

# Дървовидни Структури от данни (1)

Лекция 5.1 по СДА, Софтуерно Инженерство  
Зимен семестър 2019-2020г  
д-р Милен Чечев

# План за следващите 3 лекции

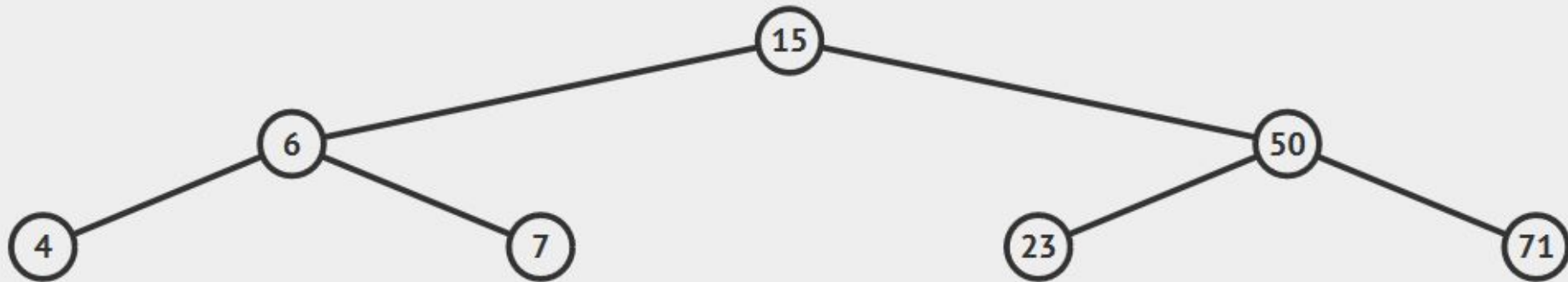
1. Какво е структура от данни дърво и за какво се използва?
2. Основни операции и дефиниции.
3. Двоично дърво за търсене
4. Балансирани дървета. 2-3-4, Red-Black, AVL, Treap
5. Heap. Priority Queue
6. B-tree
7. Задачи и примери

# Дърво

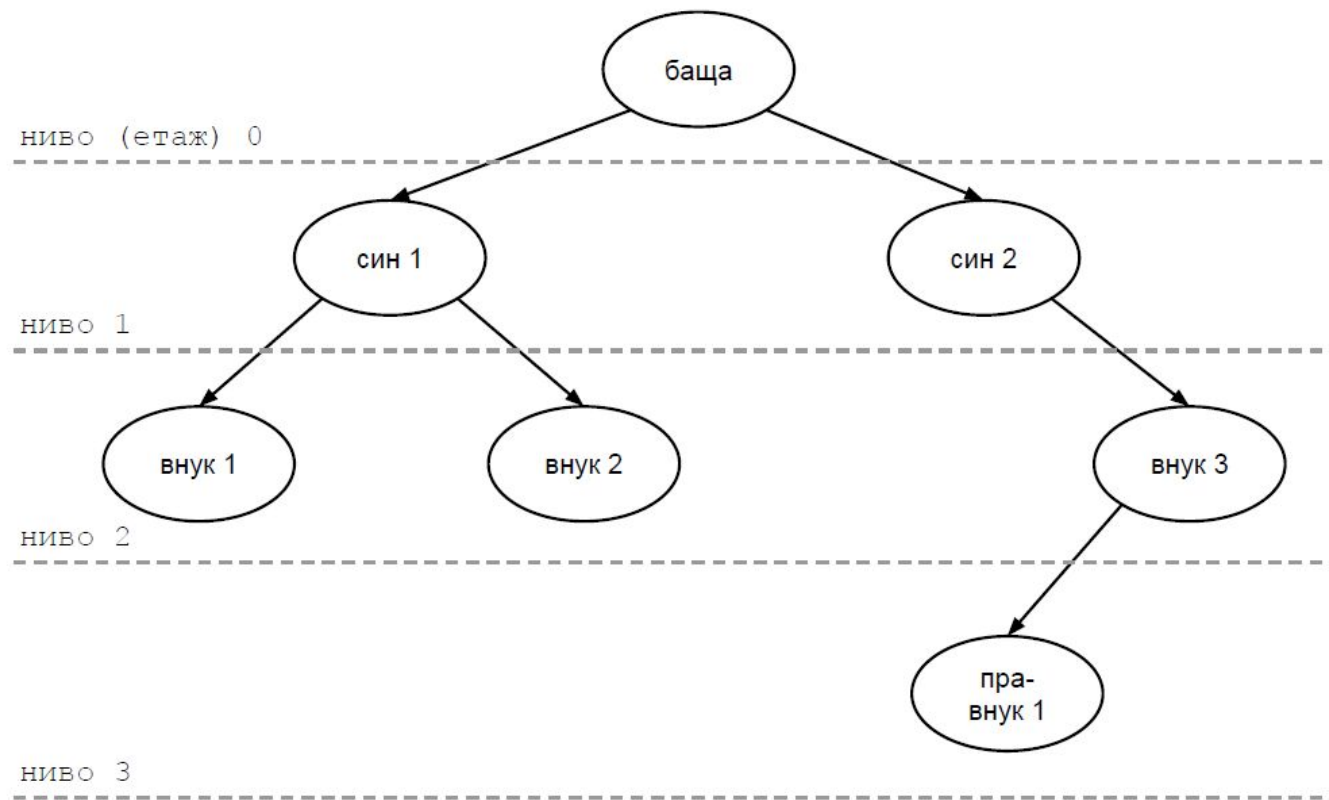
Йерархична структура от данни.

