数据结构实验报告

黄子懿　软件学院　1913082

实验题目：搜索树性能比较

1　数据结构说明

1.1　BSTree

性质

一棵非空的二叉搜索树满足：

(1) 每个元素有一个关键字，并且任意两个元素的关键字都不同；所以关键字唯一。

(2) 在根节点的左子树中，元素的关键字都小于根节点的关键字。

(3) 在根节点的右子树中，元素的关键字都大于根节点的关键字。

(4) 根几点的左右子树也是二叉搜索树。

搜索

如果需要查找的关键字比当前节点关键字小，在当前节点左子树查找；

如果需要查找的关键字比当前节点关键字大，在当前节点右子树查找。

插入

通过查找确认当前的树中是否已经有插入关键字的节点；如果没有，则在中断节点的孩子处插入。

删除

查找到需要删除的节点：

如果该节点为叶节点，则直接删除；

如果该节点只有一个孩子，则删除该节点并将父节点指向该节点的子节点；

如果该节点有 2 个孩子，则将该节点替换为其前驱/后缀，然后将前驱/后缀删除。

缺点

二叉搜索树在最坏的情况下树的高度可以达到 O(n)，插入、搜索和删除的复杂度最大可以达到 O(n)。

1.2 AVLTree

性质

如果 T 是一棵非空的 AVL 树，T_L 和 T_R 分别是其左子树和右子树，则

(1) T_L 和 T_R 都是 AVL 树

(2) $|h_L - h_R| \leq 1$

高度

对于一棵高度为 h 的 AVL 树，令 N_h 为其最少节点数。最坏情况下

$$N_h = N_{h-1} + N_{h-2} + 1, \quad N_0 = 0, \quad N_1 = 1$$

可得

$$N_h \approx \phi^{h+2}/\sqrt(5) - 1$$

所以高度为 O(logn)

搜索

同 BSTree。(红黑树的搜索操作也与 BSTree 相同，不再赘述)

插入和删除

基本操作同 BSTree，操作完成后通过旋转保持平衡。

四种旋转模式分别为 LL, RR, LR, RL

LL 型旋转

```
void LL(Node *&u)
{
    Node *ul = u->left;
    u->left = ul->right; u->lh = ul->rh;
    ul->right = u; ul->rh = max(u->lh, u->rh)+1;
    u = ul;
}
```

RR 型旋转

```
void RR(Node *&u)
{
    Node *ur = u->right;
    u->right = ur->left; u->rh = ur->lh;
    ur->left = u; ur->lh = max(u->lh, u->rh)+1;
    u = ur;
}
```

LR 型旋转

```
void LR(Node *&u)
{
    RR(u->left);
    LL(u);
}
```

RL 型旋转

```
void RL(AVLNode *&u)
{
    LL(u->right);
    RR(u);
}
```

1.3 RBTree

性质

满足一下条件的一棵扩展二叉树:
(1) 根节点和所有外部节点都是黑色;
(2) 在根至外部节点路径上，没有连续两个节点是红色;
(3) 在所有根至外部节点的路径上，黑色节点的数量相同。

高度

设 h 是一棵红黑树的高度，n 是树内部节点数量，r 是根节点的阶，则

$$h \leq 2log_2(n+1)$$

插入

插入新节点为红色，仅在父节点也为红色的情况下，会破坏性质 (2)。

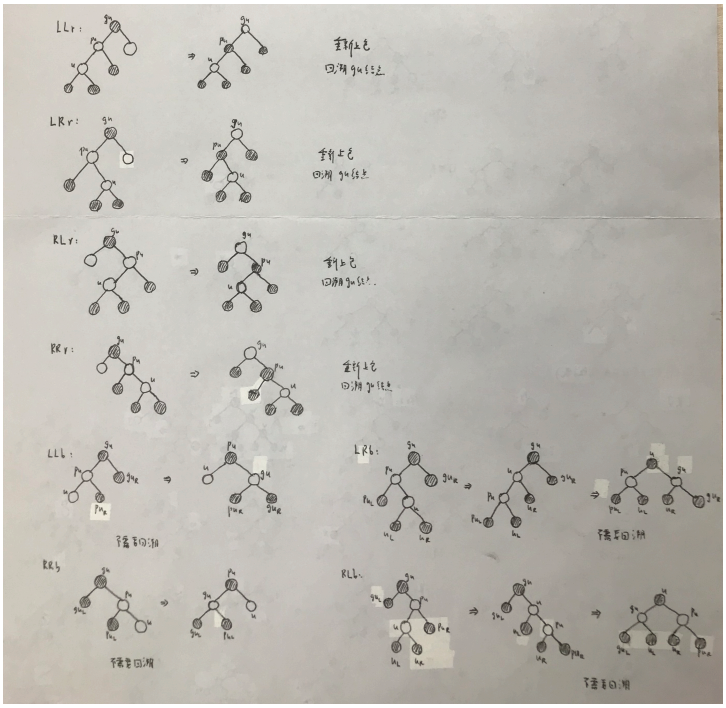


图 1: 红黑树插入

删除

当且仅当被删除节点为黑色且非树根时，会破坏性质 (3)。

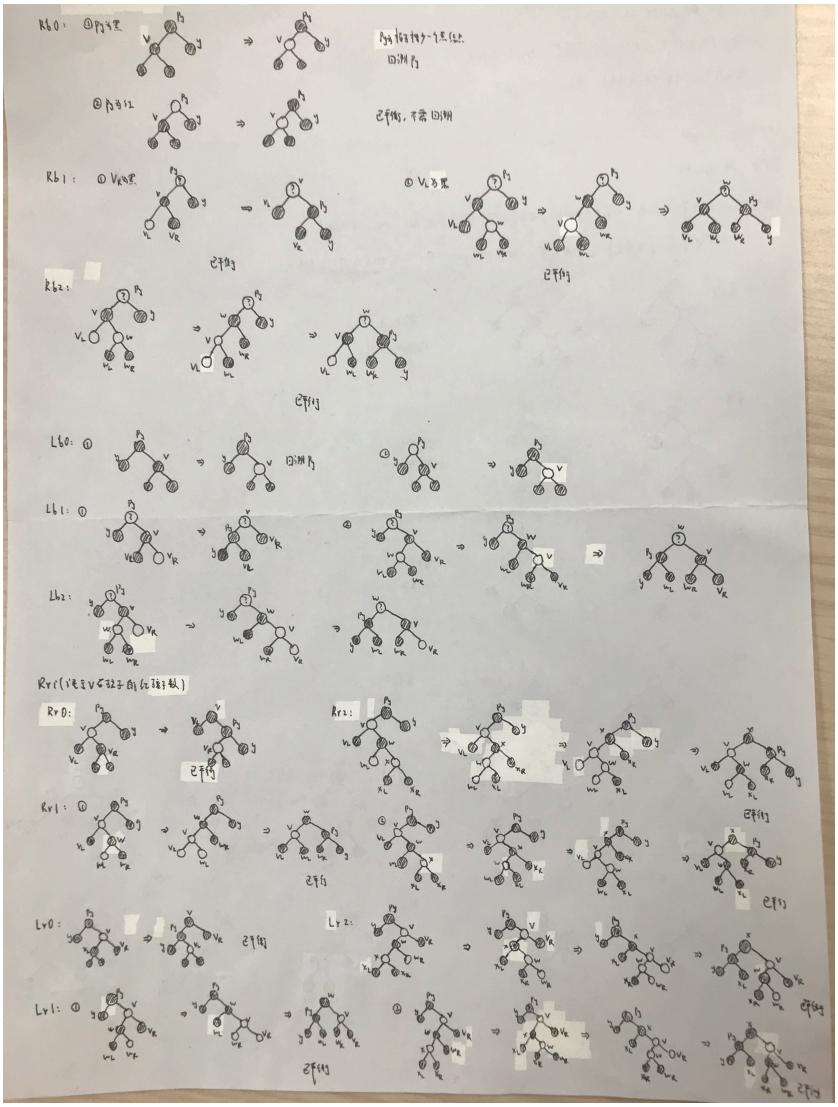


图 2: 红黑树删除

1.4 B-Tree

性质

m 阶 B-树是一棵 m 叉搜索树。如果 B-树非空，那么满足：

(1) 根节点至少 2 个孩子；

(2) 除根节点以外，所有内部节点至少有 $\lceil m/2 \rceil$ 个孩子；

(3) 所有外部节点在同一层。

高度

设 T 是一棵高度为 h 的 m 阶 B-树。令 $d = \lceil m/2 \rceil$，n 是 T 的元素个数，则

$$log_m(n+1) \leq h \leq log_d(\frac{n+1}{2}) + 1$$

搜索

和 m 阶搜索树的搜索方式相同。在一个节点中，如果需要搜索的值处于两个元素之间，则沿着两个元素之间的指针向下搜索。

插入

如果插入的节点不饱和，直接插入即可；

如果插入的节点已经饱和，则将该节点分裂，并将中间的节点传送给父节点，再回溯父节点。

删除

如果要删除的元素位于非叶节点，则通过查找前驱/后缀的方法转化为删除叶节点中元素的情况。

如果要删除元素所在的叶节点元素个数大于最少数，直接删除即可；

否则，若其左/右兄弟元素个数大于最少数，通过旋转向兄弟节点借来一个元素；若其左/右兄弟元素个数等于最少数，向父节点借一个元素并将这个节点和其兄弟节点合并，回溯父节点。

2 数据集设计

考察四棵树在“插入 N 个整数，随机查找 1000 个数，按某一顺序删除”这一模式下的性能。

其中数组 in[i] 为插入的顺序，se[i] 为需要搜索的 1000 个数，de[key[i]] 为删除的顺序。

在每种情况下，对于 N 从 10000 到 500000 间隔 10000，随机生成 in,se,de,key 数组对四棵树进行 3 组测试。测试代码详见附录。

2.1 正序插入，正序删除

按递增顺序插入 N 个整数，随机查找 1000 个数，按相同顺序删除。

```
void order()
{
    srand(time(NULL));
    for(int i = 0; i < n; i++)
        in[i] = rand();
    sort(in, in+n);
    for(int i = 0; i < n; i++)
        de[i] = in[i], key[i] = i;
    for(int i = 0; i < ms; i++)
        se[i] = rand();

    solve();
}
```

2.2 正序插入，逆序删除

按递增顺序插入 N 个整数，随机查找 1000 个数，按相反顺序删除。

```
void invorder()
{
    srand(time(NULL));
    for(int i = 0; i < n; i++)
        in[i] = rand();
```

```
    sort(in, in+n);
    for(int i = 0; i < n; i++)
        de[i] = in[i], key[i] = n-i-1;
    for(int i = 0; i < ms; i++)
        se[i] = rand();

    solve();
}
```

