Struktur Data

Binary Search Tree (BST)

Universitas Muslim Indonesia

Oleh
Lutfi Budi Ilmawan

Introduction

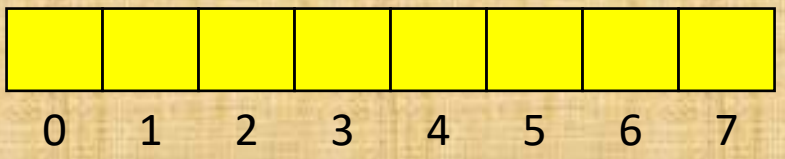
- BST merupakan jenis khusus dari *binary tree* yang mengorganisasikan data secara efisien untuk proses *search*/pencarian, *update*/pengubahan, dan *remove*/penghapusan sehingga prosesnya bisa lebih cepat.

Introduction

- Struktur data yang dapat digunakan untuk menyimpan sekumpulan data yang isinya dapat dimodifikasi secara cepat.
  - Search(x)
  - Insert(x)
  - Remove(x)
- Kita dapat menggunakan ***array*** atau ***linked list***.
- Berapa running time yang dibutuhkan?

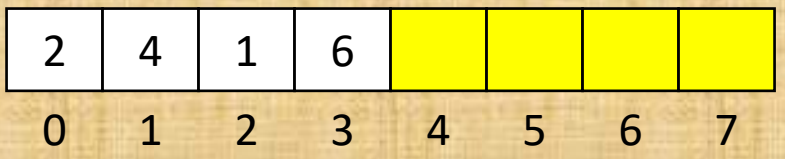
Kompleksitas waktu

Array



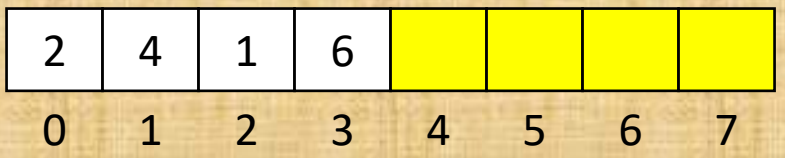
Kompleksitas waktu

Array



Kompleksitas waktu

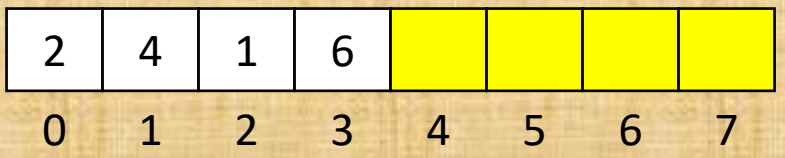
Array



Search(x) O(n)

Kompleksitas waktu

Array



Search(x) O(n)

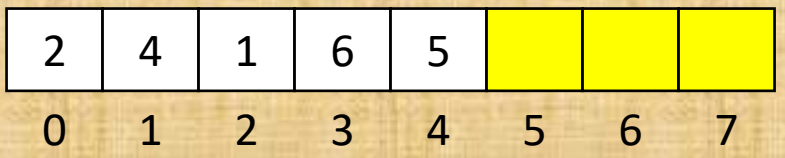
Insert(x)

Insert(5)

Kompleksitas waktu

Array

Search(x)	O(n)
Insert(x)	O(1)

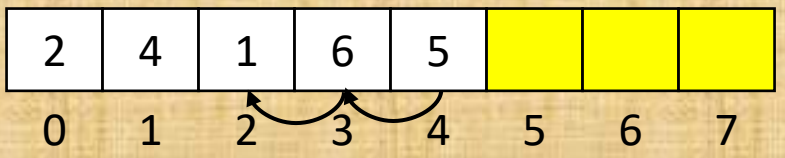


Insert(5)

Kompleksitas waktu

Array

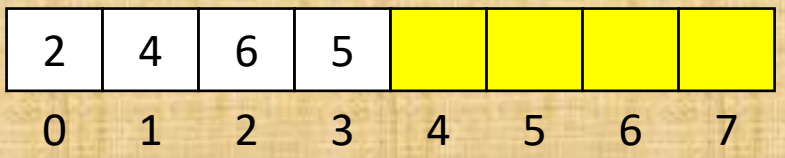
Search(x)	O(n)
Insert(x)	O(1)
Remove(x)	