Използва се за организиране на данни в йерархия.

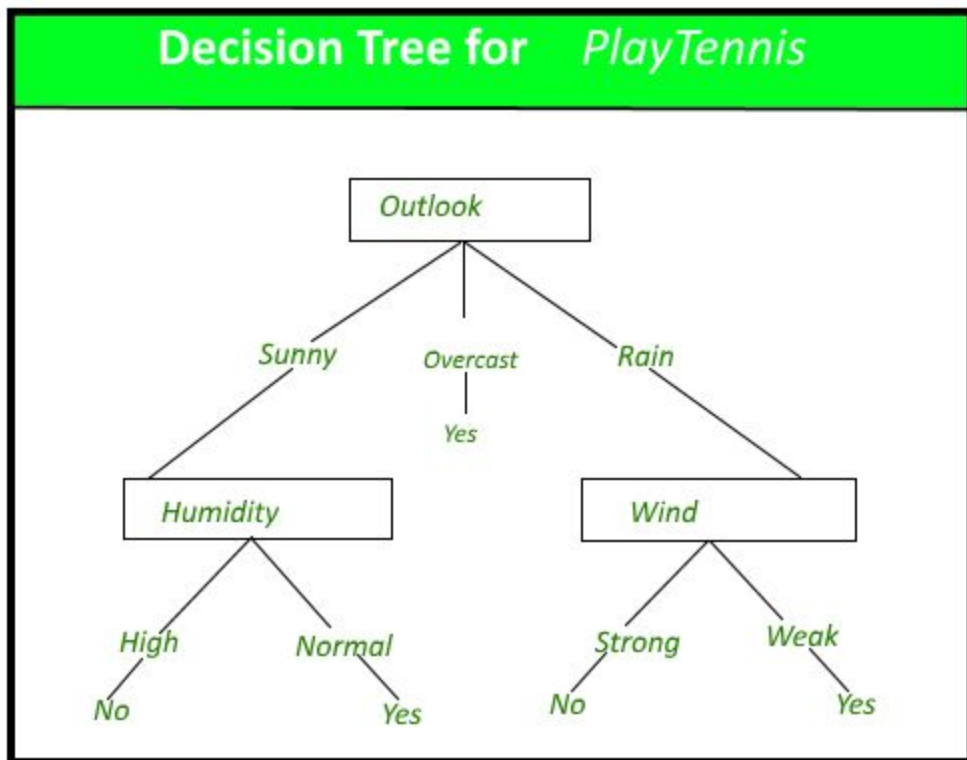
Основна търсена характеристика: бързо добавяне и търсене на елемент