2.3 随机插入，随机删除

按随机顺序插入 N 个整数，随机查找 1000 个数，按随机顺序删除。

```
void rdorder()
{
    srand(time(NULL));
    for(int i = 0; i < n; i++)
        in[i] = rand();
    for(int i = 0; i < n; i++)
        de[i] = in[i], key[i] = i, rd[i] = rand();
    sort(key, key+n, cmp);
    for(int i = 0; i < ms; i++)
        se[i] = rand();

    solve();
}
```

3 数据处理

完整代码详见附件。

程序运行结果生成了 3 种情况下，四棵树在 N 从 10000 到 500000 间隔 10000 的时候的运行时间各 3 组。

使用 python 求 3 组数据的平均值并用 matplotlib 库作图。

3.1 正序插入，正序删除

插入

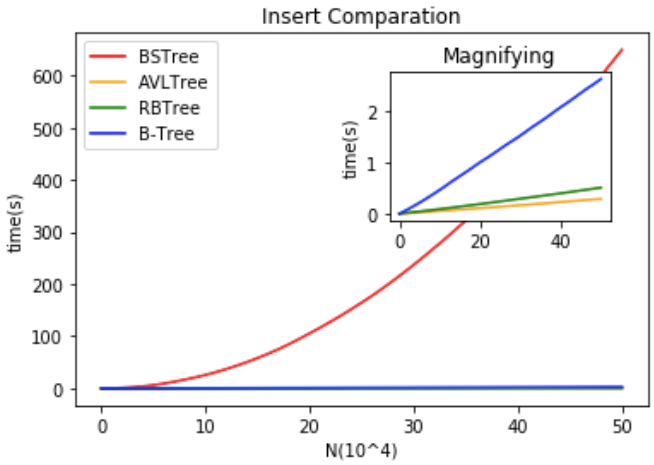


图 3: 1-插入

搜索

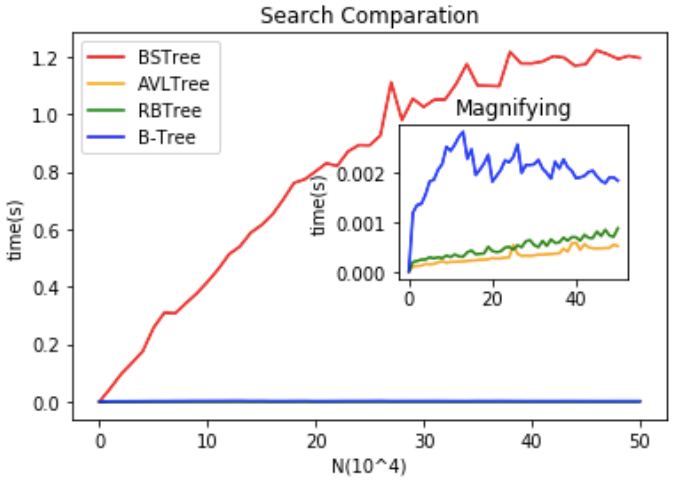


图 4: 1-搜索

删除

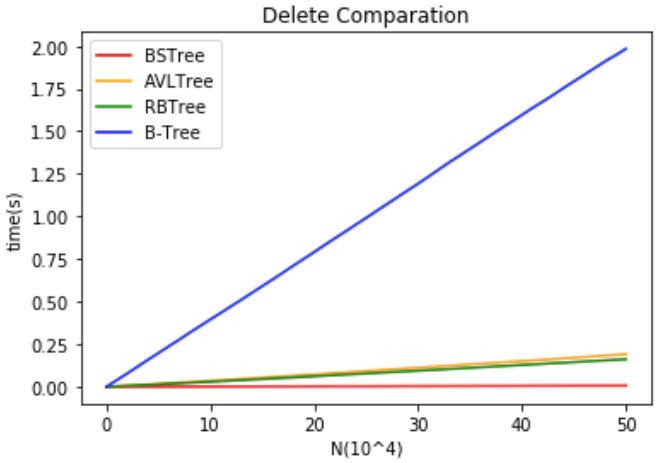


图 5: 1-删除

3.2 正序插入，逆序删除

插入

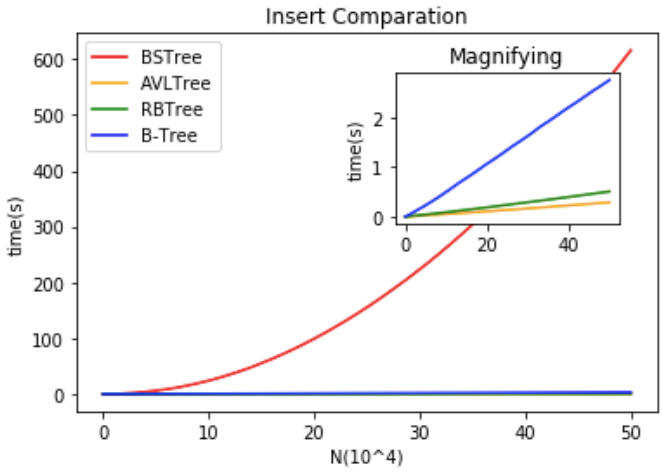


图 6: 2-插入

搜索

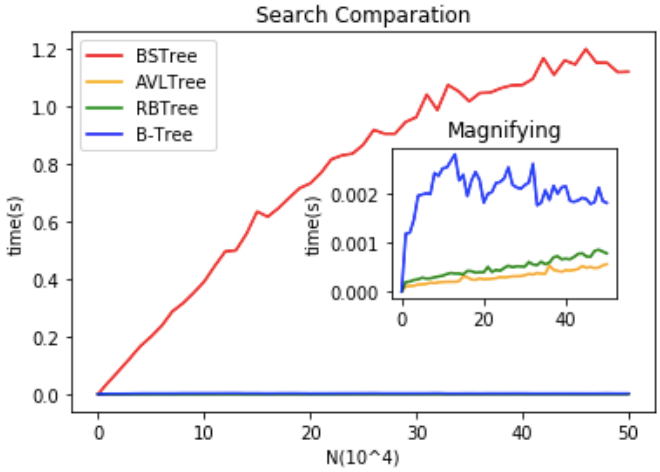


图 7: 2-搜索

删除

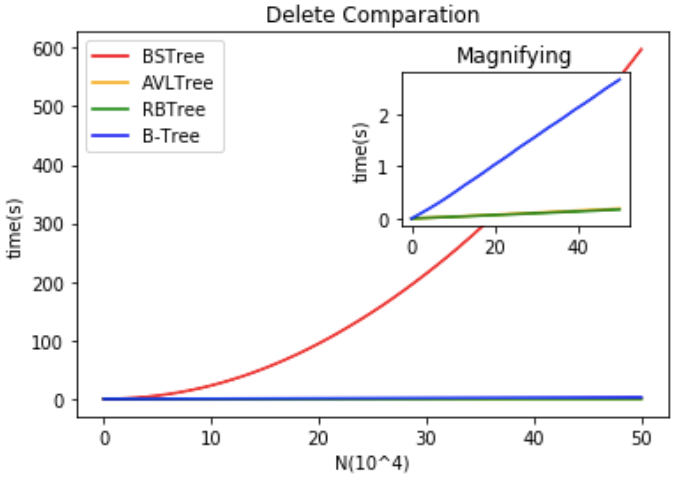


图 8: 2-删除

3.3 随机插入，随机删除

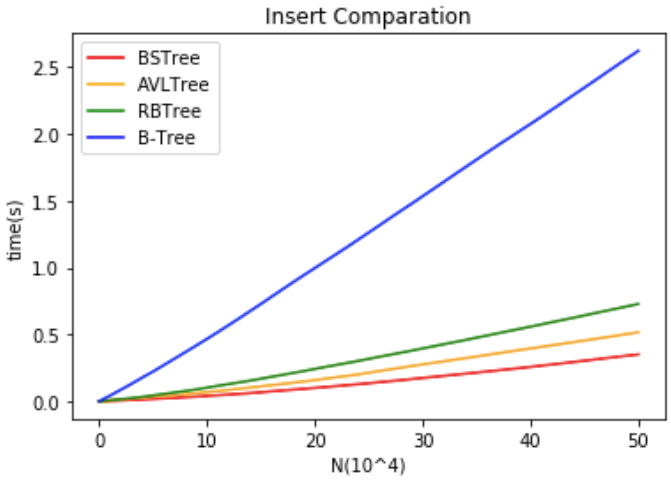


图 9: 3-插入

搜索

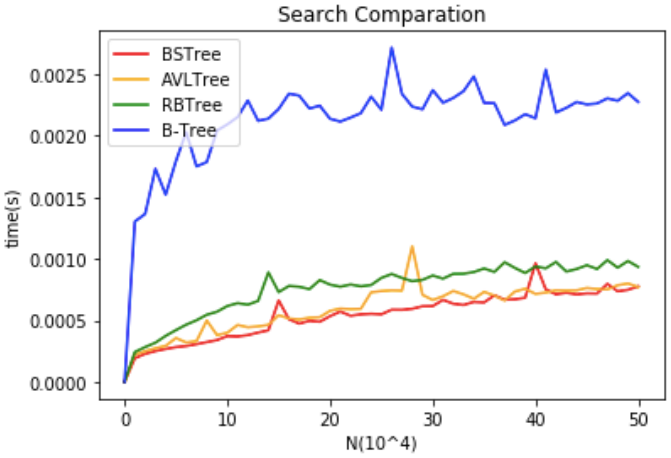


图 10: 3-搜索

删除

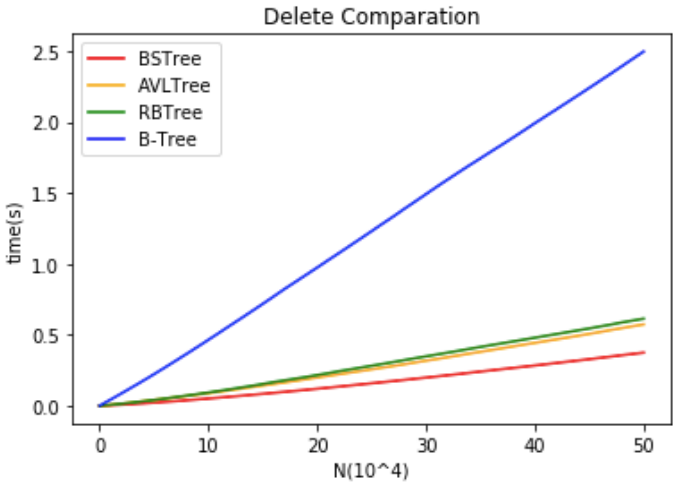


图 11: 3-删除

4 分析和结论

4.1 BSTree

(1) 普通二叉搜索树在升序插入时候的时间复杂度是灾难性的。因为树的高度是 O(n)，所以每次插入、搜索的复杂度都会为 O(n)，n 次操作的时间复杂度可以达到 O(n^2)。

