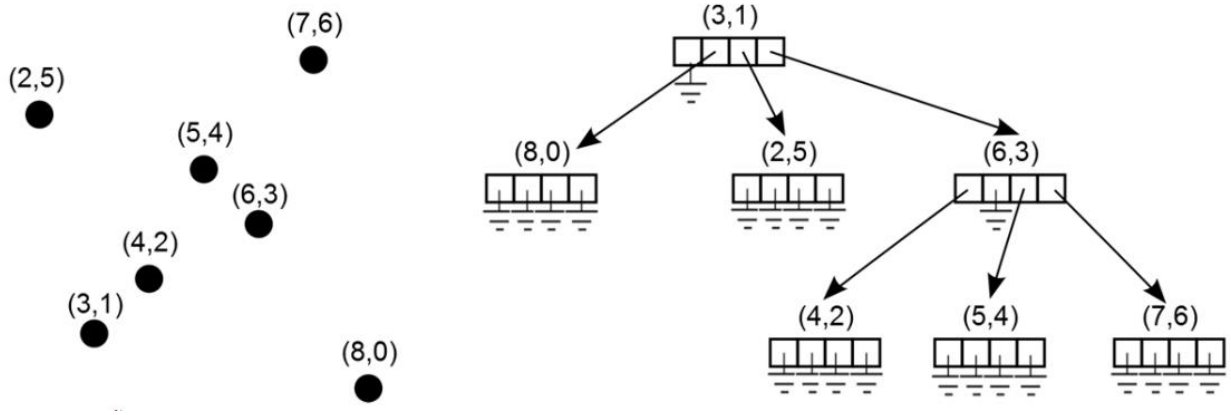
โดยในข้อนี้นิสิตสามารถใช้ STL ใดๆมาช่วยก็ได้ (ถือว่า include ให้ตามเหมาะสมเรียบร้อยแล้ว) และ เวลาการทำงานของโปรแกรมจะต้องไม่มากกว่า O(n) โดยที่ n คือจำนวน tag ที่มี

```
bool isWellFormed(const vector<HtmlElement>& v ) {









}
```

9 (10 คะแนน) มีโครงสร้างข้อมูลชนิดหนึ่ง ซึ่งมีชื่อว่า Quad Tree โดย Quad Tree เป็นโครงสร้างข้อมูลแบบต้นไม้ที่ใช้เก็บจุดในสองมิติ โดยแต่ละปมอาจจะเป็น NULL หรือไม่ก็เก็บ พิกัด (x, y) ของจุด และจะมีลูก 4 ปม โดยลูกแต่ละตัวจะต้องมีคุณสมบัติดังนี้

- ลูก bottomLeft จะต้องเป็น NULL หรือมีพิกัด (x1, y1) โดยที่ x1 < x และ y1 < y
- ลูก bottomRight จะต้องเป็น NULL หรือมีพิกัด (x2, y2) โดยที่ x2 > x และ y2 < y
- ลูก topLeft จะต้องเป็น NULL หรือมีพิกัด (x3, y3) โดยที่ x3 < x และ y3 > y
- ลูก topRight จะต้องเป็น NULL หรือมีพิกัด (x4, y4) โดยที่ x4 > x และ y4 > y

ตัวอย่างเช่น เมื่อเรามีจุด (3,1), (8,0), (2,5), (6,3), (4,2), (5,4), (7, 6) ตัวอย่าง Quad Tree หนึ่งที่เป็นไปได้