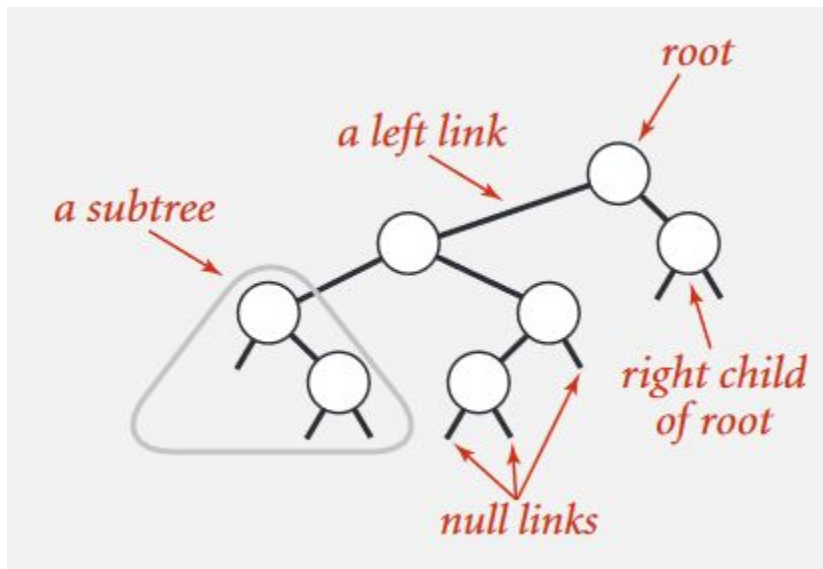
# Пример: фамилно дърво



# Пример: класификационно дърво



# Терминология



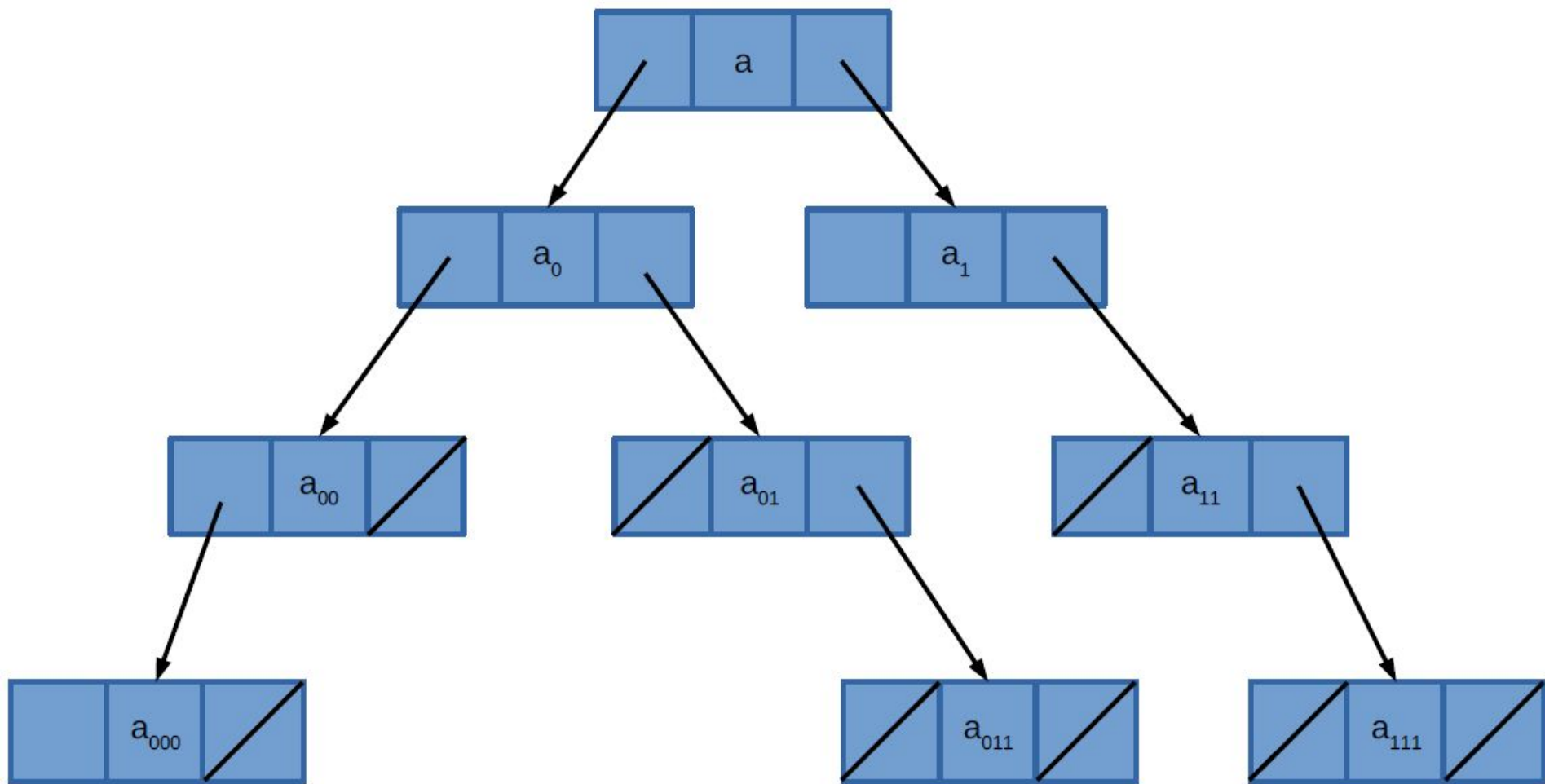
# Двоично наредено дърво(Binary Search Tree)

Двоично дърво: дърво в което всеки възел има максимум два наследника(ляв и десен)

Индуктивна дефиниция на двоично дърво:

- Празно дърво е двоично дърво
- Възел, който има два наследника, които са двоични дървета е двоично дърво.