(2) 当插入序列随机时，BSTree 的高度维持在 O(logn) 左右，这时由于它没有 AVL 树和 RB 树用于维持平衡的操作，其插入和删除的速度会比 AVL 树和 RB 树略快。

(3) 当插入序列为升序并且按照插入顺序删除时 (即第一种情况),BSTree 虽然插入的时间复杂度非常高，但是由于没有平衡过程，在删除时几乎不需要在搜索上花费时间，因此在第一种情况下 BSTree 删除的速度非常快。

4.2 AVLTree 和 RBTree

(1) 在三个数据集上，AVL 树和 RB 树的效率比较统一，两者消耗时间也很接近，并且整体上优于 BSTree 和 B-树。

(2) 虽然 AVL 树和 RB 树的树高都是 O(logn) 量级，但是 AVL 树的常数大约为 1.44，RB 树的常数大约为 2，因此在搜索时 AVL 树的时间会略小于 RB 树。这从三组数据的图表中可以看出。

4.3 B-Tree

(1) 除去 BSTree 在顺序插入的时候复杂度非常高，B-Tree 是四棵树中插入、查找、删除性能最低的。这是因为 B-Tree 每个节点中的元素是以线性表的形式组织的，因此实际每次插入、查找、删除时需要遍历节点中的线性表，单次操作时间复杂度可以到 O($mlog_mn$)。

(2) B-Tree 的搜索曲线不符合正常的逐渐递增模式，而是维持在一个较高的值上下波动。

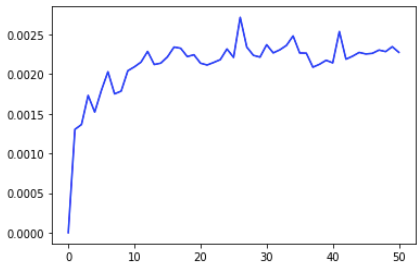


图 12: B-Tree 搜索模式图

这是因为搜索单次执行的时间复杂度为 O($mlog_mn$)，共执行 1000 次，实际对于时间复杂度贡献较大的为 m，所以随着 n 增大时间复杂度变化较小。而在插入和删除中，一共执行 n 次，对于时间复杂度贡献较大的为 n，因此符合逐渐递增的模式。

(3) 可以对于 B-Tree 进行改进，使其复杂度接近 AVL 树和 RB 树。在每个节点中使用 Treap 数据结构而不是线性表来记录节点中的元素，这样查找每个值的前驱、后缀以及插入和删除都可以在 O(logm) 的复杂度内完成，这时 B-Tree 的单次操作复杂度可以降到 O(logn)。

附录：cpp 代码和 python 代码

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <ctime>
using namespace std;

// BSTree
struct BSNode
{
    int val;
    BSNode *left, *right;

    BSNode(): val(0), left(NULL), right(NULL) {}
};
class BSTree
{
public:
    BSNode *root;
    BSTree() { root = NULL; }

private:
    int findMinMax(BSNode *&u);

    BSNode* searchNode(int x);
    void insertNode(int x);
    void delNode(int x);

public:
    BSNode* search(int x) { return searchNode(x); }
    void insert(int x) { insertNode(x); }
    void del(int x) { delNode(x); }
};
int BSTree::findMinMax(BSNode *&u)
{
    if(u->left == NULL) return u->val;
    BSNode *v = u->left;
    while(v->right != NULL) v = v->right;
    return v->val;
}
BSNode* BSTree::searchNode(int x)
{
    BSNode *v = root;
    while(v->val != x)
    {
        if(v->val > x)
        {
            if(v->left != NULL) v = v->left;
            else return NULL;
        }
        else if(v->val < x)
        {
            if(v->right != NULL) v = v->right;
            else return NULL;
        }
    }
    return v;
}
void BSTree::insertNode(int x)
{
    if(root == NULL)
    {
        root = new BSNode; root->val = x;
        return;
    }
    BSNode *v = root;
    while(v->val != x)
    {
        if(v->val > x)
```

```
        {
            if(v->left != NULL) v = v->left;
            else
            {
                v->left = new BSNode, v->left->val = x;
                return;
            }
        }
        else if(v->val < x)
        {
            if(v->right != NULL) v = v->right;
            else
            {
                v->right = new BSNode, v->right->val = x;
                return;
            }
        }
    }
}
void BSTree::delNode(int x)
{
    BSNode *v = root, *fa;

    while(x != v->val)
    {
        if(v->val > x)
        {
            if(v->left == NULL) return;
            else fa = v, v = v->left;
        }
        else if(v->val < x)
        {
            if(v->right == NULL) return;
            else fa = v, v = v->right;
        }
    }

    if(v->left == NULL && v->right == NULL)
    {
        if(v == this->root) root = NULL;
        else
        {
            if(fa->left == v) fa->left = NULL;
            else fa->right = NULL;
        }
    }
    else if(v->left != NULL && v->right == NULL)
    {
        if(v == this->root) root = v->left;
        else
        {
            if(fa->left == v) fa->left = v->left;
            else fa->right = v->left;
        }
    }
    else if(v->left == NULL && v->right != NULL)
    {
        if(v == this->root) root = v->right;
        else
        {
            if(fa->left == v) fa->left = v->right;
            else fa->right = v->right;
        }
    }
    else
    {
        int pre = findMinMax(v);
        delNode(pre); v->val = pre;
    }
}
```

```
// AVLTree
struct AVLNode
{
    int val, lh, rh;
    AVLNode *left, *right;

    AVLNode(): val(0), lh(0), rh(0), left(NULL), right(NULL) {}
};
class AVLTree
{
public:
    AVLNode *root;
    AVLTree() { root = NULL; }

private:
    void LL(AVLNode *&u);
    void RR(AVLNode *&u);
    void LR(AVLNode *&u);
    void RL(AVLNode *&u);
    int findMinMax(AVLNode *&u);

    AVLNode* searchNode(AVLNode *&u, int x);
    void insertNode(AVLNode *&u, int x);
    AVLNode* delNode(AVLNode *&u, AVLNode *&fa, int x);

public:
    AVLNode* search(int x) { return searchNode(root, x); }
    void insert(int x) { insertNode(root, x); }
    AVLNode* del(int x) { AVLNode *fa = new AVLNode; return delNode(root, fa, x); }
};
void AVLTree::LL(AVLNode *&u)
{
    AVLNode *ul = u->left;
    u->left = ul->right; u->lh = ul->rh;
    ul->right = u; ul->rh = max(u->lh, u->rh)+1;
    u = ul;
}
void AVLTree::RR(AVLNode *&u)
{
    AVLNode *ur = u->right;
    u->right = ur->left; u->rh = ur->lh;
    ur->left = u; ur->lh = max(u->lh, u->rh)+1;
    u = ur;
}
void AVLTree::LR(AVLNode *&u)
{
    AVLTree::RR(u->left);
    AVLTree::LL(u);
}
void AVLTree::RL(AVLNode *&u)
{
    AVLTree::LL(u->right);
    AVLTree::RR(u);
}
int AVLTree::findMinMax(AVLNode *&u)
{
    if(u->left == NULL) return u->val;
    AVLNode *v = u->left;
    while(v->right != NULL) v = v->right;
    return v->val;
}
AVLNode* AVLTree::searchNode(AVLNode *&u, int x)
{
    if(u->val == x) return u;
    if(u->val > x && u->left != NULL)
    {
        AVLNode *v = AVLTree::searchNode(u->left, x);
        if(v != NULL) return v;
```

```
    }
    if(u->val < x && u->right != NULL)
    {
        AVLNode *v = AVLTree::searchNode(u->right, x);
        if(v != NULL) return v;
    }
    return NULL;
}
void AVLTree::insertNode(AVLNode *&u, int x)
{
    if(u == NULL)
    {
        u = new AVLNode; u->val = x;
        return;
    }
    if(x < u->val)
    {
        if(u->left == NULL)
        {
            AVLNode *v = new AVLNode; v->val = x;
            u->left = v; u->lh = 1;
        }
        else
        {
            insertNode(u->left, x);
            u->lh = max(u->left->lh, u->left->rh) + 1;
            if(u->lh > u->rh + 1)
            {
                if(x > u->left->val) LR(u);
                else if(x < u->left->val) LL(u);
            }
        }
    }
    else if(x > u->val)
    {
        if(u->right == NULL)
        {
            AVLNode *v = new AVLNode; v->val = x;
            u->right = v; u->rh = 1;
        }
        else
        {
            insertNode(u->right, x);
            u->rh = max(u->right->lh, u->right->rh) + 1;
            if(u->rh > u->lh + 1)
            {
                if(x < u->right->val) RL(u);
                else if(x > u->right->val) RR(u);
            }
        }
    }
}
AVLNode* AVLTree::delNode(AVLNode *&u, AVLNode *&fa, int x)
{
    AVLNode *res;
    if(x < u->val)
    {
        if(u->left == NULL) return NULL;
        else
        {
            res = AVLTree::delNode(u->left, u, x);
            if(u->left != NULL) u->lh = max(u->left->lh, u->left->rh) + 1;
            else u->lh = 0;

            if(u->rh - u->lh > 1)
            {
                if(u->right->lh > u->right->rh) RL(u);
                else RR(u);
            }
            return res;
```

```
        }
    }
    else if(x > u->val)
    {
        if(u->right == NULL) return NULL;
        else
        {
            res = AVLTree::delNode(u->right, u, x);
            if(u->right != NULL) u->rh = max(u->right->lh, u->right->rh) + 1;
            else u->rh = 0;

            if(u->lh - u->rh > 1)
            {
                if(u->left->rh > u->left->lh) LR(u);
                else LL(u);
            }
            return res;
        }
    }
    else
    {
        res = u;
        if(u->left == NULL && u->right == NULL)
        {
            if(u == this->root) root = NULL;
            else
            {
                if(fa->left == u) fa->left = NULL;
                else fa->right = NULL;
            }
            return res;
        }
        else if(u->left != NULL && u->right == NULL)
        {
            if(u == this->root) root = u->left;
            else
            {
                if(fa->left == u) fa->left = u->left;
                else fa->right = u->left;
            }
            return res;
        }
        else if(u->left == NULL && u->right != NULL)
        {
            if(u == this->root) root = u->right;
            else
            {
                if(fa->left == u) fa->left = u->right;
                else fa->right = u->right;
            }
            return res;
        }
        else
        {
            int pre = findMinMax(u);
            u->val = pre;
            AVLTree::delNode(u->left, u, pre);
            if(u->left != NULL) u->lh = max(u->left->lh, u->left->rh) + 1;
            else u->lh = 0;

            if(u->rh - u->lh > 1)
            {
                if(u->right->lh > u->right->rh) RL(u);
                else RR(u);
            }
            return res;
        }
    }
}
```