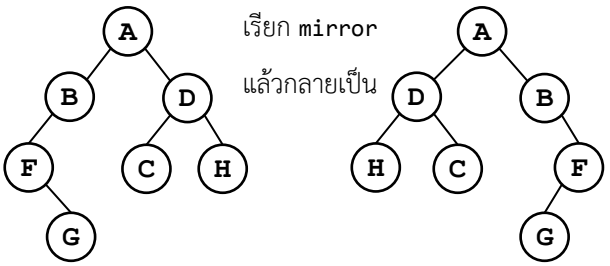


ในข้อนี้เรารับรองว่าจะไม่มีสองจุดใด ๆ ที่มีค่าพิกัดแกน x หรือค่าพิกัดแกน y เท่ากันเลย ตัวอย่างเช่น รับรองว่าถ้ามีจุด (1,3) แล้ว จะไม่มีจุด (1,4) เป็นต้น จงเติมส่วนของโปรแกรมของฟังก์ชัน `findPointAt(int x, int y)` ซึ่งหาจุดที่อยู่ที่พิกัด (x,y) ที่อยู่ในโครงสร้างข้อมูลประเภท `QuadTree` ให้สมบูรณ์โดยจะคืนค่า NULL ถ้าไม่พบจุดที่พิกัดนี้

```
template <typename T>
class QuadTree {
protected:
    class QNode{
    public:
        int x, y;
        T data;
        QNode *bottomLeft, *bottomRight, *topLeft, *topRight;
    };
    QNode* root;
```

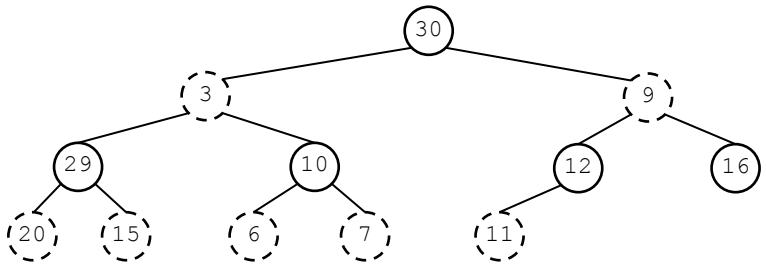
```
    int mSize; // จำนวนข้อมูลที่เก็บในต้นไม้
    // ประกาศฟังก์ชันหรือตัวแปรเพิ่มที่นี่ (หากต้องการ)




    QNode* findPointAt(int x, int y) {          // ใส่ code ของคุณที่นี่









    }
};
```

10 (10 คะแนน) กำหนดให้มีโครงสร้างข้อมูลแบบ Binary Tree ดังปรากฏด้านล่างนี้ จงเพิ่มบริการ `void mirror( )` ซึ่งจะทำให้ Binary Tree ต้นนั้นกลายเป็น Binary Tree ที่ลูกซ้ายขวาของทุกปมนั้นสลับกัน ดังตัวอย่างในรูปด้านข้างนี้ การเขียนบริการ `mirror` นั้นให้***<u>เขียนโดยใช้วิธีการแบบ recursive เท่านั้น</u>*** นิสิตสามารถเขียนฟังก์ชันอื่นเพิ่มเติมได้ตามความเหมาะสม (หมายเหตุ: บริการ `mirror` นั้นไม่จำเป็นต้องมีการสร้าง node ใหม่ แต่จะสร้างก็ได้ ถ้ามีการสร้าง node ใหม่ ต้องลบ node เก่าด้วย)




```
template <typename T>
class BinaryTree {
public:
  class node {
    friend class BinaryTree;
    protected:
      T data; node  *left; node  *right; node  *parent;
      node() : data(T()), left(NULL), right(NULL), parent( NULL ) { }
      node(const T& data, node* left, node* right, node* parent) :
           data (data), left(left), right(right), parent(parent) { }
  };
  node *mRoot;
  // เพิ่ม code ต่าง ๆ ด้านล่างนี้ โดยจะต้องมี void mirror() ด้วย









};
```