Двоично наредено дърво: двоично дърво, в което всеки възел и наследниците му са в симетрична подредба( например: Всички леви наследници са по-малки от бащата, всички десни по-големи)





# Основни операции

`search(X)` - търси елемент в дървото

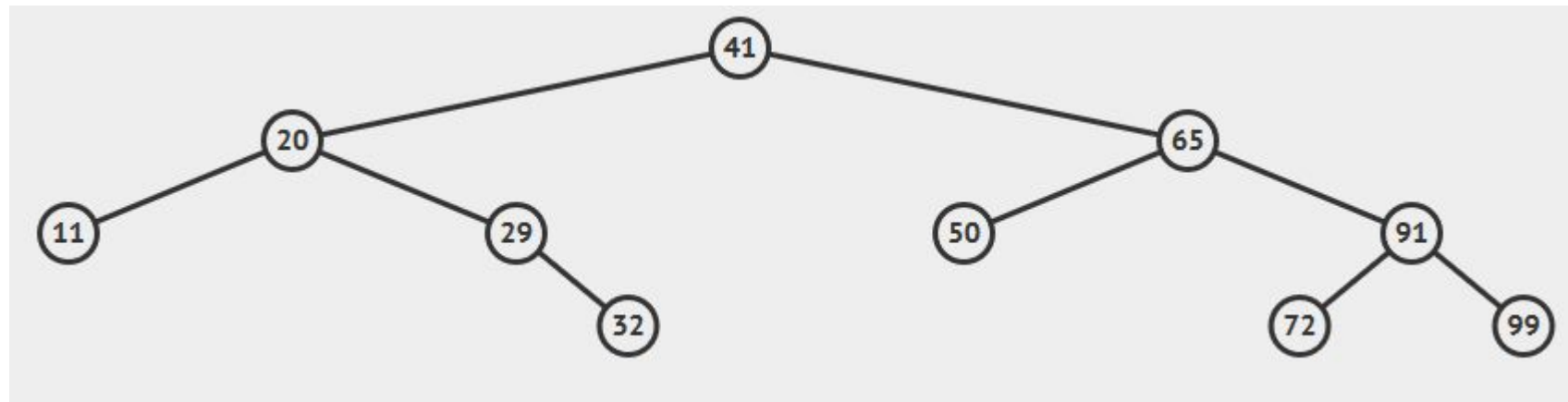
`insert(X)` - добавя елемент в дървото

`remove(X)` - изтрива елемент от дървото

`print()` - принтира дървото в специфичен ред

# Добавяне на елемент

<https://visualgo.net/en/bst>



# Имплементация.

```
struct node{
    int key;
    struct node *left, *right;
};
node* insert(node* root, int key){
    if (root == NULL){
        return new Node(key);
    }
    if (key < root->key)
        root->left = insert(root->left, key);
    else if (key > root->key)
        root->right = insert(root->right, key);
    return root;
}
```

# Принтиране на дърво

Различни възможности за обхождане:

- префиксни
  - КЛД(корен, ляво дърво, дясно дърво)
  - КДЛ
- инфиксни
  - ЛКД
  - ДКЛ
- постфиксни
  - ЛДК
  - ДЛК