```
// RBTree
struct RBNode
{
    int val, col;   // 黑色为1，红色为0
    RBNode *left, *right;

    RBNode(): val(0), col(1), left(NULL), right(NULL) {}
};
class RBTree
{
public:
    RBNode *root;
    RBTree() { root = NULL; }

private:
    void LL(RBNode *&u);
    void RR(RBNode *&u);
    void LR(RBNode *&u);
    void RL(RBNode *&u);
    inline bool isBlack(RBNode *&u) { return u == NULL || u->col == 1; }
    inline void r(RBNode *&u) { if(u != NULL) u->col = 0; }
    inline void b(RBNode *&u) { if(u != NULL) u->col = 1; }
    void insertFix(RBNode *&u, RBNode *&pu, RBNode *&gu);
    int findSuf(RBNode *&u);
    int delFix(RBNode *&y, RBNode *&py);   // 如果需要回溯则返回1

    RBNode* searchNode(RBNode *&u, int x);
    void insertNode(RBNode *&u, RBNode *&pu, RBNode *&gu, int x);
    RBNode* delNode(RBNode *&y, RBNode *&py, int x, int &flag);

public:
    RBNode* search(int x) { return searchNode(root, x); }
    void insert(int x) { RBNode *pu = new RBNode, *gu = new RBNode; insertNode(root, pu, gu,
x); }
    RBNode* del(int x) { RBNode *py = new RBNode; int flag = 0; return delNode(root, py, x,
flag); }
};
void RBTree::LL(RBNode *&u)
{
    RBNode *ul = u->left;
    u->left = ul->right; ul->right = u;
    u = ul;
}
void RBTree::RR(RBNode *&u)
{
    RBNode *ur = u->right;
    u->right = ur->left; ur->left = u;
    u = ur;
}
void RBTree::LR(RBNode *&u)
{
    RBTree::RR(u->left);
    RBTree::LL(u);
}
void RBTree::RL(RBNode *&u)
{
    RBTree::LL(u->right);
    RBTree::RR(u);
}
void RBTree::insertFix(RBNode *&u, RBNode *&pu, RBNode *&gu)
{
    if(isBlack(u) || isBlack(pu)) return;
    if(gu->left == pu)
    {
        if(!isBlack(gu->right))   // LLr和LRr
            r(gu), b(pu), b(gu->right);
        else
        {
```

```
            if(pu->left == u)   // LLb
                r(gu), b(pu), LL(gu);
            else   // LRb
                b(u), r(gu), LR(gu);
        }
    }
    else
    {
        if(!isBlack(gu->left))   // RLr和RRr
            r(gu), b(gu->left), b(pu);
        else
        {
            if(pu->left == u)   // RLb
                b(u), r(gu), RL(gu);
            else   // RRb
                r(gu), b(pu), RR(gu);
        }
    }
    if(gu == this->root) b(gu);
}
int RBTree::findSuf(RBNode *&u)
{
    if(u->right == NULL) return u->val;
    RBNode *v = u->right;
    while(v->left != NULL) v = v->left;
    return v->val;
}
int RBTree::delFix(RBNode *&y, RBNode *&py)
{
    if(!isBlack(y)) { b(y); return 0; }
    if(y == py->right)
    {
        RBNode *&v = py->left;
        if(isBlack(v))
        {
            if(isBlack(v->left) && isBlack(v->right))   // Rb0
            {
                if(isBlack(py)) { r(v); return 1; }
                else { r(v); b(py); return 0; }
            }
            else if(isBlack(v->right) && !isBlack(v->left))   // Rb11
            {
                v->col = py->col; b(py); b(v->left);
                LL(py); return 0;
            }
            else   // Rb12和Rb2
            {
                RBNode *&w = v->right; w->col = py->col; b(py);
                LR(py); return 0;
            }
        }
        else
        {
            RBNode *&w = v->right;
            if(isBlack(w->left) && isBlack(w->right))   // Rr0
            {
                b(v); r(w);
                LL(py); return 0;
            }
            else if(isBlack(w->right) && !isBlack(w->left))   // Rr11
            {
                b(w->left);
                LR(py); return 0;
            }
            else   // Rr12和Rr2
            {
                RBNode *&x = w->right; b(x);
                RR(w); RR(v); LL(py); return 0;
            }
```

```
        }
    }
    else
    {
        RBNode *&v = py->right;
        if(isBlack(v))
        {
            if(isBlack(v->left) && isBlack(v->right))   // Lb0
            {
                if(isBlack(py)) { r(v); return 1; }
                else { r(v); b(py); return 0; }
            }
            else if(isBlack(v->left) && !isBlack(v->right))   // Lb11
            {
                v->col = py->col; b(v->right); b(py);
                RR(py); return 0;
            }
            else   // Lb12和Lb2
            {
                RBNode *&w = v->left; w->col = py->col; b(py);
                RL(py); return 0;
            }
        }
        else
        {
            RBNode *&w = v->left;
            if(isBlack(w->left) && isBlack(w->right))   // Lr0
            {
                b(v); r(w);
                RR(py); return 0;
            }
            else if(isBlack(w->left) && !isBlack(w->right))   // Lr11
            {
                b(w->right);
                RL(py); return 0;
            }
            else   // Lr12和Lr2
            {
                RBNode *&x = w->left; b(x);
                LL(w); LL(v); RR(py); return 0;
            }
        }
    }
}
RBNode* RBTree::searchNode(RBNode *&u, int x)
{
    if(u->val == x) return u;
    if(u->val > x && u->left != NULL)
    {
        RBNode *v = RBTree::searchNode(u->left, x);
        if(v != NULL) return v;
    }
    if(u->val < x && u->right != NULL)
    {
        RBNode *v = RBTree::searchNode(u->right, x);
        if(v != NULL) return v;
    }
    return NULL;
}
void RBTree::insertNode(RBNode *&u, RBNode *&pu, RBNode *&gu, int x)
{
    if(u == NULL)
    {
        u = new RBNode; u->val = x;
        u->col = u==this->root ? 1 : 0;
        insertFix(u, pu, gu);
        return;
    }
    if(x < u->val)
```

```
    {
        insertNode(u->left, u, pu, x);
        insertFix(u, pu, gu);
    }
    else if(x > u->val)
    {
        insertNode(u->right, u, pu, x);
        insertFix(u, pu, gu);
    }
}
RBNode* RBTree::delNode(RBNode *&y, RBNode *&py, int x, int &flag)
{
    RBNode *res;
    if(x < y->val)
    {
        if(y->left == NULL) return NULL;
        res = RBTree::delNode(y->left, y, x, flag);
        if(y == this->root) b(root);
        else if(flag) flag = delFix(y, py);
        return res;
    }
    else if(x > y->val)
    {
        if(y->right == NULL) return NULL;
        res = RBTree::delNode(y->right, y, x, flag);
        if(y == this->root) b(root);
        else if(flag) flag = delFix(y, py);
        return res;
    }
    else
    {
        res = y;
        if(y->left == NULL && y->right == NULL)
        {
            if(isBlack(y))
            {
                if(y == this->root) root = NULL;
                else
                {
                    if(py->left == y) py->left = NULL;
                    else py->right = NULL;
                    flag = delFix(y, py);
                }
            }
            else
            {
                if(py->left == y) py->left = NULL;
                else py->right = NULL;
            }
            return res;
        }
        else if(y->left != NULL && y->right == NULL)
        {
            if(y == this->root) root = y->left, b(root);
            else
            {
                if(py->left == y) py->left = y->left;
                else py->right = y->left;
                if(!isBlack(y)) b(y);
                else flag = delFix(y, py);
            }
            return res;
        }
        else if(y->left == NULL && y->right != NULL)
        {
            if(y == this->root) root = y->right, b(root);
            else
            {
                if(py->left == y) py->left = y->right;
                else py->right = y->right;
```

```
                if(!isBlack(y)) b(y);
                else flag = delFix(y, py);
            }
            return res;
        }
        else
        {
            int suf = findSuf(y);
            y->val = suf;
            RBTree::delNode(y->right, y, suf, flag);
            if(y == this->root) b(root);
            else if(flag) flag = delFix(y, py);
            return res;
        }
    }
}


// BTree
struct BNode
{
    bool leaf; int s;
    int e[600];
    BNode *c[600];

    BNode(): leaf(true), s(0) {}

    void insert_ele(int i, int x, BNode *xr)
    {
        this->s++;
        for(int j = s; j > i; j--) e[j] = e[j-1], c[j] = c[j-1];
        e[i] = x; c[i] = xr;
    }
};
class BTree
{
public:
    int m;   // m阶B树
    BNode *root;
    BTree(int m) { this->m = m; root = NULL; }

private:
    void push_up(BNode *&u, BNode *&fa, int id);   // fa->c[id]=u
    int findPre(BNode *&u);
    void push_down(BNode *&u, BNode *&fa, int id);

    BNode* searchNode(BNode *&u, int x);
    void insertNode(BNode *&u, BNode *&fa, int id, int x);
    void delNode(BNode *&u, BNode *&fa, int id, int x);


public:
    BNode* search(int x) { return searchNode(root, x); }
    void insert(int x) { BNode *fa = new BNode; insertNode(root, fa, 0, x); }
    void del(int x) { BNode *fa = new BNode; delNode(root, fa, 0, x); }
};
void BTree::push_up(BNode *&u, BNode *&fa, int id)
{
    if(u->s < m) return;
    if(u == root)
    {
        BNode *l = new BNode, *r = new BNode, *p = new BNode;
        l->leaf = r->leaf = u->leaf;
        int d = (m-1)/2 + 1;

        l->s = d-1; l->c[0] = u->c[0];
        for(int i = 1; i <= d-1; i++) l->e[i] = u->e[i], l->c[i] = u->c[i];
        r->s = m-d; r->c[0] = u->c[d];
        for(int i = 1; i <= m-d; i++) r->e[i] = u->e[d+i], r->c[i] = u->c[d+i];
```