11 (10 คะแนน) โครงสร้างข้อมูลประเภท Binary Heap ที่เรียนในชั้นเรียนนั้น สามารถ “ลบ” และ “ถามหา” ข้อมูลที่มีค่ามากสุดได้ง่าย เราต้องการปรับ Binary Heap ให้สามารถ “ลบ” และ “ถามหา” ข้อมูลที่มีค่า มากสุด และ น้อยสุด ได้เร็ว โดยที่การใส่ข้อมูล ยังเป็น O(log N) อยู่เช่นเดิม วิธีการหนึ่งที่ทำได้คือ ใช้ Heap แบบที่เรียกว่า Min Max Heap ซึ่ง มีโครงสร้างเป็น Binary Tree ที่เต็มทุกชั้น โดยที่ชั้นล่างสุดเต็มจากซ้ายไปขวาเหมือน Binary Heap ปรกติ แต่เราจะแบ่ง ปมออกเป็นสองกลุ่ม คือ ปมในชั้น min และ ปมในชั้น max สำหรับปม x ใด ๆ ใน ชั้น min มีข้อกำหนดคือ ปม x จะต้องมีค่าน้อยกว่าทุก ๆ ปมที่อยู่ใต้ปม x นั้น และสำหรับปม y ใด ๆ ในชั้น max มีข้อกำหนดคือปม y จะต้องมีค่ามากกว่าทุก ๆ ปมที่อยู่ใต้ปม y เราจะถือว่าปมรากนั้นเป็นปมชั้น max ปมที่เป็นลูกของปมในชั้น max จะเป็นปมในชั้น min และ ปมที่เป็นลูกของปมในชั้น min นั้นจะเป็นปมในชั้น max กล่าวอีกนัยหนึ่งก็คือ ปมใน Min Max Binary Heap นั้นจะสลับไปมาระหว่างปมในชั้น min กับ ปมในชั้น max รูปด้านขวานี้แสดงถึงตัวอย่างของ Min Max Binary Heap ปมที่วาดด้วยเส้นประคือปมในชั้น min



จงเขียน method ดังต่อไปนี้สำหรับโครงสร้างข้อมูลแบบ Min Max Heap ที่ทำงานในเวลาที่กำหนด

- bool isMinLevel(size_t idx) ซึ่งจะคืนค่า true ก็ต่อเมื่อปมที่อยู่ ณ ตำแหน่ง idx นั้นเป็นปมในชั้น min (ต้องใช้เวลาเป็น O(1))
- T top() ซึ่งจะคืนค่าที่มีค่ามากสุดใน Min Max Heap (ต้องใช้เวลาเป็น O(1))
- T bottom()ซึ่งจะคืนค่าที่มีค่าน้อยสุดใน Min Max Heap (ต้องใช้เวลาเป็น O(1))
- void push(T data) ซึ่งจะเพิ่มค่า data เข้าไปใน Min Max Binary Heap นี้ (ต้องใช้เวลาเป็น O(log N)) การเพิ่มข้อมูลนั้นมีหลักการทำงานดังนี้ ให้สร้างปมใหม่ ณ ตำแหน่งเดียวกับการ push ของ Binary Heap ปรกติ
    a. สมมติให้ปมที่เพิ่มนั้นเป็นปมในชั้น min 1) ถ้าค่า data นั้นมากกว่าปมพ่อ (ซึ่งปมพ่อนั้นจะเป็นปมในชั้น max) แสดงว่าเงื่อนไขของปมนั้นกับปมพ่อผิด ก็จะทำการสลับค่า data กับปมพ่อ หลังจากสลับ จะเห็นได้ว่า ปมใหม่นั้นมีค่าที่ถูกต้องแล้ว และบรรดาปมในชั้น min ตั้งแต่ปมใหม่ไปจนถึงรากก็จะมีค่าที่ถูกต้องเช่นกัน เราก็จะทำการพิจารณาเฉพาะปมในชั้น max ตั้งแต่ปมพ่อของเราไปจนถึงปมราก และทำการสลับค่าให้ถูกต้องด้วยวิธีการทำนองเดียวกับ Binary Heap ปรกติ 2) แต่ถ้าค่า data นั้นน้อยกว่าปมพ่อ แสดงว่าบรรดาปมในชั้น max ตั้งแต่ปมพ่อไปจนถึงรากก็จะมีค่าที่ถูกต้องแล้ว เราก็จะทำการพิจารณาเฉพาะปมในชั้น min ตั้งแต่ปมเราไปจนถึงปมราก และทำการสลับค่าให้ถูกต้องด้วยวิธีการทำนองเดียวกับ Binary Heap ปรกติ
    b. สมมติให้ปมที่เพิ่มนั้นเป็นปมในชั้น max ก็ใช้หลักการเดียวกัน ก็คือเปรียบเทียบค่า data กับปมพ่อ (ซึ่งปมพ่อจะเป็นปมในชั้น min) ถ้าค่าระหว่าง data กับพ่อของ data นั้นผิดเงื่อนไขของปมพ่อ ก็จะทำการสลับ ซึ่งจะทำให้ปมในชั้น max ตั้งแต่ปมใหม่เป็นต้นไปมีค่าถูกต้อง เราก็จะพิจารณาเฉพาะปมในชั้น min ตั้งแต่พ่อขึ้นไปแทน แต่ถ้าค่า data ไม่ผิดเงื่อนไขกับปมพ่อ ก็จะพิจารณาเฉพาะปมในชั้น max ตั้งแต่ปมใหม่ขึ้นไปแทน

```
template <typename T,typename Comp = std::less<T> >
class MinMaxBHeap {
  protected:
    T *mData;
    size_t mCap;
    size_t mSize;
    Comp mLess;

  public:
    // ให้ถือว่าคลาสนี้มีฟังก์ชันต่าง ๆ เหมือนกับ priority_queue อยู่แล้ว ให้นิสิตเขียนฟังก์ชันตามที่โจทย์กำหนดให้เพิ่มเติม
    // ในกรณีที่ฟังก์ชันที่เขียนขึ้นมาใหม่นั้นมี parameter และ ชื่อตรงกับฟังก์ชันที่มีอยู่แล้ว จะถือว่าฟังก์ชันที่เขียนมานั้นมาแทนที่ตัวเดิม
```

```
};
```