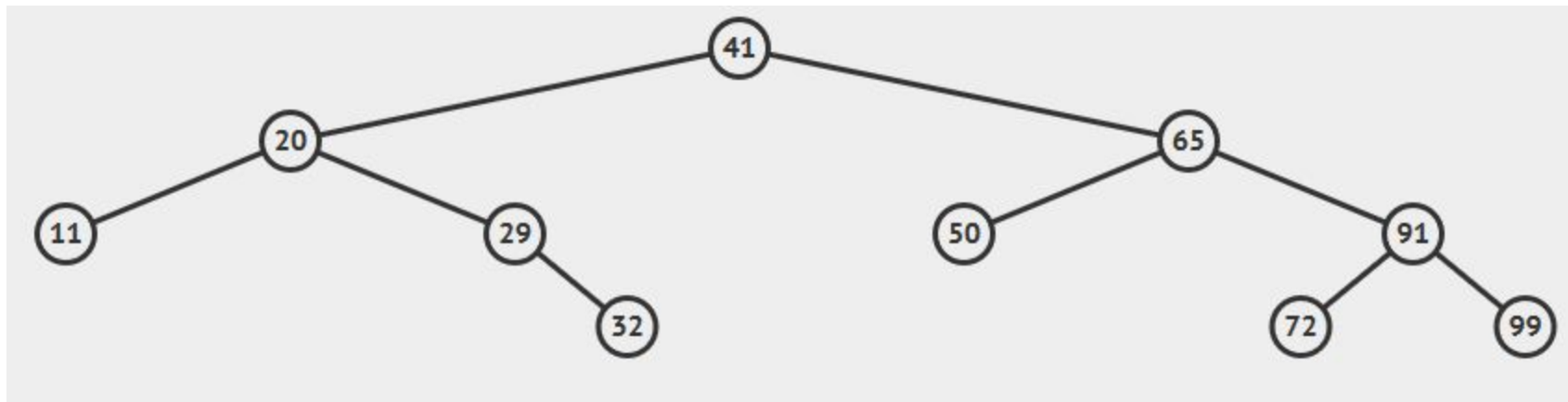
# Inorder принтиране (ЛКД)

```
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d \n", root->key);
        inorder(root->right);
    }
}
```

# Изтриване на елемент

Алгоритъм: <https://visualgo.net/en/bst>

- Намиране на възела за изтриване
- Ако е листо - изтриваме
- Ако има само един наследник, поставяме наследника на негово място
- Ако има двама наследника, намираме следващото по-големина число в дървото, разменяме го с елемента за изтриване и премахваме елемента.



```
node* deleteNode(struct node* root, int key){
    if (root == NULL) return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else{
        if (root->left == NULL){
            node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL){
            node *temp = root->left;
            free(root);
            return temp;
        }
        //... continue on next page
    }
}
```

```
// ... continue from previous page

// now we are at the case of node with two children
// Get the inorder successor (smallest in the right subtree)
struct node* temp = minValueNode(root->right);

// Copy the inorder successor's content to this node
root->key = temp->key;

// Delete the inorder successor
root->right = deleteNode(root->right, temp->key);
}
return root;
}
```



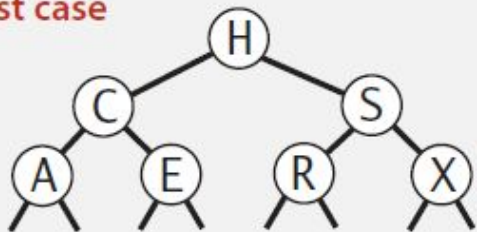
```
node * minValueNode( node* node)
{
    node* current = node;

    /* loop down to find the leftmost leaf */
    while (current->left != NULL)
        current = current->left;

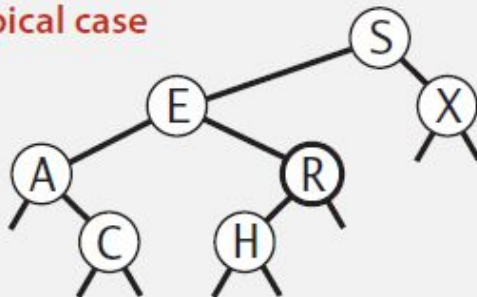
    return current;
}
```

# Форма на дървото

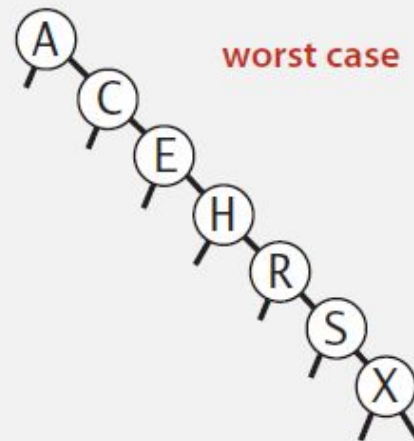
best case



typical case



worst case



# Сложность

implementation	guarantee		average case	
	search	insert	search hit	insert
sequential search (unordered list)	$n$	$n$	$n$	$n$
binary search (ordered array)	$\log n$	$n$	$\log n$	$n$
BST	$n$	$n$	$\log n$	$\log n$