```
        p->leaf = false; p->s = 1;
        p->e[1] = u->e[d]; p->c[0] = l; p->c[1] = r;
        root = p;
    }
    else
    {
        BNode *l = new BNode, *r = new BNode;
        l->leaf = r->leaf = u->leaf;
        int d = (m-1)/2 + 1;

        l->s = d-1; l->c[0] = u->c[0];
        for(int i = 1; i <= d-1; i++) l->e[i] = u->e[i], l->c[i] = u->c[i];
        r->s = m-d; r->c[0] = u->c[d];
        for(int i = 1; i <= m-d; i++) r->e[i] = u->e[d+i], r->c[i] = u->c[d+i];
        int x = u->e[d]; u = l;

        fa->insert_ele(id+1, x, r);
    }
}
int BTree::findPre(BNode *&u)
{
    if(u->leaf) return u->e[u->s];
    else return findPre(u->c[u->s]);
}
void BTree::push_down(BNode *&u, BNode *&fa, int id)
{
    int d = (m-1)/2 + 1;
    if(u == root || u == NULL || fa == NULL || u->s >= d-1) return;

    BNode *&v = id==0 ? fa->c[id+1] : fa->c[id-1];
    if(v->s > d-1)
    {
        if(id)
        {
            u->s++;
            for(int j = u->s; j > 1; j--) u->e[j] = u->e[j-1], u->c[j] = u->c[j-1];
            u->e[1] = fa->e[id]; u->c[1] = u->c[0]; u->c[0] = v->c[v->s];
            fa->e[id] = v->e[v->s];
            v->c[v->s] = NULL; v->e[v->s] = 0; v->s--;
        }
        else
        {
            u->s++;
            u->e[u->s] = fa->e[id+1]; u->c[u->s] = v->c[0];
            fa->e[id+1] = v->e[1];
            v->c[0] = v->c[1];
            for(int j = 1; j < v->s; j++) v->e[j] = v->e[j+1], v->c[j] = v->c[j+1];
            v->e[v->s] = 0; v->c[v->s] = NULL; v->s--;
        }
    }
    else
    {
        if(id)
        {
            BNode *w = new BNode;
            w->leaf = v->leaf;

            w->s = v->s + 1 + u->s; w->c[0] = v->c[0];
            for(int i = 1; i <= v->s; i++) w->e[i] = v->e[i], w->c[i] = v->c[i];
            w->e[v->s+1] = fa->e[id]; w->c[v->s+1] = u->c[0];
            for(int i = 1; i <= u->s; i++) w->e[v->s+1+i] = u->e[i], w->c[v->s+1+i] = u-
>c[i];

            for(int j = id; j < fa->s; j++) fa->e[j] = fa->e[j+1], fa->c[j] = fa->c[j+1];
            fa->e[fa->s] = 0; fa->c[fa->s] = NULL; fa->s--; fa->c[id-1] = w;

            if(fa == root && fa->s == 0) root = w;
        }
        else
```

```
        {
            BNode *w = new BNode;
            w->leaf = u->leaf;

            w->s = u->s + 1 + v->s; w->c[0] = u->c[0];
            for(int i = 1; i <= u->s; i++) w->e[i] = u->e[i], w->c[i] = u->c[i];
            w->e[u->s+1] = fa->e[id+1]; w->c[u->s+1] = v->c[0];
            for(int i = 1; i <= v->s; i++) w->e[u->s+1+i] = v->e[i], w->c[u->s+1+i] = v-
>c[i];

            for(int j = id+1; j < fa->s; j++) fa->e[j] = fa->e[j+1], fa->c[j] = fa->c[j+1];
            fa->e[fa->s] = 0; fa->c[fa->s] = NULL; fa->s--; fa->c[id] = w;

            if(fa == root && fa->s == 0) root = w;
        }
    }
}
BNode* BTree::searchNode(BNode *&u, int x)
{
    for(int i = 1; i <= u->s; i++)
    {
        if(u->e[i] == x) return u;
        else if(u->e[i] > x && u->c[i-1] != NULL) return searchNode(u->c[i-1], x);
    }
    if(u->e[u->s] < x && u->c[u->s] != NULL) return searchNode(u->c[u->s], x);
    return NULL;
}
void BTree::insertNode(BNode *&u, BNode *&fa, int id, int x)
{
    if(u == NULL)
    {
        u = new BNode; u->s = 1; u->e[1] = x;
        return;
    }
    for(int i = 1; i <= u->s; i++)
    {
        if(u->e[i] == x) return;
        else if(u->e[i] > x)
        {
            if(u->leaf) { u->insert_ele(i, x, NULL); push_up(u, fa, id); }
            else { insertNode(u->c[i-1], u, i-1, x); push_up(u, fa, id); }
            return;
        }
    }
    if(u->e[u->s] < x)
    {
        if(u->leaf) { u->insert_ele(u->s+1, x, NULL); push_up(u, fa, id); }
        else { insertNode(u->c[u->s], u, u->s, x); push_up(u, fa, id); }
    }
}
void BTree::delNode(BNode *&u, BNode *&fa, int id, int x)
{
    if(u == NULL) return;
    for(int i = 1; i <= u->s; i++)
    {
        if(u->e[i] == x)
        {
            if(!u->leaf)
            {
                int pre = findPre(u->c[i-1]); u->e[i] = pre;
                delNode(u->c[i-1], u, i-1, pre);
                push_down(u, fa, id);
            }
            else
            {
                for(int j = i; j < u->s; j++) u->e[j] = u->e[j+1];
                u->e[u->s] = 0; u->s--;
                push_down(u, fa, id);
            }
            if(u == root && u->s == 0) root = NULL;
```

```
            return;
        }
        else if(u->e[i] > x)
        {
            if(u->leaf) return;
            else
            {
                delNode(u->c[i-1], u, i-1, x); push_down(u, fa, id);
                if(u == root && u->s == 0) root = NULL;
                return;
            }
        }
    }
    if(u->e[u->s] < x)
    {
        if(u->leaf) return;
        else
        {
            delNode(u->c[u->s], u, u->s, x);
            push_down(u, fa, id);
            return;
        }
    }
}

BSTree *bs;
AVLTree *avl;
RBTree *rb;
BTree *bt;

const int ms = 1000;
const int maxn = 500005;
int in[maxn], de[maxn], se[ms], key[maxn], rd[maxn];

bool cmp(const int i, const int j) { return rd[i] < rd[j]; }

int n = 500000;

clock_t s, sse;
double ein[52], ese[52], ede[52];

void solve()
{
    s = clock();
    for(int i = 0; i < n; i++)
    {
        bs->insert(in[i]);
        if((i+1) % 10000 == 0)
        {
            ein[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
            s = clock();
            sse = clock();
            for(int j = 0; j < ms; j++)
                bs->search(se[j]);
            ese[(i+1)/10000] = (double)(clock()-sse) / CLOCKS_PER_SEC;
        }
    }
    for(int i = 1; i <= 50; i++)
        ein[i] += ein[i-1];
    s = clock();
    for(int i = 0; i < n; i++)
    {
        bs->del(de[key[i]]);
        if((i+1) % 10000 == 0) ede[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
    }
    for(int i = 0; i <= 50; i++)
        printf("%.6lf,%.6lf,%.6lf\n", ein[i], ese[i], ede[50]-ede[50-i]);
    cout << endl;
```

```
s = clock();
for(int i = 0; i < n; i++)
{
    avl->insert(in[i]);
    if((i+1) % 10000 == 0)
    {
        ein[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
        s = clock();
        sse = clock();
        for(int j = 0; j < ms; j++)
            avl->search(se[j]);
        ese[(i+1)/10000] = (double)(clock()-sse) / CLOCKS_PER_SEC;
    }
}
for(int i = 1; i <= 50; i++)
    ein[i] += ein[i-1];
s = clock();
for(int i = 0; i < n; i++)
{
    avl->del(de[key[i]]);
    if((i+1) % 10000 == 0) ede[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
}
for(int i = 0; i <= 50; i++)
    printf("%.6lf,%.6lf,%.6lf\n", ein[i], ese[i], ede[50]-ede[50-i]);
cout << endl;

s = clock();
for(int i = 0; i < n; i++)
{
    rb->insert(in[i]);
    if((i+1) % 10000 == 0)
    {
        ein[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
        s = clock();
        sse = clock();
        for(int j = 0; j < ms; j++)
            rb->search(se[j]);
        ese[(i+1)/10000] = (double)(clock()-sse) / CLOCKS_PER_SEC;
    }
}
for(int i = 1; i <= 50; i++)
    ein[i] += ein[i-1];
s = clock();
for(int i = 0; i < n; i++)
{
    rb->del(de[key[i]]);
    if((i+1) % 10000 == 0) ede[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
}
for(int i = 0; i <= 50; i++)
    printf("%.6lf,%.6lf,%.6lf\n", ein[i], ese[i], ede[50]-ede[50-i]);
cout << endl;

s = clock();
for(int i = 0; i < n; i++)
{
    bt->insert(in[i]);
    if((i+1) % 10000 == 0)
    {
        ein[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
        s = clock();
        sse = clock();
        for(int j = 0; j < ms; j++)
            bt->search(se[j]);
        ese[(i+1)/10000] = (double)(clock()-sse) / CLOCKS_PER_SEC;
    }
}
for(int i = 1; i <= 50; i++)
    ein[i] += ein[i-1];
s = clock();
for(int i = 0; i < n; i++)
```

```
    {
        bt->del(de[key[i]]);
        if((i+1) % 10000 == 0) ede[(i+1)/10000] = (double)(clock()-s) / CLOCKS_PER_SEC;
    }
    for(int i = 0; i <= 50; i++)
        printf("%.6lf,%.6lf,%.6lf\n", ein[i], ese[i], ede[50]-ede[50-i]);
    cout << endl;
}

void order()   // 插入n个数，查找ms个，顺序删除
{
    srand(time(NULL));
    for(int i = 0; i < n; i++)
        in[i] = rand();
    sort(in, in+n);
    for(int i = 0; i < n; i++)
        de[i] = in[i], key[i] = i;
    for(int i = 0; i < ms; i++)
        se[i] = rand();

    solve();
}

void invorder()   // 插入n个数，查找ms个，逆序删除
{
    srand(time(NULL));
    for(int i = 0; i < n; i++)
        in[i] = rand();
    sort(in, in+n);
    for(int i = 0; i < n; i++)
        de[i] = in[i], key[i] = n-i-1;
    for(int i = 0; i < ms; i++)
        se[i] = rand();

    solve();
}

void rdorder()   // 插入n个数，查找ms个，随机删除
{
    srand(time(NULL));
    for(int i = 0; i < n; i++)
        in[i] = rand();
    for(int i = 0; i < n; i++)
        de[i] = in[i], key[i] = i, rd[i] = rand();
    sort(key, key+n, cmp);
    for(int i = 0; i < ms; i++)
        se[i] = rand();

    solve();
}


signed main()
{
    bs = new BSTree; avl = new AVLTree; rb = new RBTree; bt = new BTree(512);

    freopen("1/1-1.csv", "w", stdout);
    order();
    freopen("1/1-2.csv", "w", stdout);
    order();
    freopen("1/1-3.csv", "w", stdout);
    order();

    freopen("2/2-1.csv", "w", stdout);
    invorder();
    freopen("2/2-2.csv", "w", stdout);
    invorder();
    freopen("2/2-3.csv", "w", stdout);
    invorder();
```