Common

All classes support these two capacity functions;

Capacity	`size_t size(); // return the number of items in the structure`
	`bool empty(); // return true only when size() == 0`

Container Class

All classes in this category support these two iterator functions.

Iterator	`iterator begin(); // an iterator referring to the first element`
	`iterator end(); // an iterator referring to the *past-the-end* element`

Class vector<ValueT>, list<ValueT>

Element Access	`operator[] (size_t n);`

Modifier ที่ใช้ได้ทั้ง list และ vector	`void push_back(const ValueT& val);`
	`void pop_back();`
	`iterator insert(iterator position, const ValueT& val);`
	`iterator insert(iterator position, InputIterator first, InputIterator last);`
	`iterator erase(iterator position);`
	`iterator erase(iterator first, iterator last);`

Modifier ที่ใช้ได้เฉพาะ list	`void push_front(const ValueT& val);`
	`void pop_font;`
	`void remove(const ValueT& val);`

Class set<ValueT,CompareT = less<ValueT> >,
unordered_set<ValueT, HashT = hash< ValueT >, EqualT = equal_to< ValueT > >

Operation	`iterator find (const ValueT& val);` `size_type count (const ValueT& val);`

Modifier	`pair<iterator,bool> insert (const ValueT& val);`
	`void insert (InputIterator first, InputIterator last);`
	`iterator erase (iterator position);`
	`iterator erase (iterator first, iterator last);`
	`size_type erase (const ValueT& val);`

Class map<KeyT, MappedT, CompareT = less<KeyT> >
unordered_map<KeyT, MappedT, HashT = hash<KeyT>, EqualT = equal_to<KeyT> >

Element Access	`MappedT& operator[] (const KeyT& k);`
Operation	`iterator find (const KeyT& k);` `size_type count (const KeyT& k);`

Modifier	`pair<iterator,bool> insert (const pair<KeyT,MappedT>& val);`
	`void insert (InputIterator first, InputIterator last);`
	`iterator erase (iterator position);`
	`iterator erase (iterator first, iterator last);`
	`size_type erase (const KeyT& k);`

Container Adapter

These three data structures support the same data modifiers but each has different strategy. These data structures do not support iterator.

Modifier	`void push (const ValueT& val); // add the element`
	`void pop();  // remove the element`

Class queue<ValueT>

Element Access	`ValueT front();` `ValueT back();`

Class stack<ValueT>

Element Access	`ValueT top();`

Class priority_queue<ValueT, ContainerT = vector<ValueT>, CompareT = less<ValueT> >

Element Access	`ValueT top();`

Useful function

```
iterator find(iterator first, iterator last, const T& val);  // find by iteration, using O(N)
void sort (iterator first, iterator last, Compare comp);  // sort, using O( N log(N) )
pair<T1,T2> make_pair (T1 x, T2 y);
```