```
    freopen("3/3-1.csv", "w", stdout);
    rdorder();
    freopen("3/3-2.csv", "w", stdout);
    rdorder();
    freopen("3/3-3.csv", "w", stdout);
    rdorder();
}
```

1

December 6, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
data1 = pd.read_csv("1-1.csv", header=None, names=["insert","search","delete"])
data2 = pd.read_csv("1-2.csv", header=None, names=["insert","search","delete"])
data3 = pd.read_csv("1-3.csv", header=None, names=["insert","search","delete"])
```

[3]:
```
in1, se1, de1 = data1["insert"], data1["search"], data1["delete"]
in2, se2, de2 = data2["insert"], data2["search"], data2["delete"]
in3, se3, de3 = data3["insert"], data3["search"], data3["delete"]
```

[4]:
```
ins = [(in1[i]+in2[i]+in3[i])/3 for i in range(len(in1))]
se = [(se1[i]+se2[i]+se3[i])/3 for i in range(len(se1))]
de = [(de1[i]+de2[i]+de3[i])/3 for i in range(len(de1))]
```

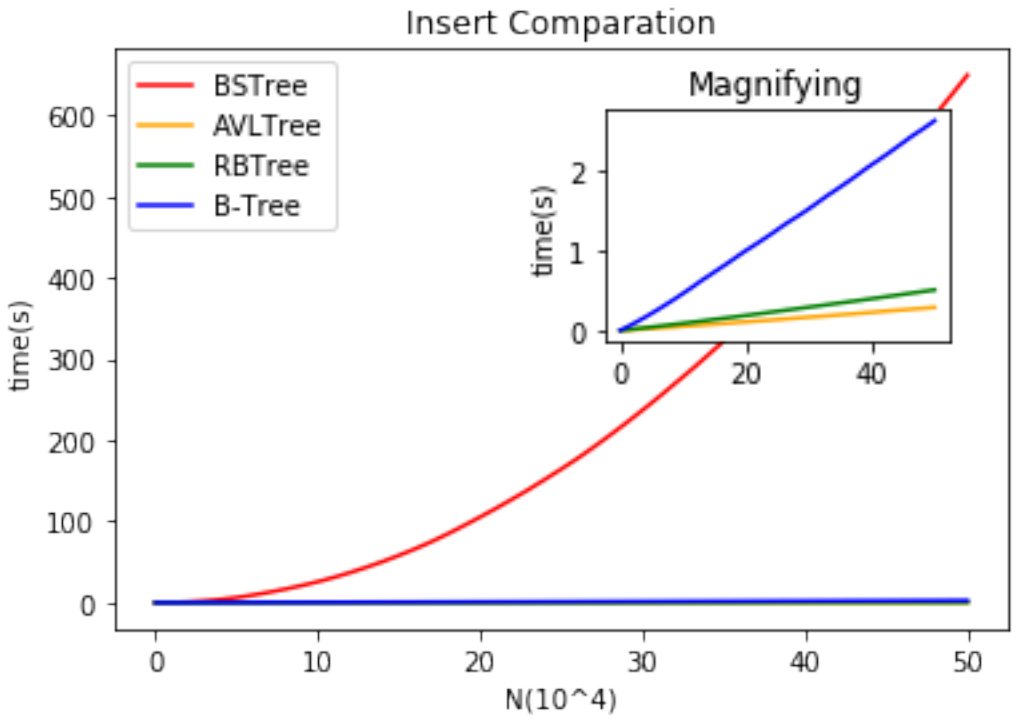
[5]:
```
bs_in, avl_in, rb_in, bt_in = ins[:51], ins[51:102], ins[102:153], ins[153:204]
bs_se, avl_se, rb_se, bt_se = se[:51], se[51:102], se[102:153], se[153:204]
bs_de, avl_de, rb_de, bt_de = de[:51], de[51:102], de[102:153], de[153:204]
```

[6]:
```
n = range(0, 51, 1)
```

[7]:
```
plt.title('Insert Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_in, c='red', label="bs")
l2, = plt.plot(n, avl_in, c='orange', label="avl")
l3, = plt.plot(n, rb_in, c='green', label="rb")
l4, = plt.plot(n, bt_in, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')

plt.axes([0.55, 0.5, 0.3, 0.3], zorder=2)
plt.title('Magnifying')
plt.ylabel("time(s)")
plt.plot(n, avl_in, c='orange', label="avl")
plt.plot(n, rb_in, c='green', label="rb")
```

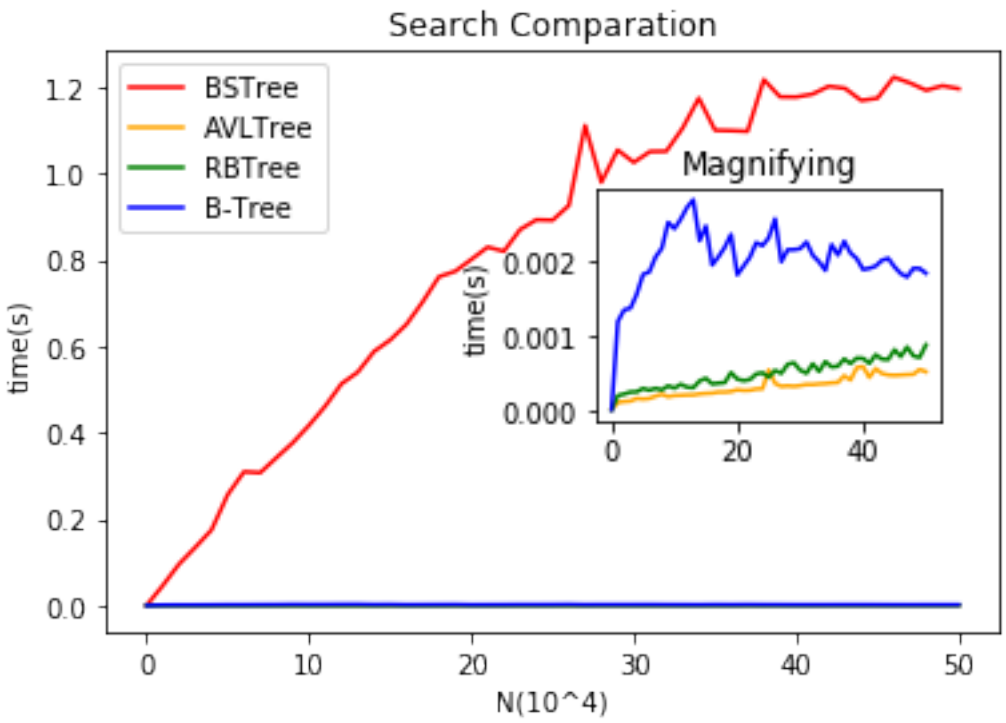
```
plt.plot(n, bt_in, c='blue', label="bt")
```

[7]: [<matplotlib.lines.Line2D at 0x7ffac23b7290>]



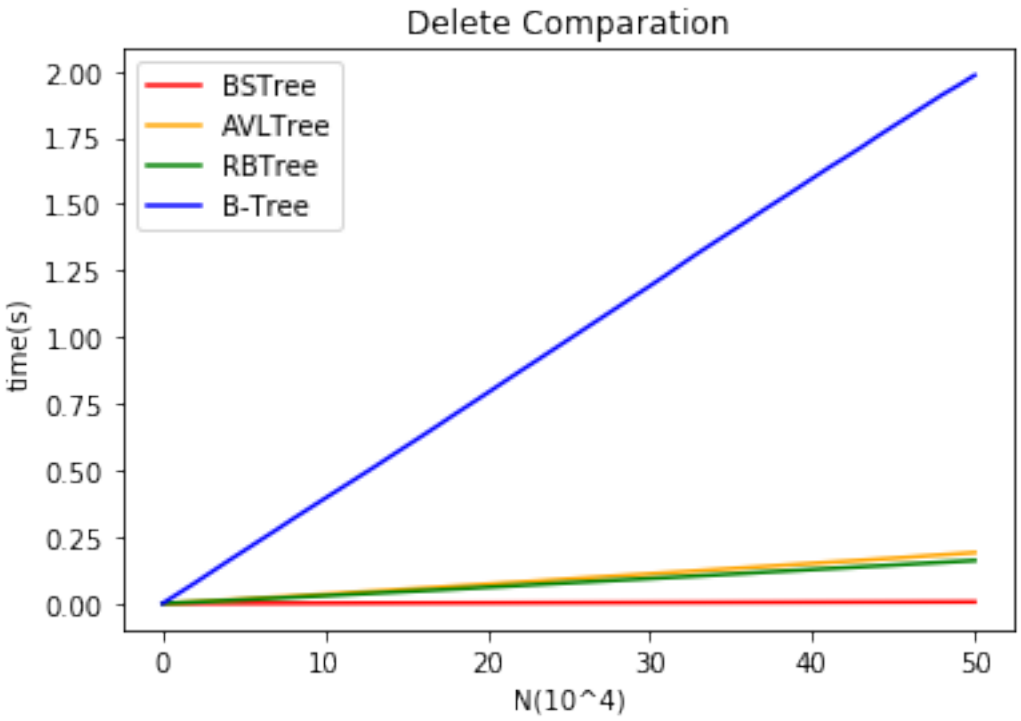
[8]:
```
plt.title('Search Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_se, c='red', label="bs")
l2, = plt.plot(n, avl_se, c='orange', label="avl")
l3, = plt.plot(n, rb_se, c='green', label="rb")
l4, = plt.plot(n, bt_se, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
 ↪'B-Tree'], loc='upper left')

plt.axes([0.55, 0.4, 0.3, 0.3], zorder=2)
plt.title('Magnifying')
plt.ylabel("time(s)")
plt.plot(n, avl_se, c='orange', label="avl")
plt.plot(n, rb_se, c='green', label="rb")
plt.plot(n, bt_se, c='blue', label="bt")
```

[8]: [<matplotlib.lines.Line2D at 0x7ffac26ca5d0>]



[9]:
```
plt.title('Delete Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_de, c='red', label="bs")
l2, = plt.plot(n, avl_de, c='orange', label="avl")
l3, = plt.plot(n, rb_de, c='green', label="rb")
l4, = plt.plot(n, bt_de, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')
```

[9]: <matplotlib.legend.Legend at 0x7ffac2732310>


Delete Comparation
BSTree
AVLTree
RBTree
B-Tree
time(s)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10
20
30
40
50
N(10^4)

2

December 6, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
data1 = pd.read_csv("2-1.csv", header=None, names=["insert","search","delete"])
data2 = pd.read_csv("2-2.csv", header=None, names=["insert","search","delete"])
data3 = pd.read_csv("2-3.csv", header=None, names=["insert","search","delete"])
```

[3]:
```
in1, se1, de1 = data1["insert"], data1["search"], data1["delete"]
in2, se2, de2 = data2["insert"], data2["search"], data2["delete"]
in3, se3, de3 = data3["insert"], data3["search"], data3["delete"]
```

[4]:
```
ins = [(in1[i]+in2[i]+in3[i])/3 for i in range(len(in1))]
se = [(se1[i]+se2[i]+se3[i])/3 for i in range(len(se1))]
de = [(de1[i]+de2[i]+de3[i])/3 for i in range(len(de1))]
```

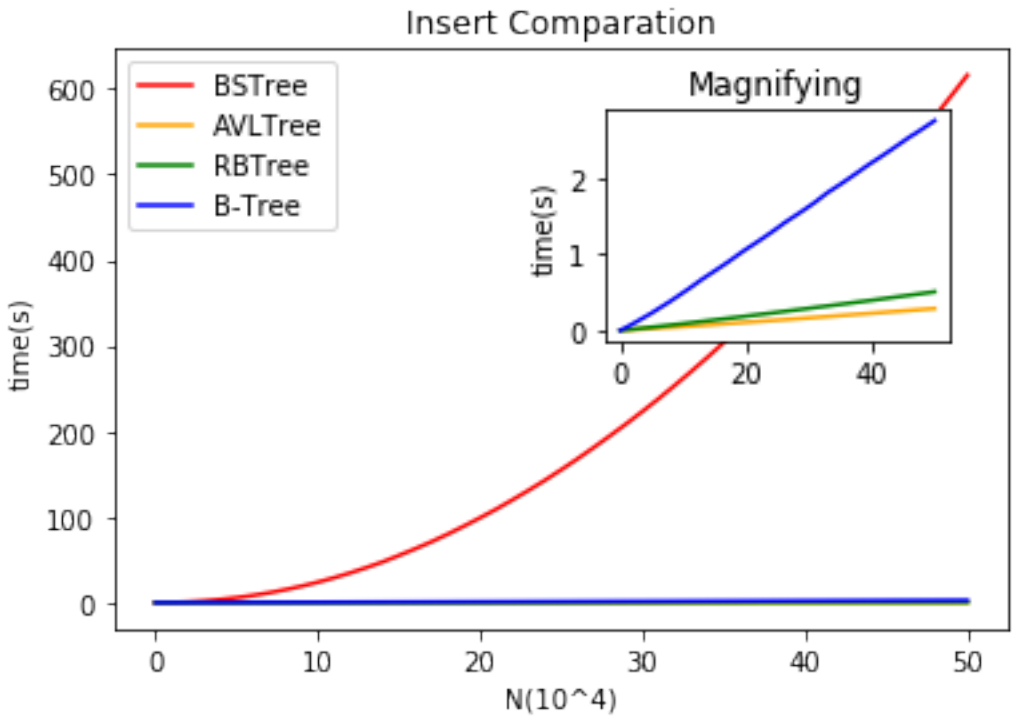
[5]:
```
bs_in, avl_in, rb_in, bt_in = ins[:51], ins[51:102], ins[102:153], ins[153:204]
bs_se, avl_se, rb_se, bt_se = se[:51], se[51:102], se[102:153], se[153:204]
bs_de, avl_de, rb_de, bt_de = de[:51], de[51:102], de[102:153], de[153:204]
```

[6]:
```
n = range(0, 51, 1)
```

[7]:
```
plt.title('Insert Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_in, c='red', label="bs")
l2, = plt.plot(n, avl_in, c='orange', label="avl")
l3, = plt.plot(n, rb_in, c='green', label="rb")
l4, = plt.plot(n, bt_in, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')

plt.axes([0.55, 0.5, 0.3, 0.3], zorder=2)
plt.title('Magnifying')
plt.ylabel("time(s)")
plt.plot(n, avl_in, c='orange', label="avl")
plt.plot(n, rb_in, c='green', label="rb")
```

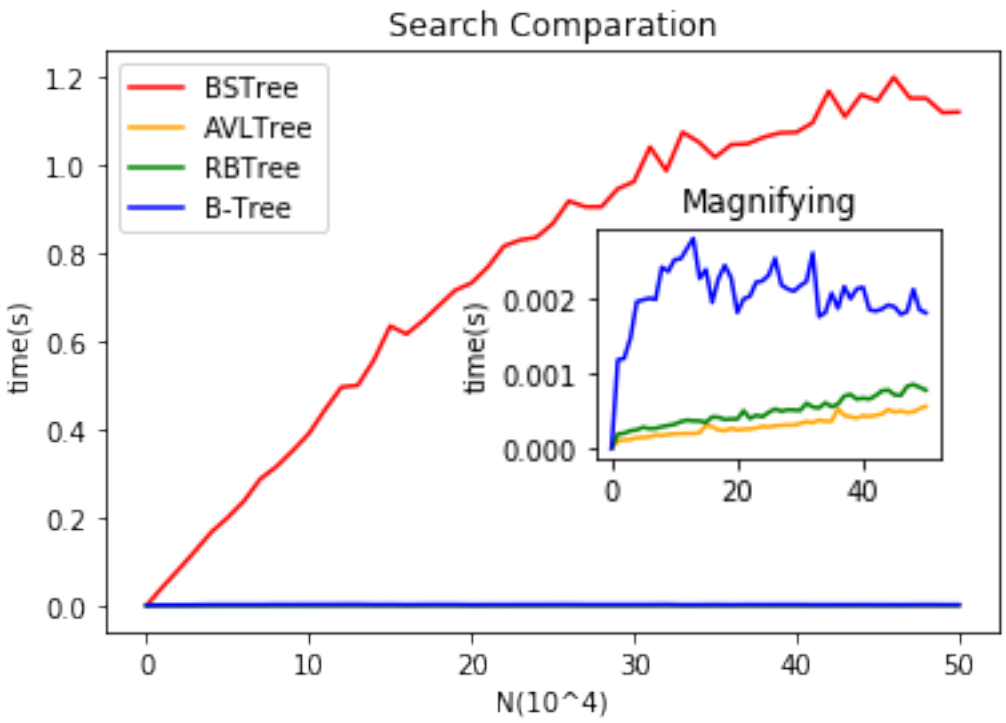
```
plt.plot(n, bt_in, c='blue', label="bt")
```

[7]: [<matplotlib.lines.Line2D at 0x7fbd0340ba90>]



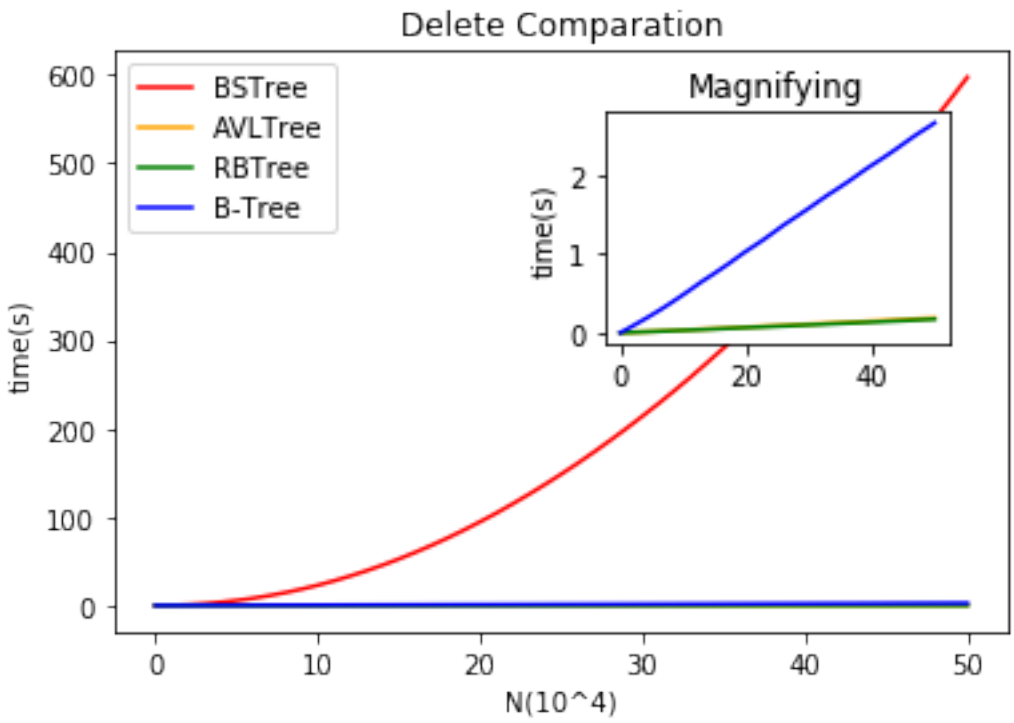
[8]:
```
plt.title('Search Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_se, c='red', label="bs")
l2, = plt.plot(n, avl_se, c='orange', label="avl")
l3, = plt.plot(n, rb_se, c='green', label="rb")
l4, = plt.plot(n, bt_se, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
 ↪'B-Tree'], loc='upper left')

plt.axes([0.55, 0.35, 0.3, 0.3], zorder=2)
plt.title('Magnifying')
plt.ylabel("time(s)")
plt.plot(n, avl_se, c='orange', label="avl")
plt.plot(n, rb_se, c='green', label="rb")
plt.plot(n, bt_se, c='blue', label="bt")
```

[8]: [<matplotlib.lines.Line2D at 0x7fbd036eb6d0>]



[9]:
```
plt.title('Delete Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_de, c='red', label="bs")
l2, = plt.plot(n, avl_de, c='orange', label="avl")
l3, = plt.plot(n, rb_de, c='green', label="rb")
l4, = plt.plot(n, bt_de, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')

plt.axes([0.55, 0.5, 0.3, 0.3], zorder=2)
plt.title('Magnifying')
plt.ylabel("time(s)")
plt.plot(n, avl_de, c='orange', label="avl")
plt.plot(n, rb_de, c='green', label="rb")
plt.plot(n, bt_de, c='blue', label="bt")
```

[9]: [<matplotlib.lines.Line2D at 0x7fbd03843c50>]


Delete Comparation
BSTree
AVLTree
RBTree
B-Tree
Magnifying
time(s)
N(10^4)
0
100
200
300
400
500
600
10
20
30
40
50
1
2

3

December 6, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
data1 = pd.read_csv("3-1.csv", header=None, names=["insert","search","delete"])
data2 = pd.read_csv("3-2.csv", header=None, names=["insert","search","delete"])
data3 = pd.read_csv("3-3.csv", header=None, names=["insert","search","delete"])
```

[3]:
```
in1, se1, de1 = data1["insert"], data1["search"], data1["delete"]
in2, se2, de2 = data2["insert"], data2["search"], data2["delete"]
in3, se3, de3 = data3["insert"], data3["search"], data3["delete"]
```

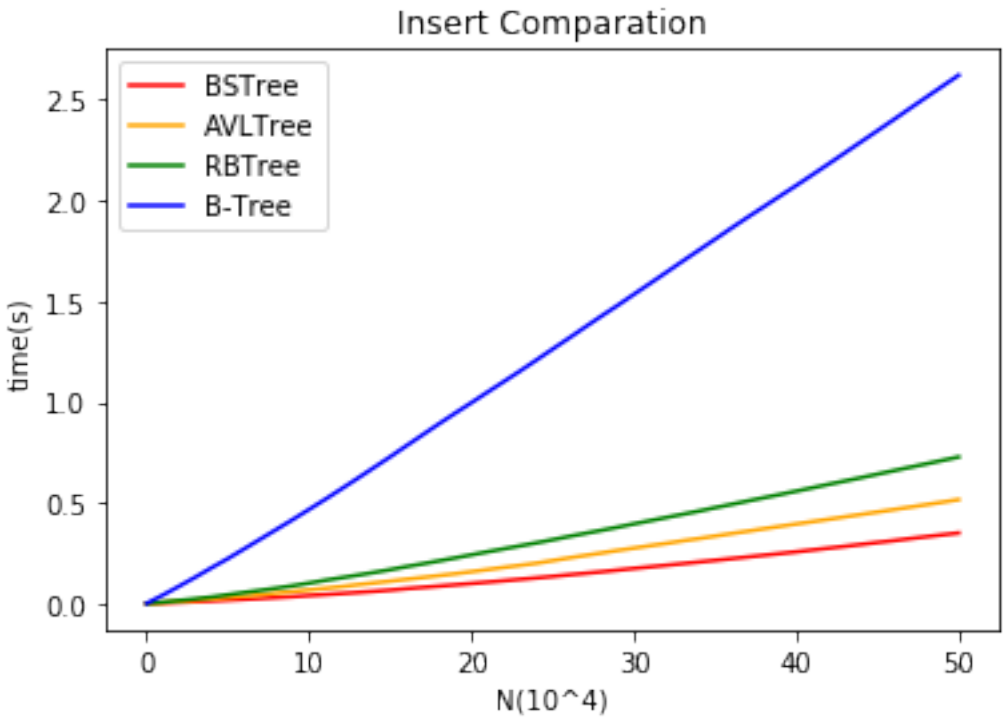
[4]:
```
ins = [(in1[i]+in2[i]+in3[i])/3 for i in range(len(in1))]
se = [(se1[i]+se2[i]+se3[i])/3 for i in range(len(se1))]
de = [(de1[i]+de2[i]+de3[i])/3 for i in range(len(de1))]
```

[5]:
```
bs_in, avl_in, rb_in, bt_in = ins[:51], ins[51:102], ins[102:153], ins[153:204]
bs_se, avl_se, rb_se, bt_se = se[:51], se[51:102], se[102:153], se[153:204]
bs_de, avl_de, rb_de, bt_de = de[:51], de[51:102], de[102:153], de[153:204]
```

[6]:
```
n = range(0, 51, 1)
```

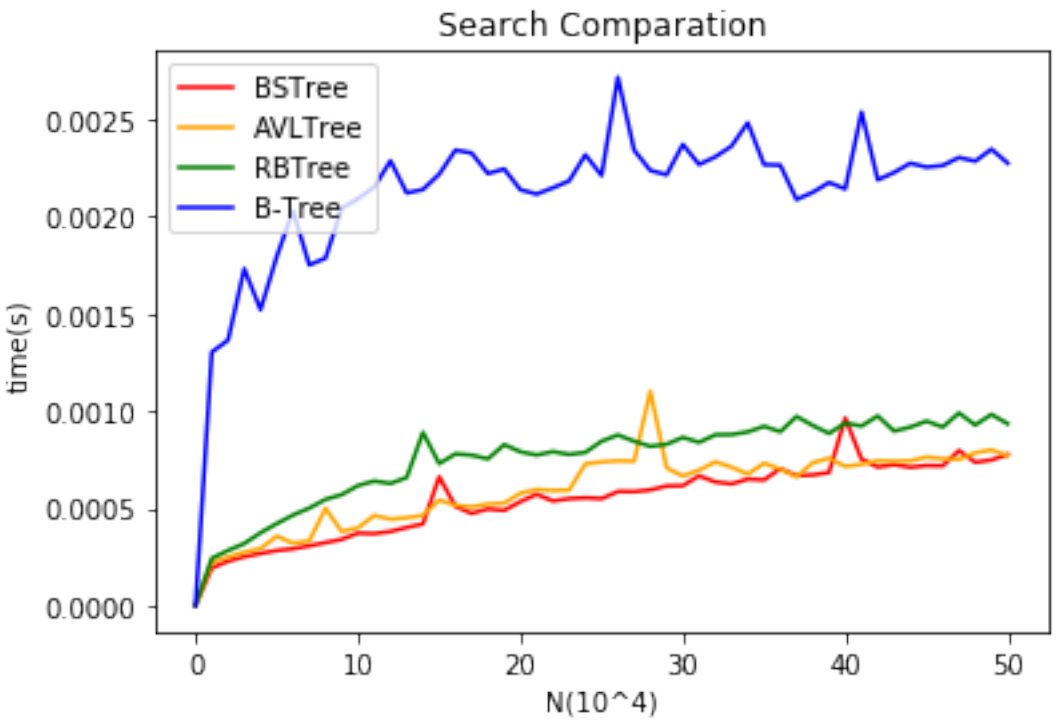
[7]:
```
plt.title('Insert Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_in, c='red', label="bs")
l2, = plt.plot(n, avl_in, c='orange', label="avl")
l3, = plt.plot(n, rb_in, c='green', label="rb")
l4, = plt.plot(n, bt_in, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')
```

[7]:
```
<matplotlib.legend.Legend at 0x7fc961cb0850>
```



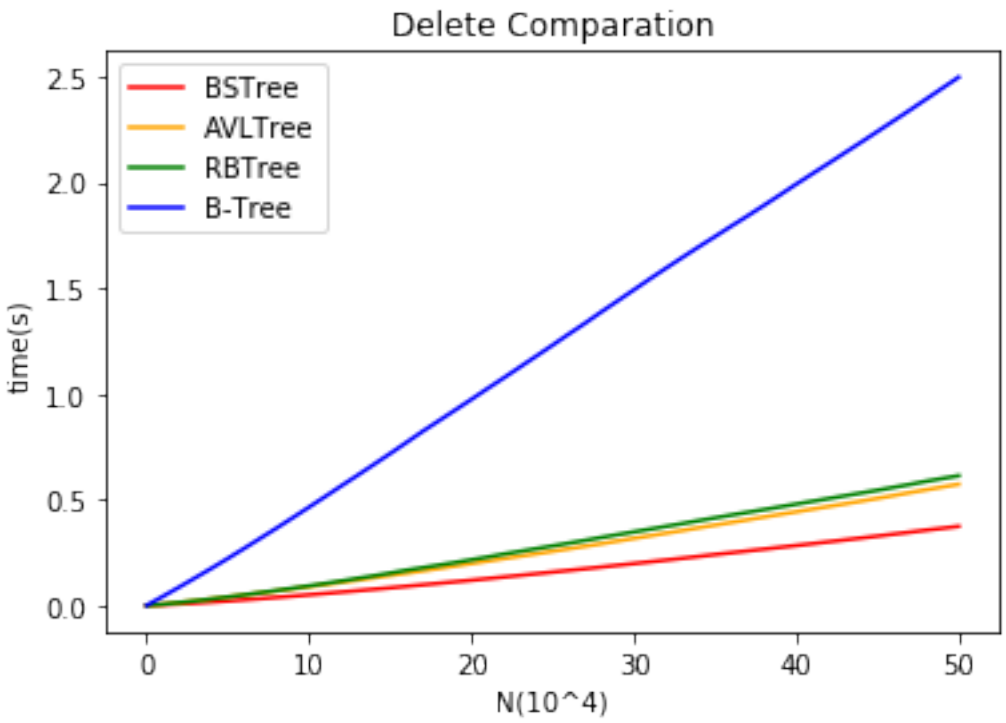
[8]:
```
plt.title('Search Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_se, c='red', label="bs")
l2, = plt.plot(n, avl_se, c='orange', label="avl")
l3, = plt.plot(n, rb_se, c='green', label="rb")
l4, = plt.plot(n, bt_se, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')
```

[8]:
```
<matplotlib.legend.Legend at 0x7fc961d52e50>
```



[9]:
```
plt.title('Delete Comparation')
plt.xlabel("N(10^4)")
plt.ylabel("time(s)")
l1, = plt.plot(n, bs_de, c='red', label="bs")
l2, = plt.plot(n, avl_de, c='orange', label="avl")
l3, = plt.plot(n, rb_de, c='green', label="rb")
l4, = plt.plot(n, bt_de, c='blue', label="bt")
plt.legend(handles=[l1, l2, l3, l4], labels=['BSTree', 'AVLTree', 'RBTree',␣
↪'B-Tree'], loc='upper left')
```

[9]: <matplotlib.legend.Legend at 0x7fc961e92490>


Delete Comparation
BSTree
AVLTree
RBTree
B-Tree
time(s)
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50
N(10^4)