Insert(5)

Remove(1)

Kompleksitas waktu



Array

Search(x)	O(n)
Insert(x)	O(1)
Remove(x)	O(n)

Insert(5)

Remove(1)

Introduction

- Proses insert cukup cepat, namun untuk proses pencarian waktu yang dibutuhkan bisa cukup lama, sebab pengecekan data dilakukan satu persatu.
- Misalkan waktu yang dibutuhkan untuk 1 pengecekan = 10^{-6} sec.
- Jika dilakukan pencarian pada array yang jumlah datanya 10^{8} (100 juta) maka waktu yang dibutuhkan = 100 sec.
- Untuk mempercepat proses kita dapat melakukan pencarian dengan binary search dengan syarat nilai dalam array telah terurut. Kompleksitas waktunya = 0(log n).

Introduction

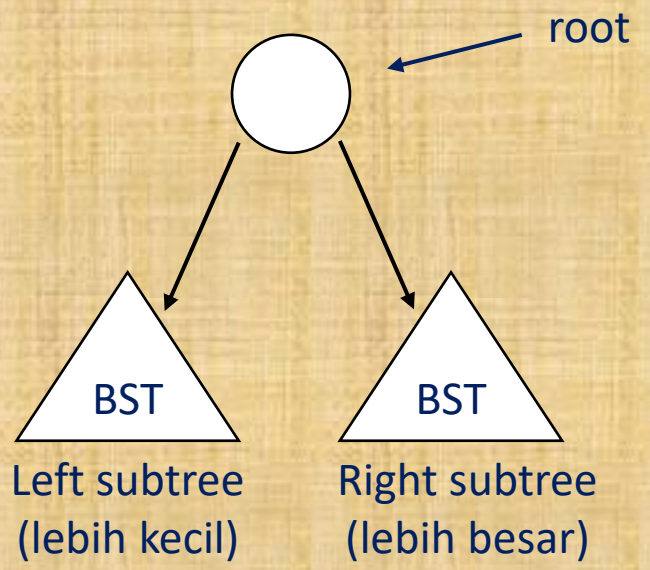
Operasi	Array (unsorted)	Linked List	Array (sorted)
Search(x)	O(n)	O(n)	$O(\log_2 n)$
Insert(x)	O(1)	O(1)	O(n)
Remove(x)	O(n)	O(n)	O(n)

Introduction

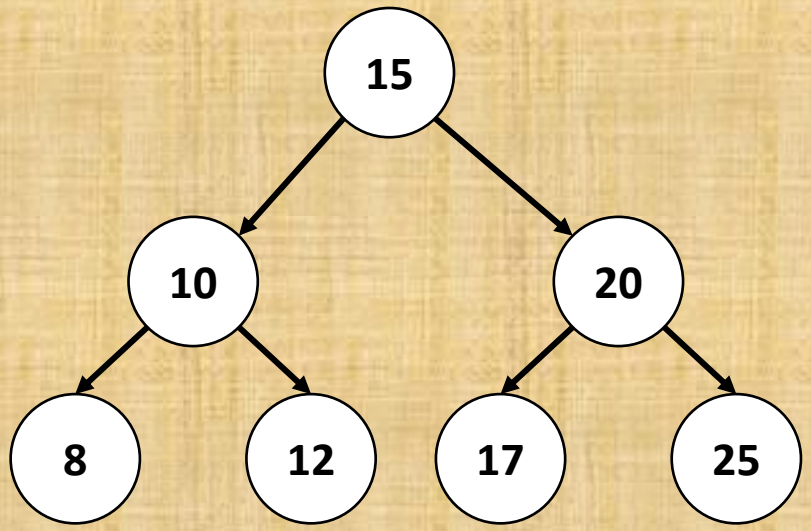
Operasi	Array (unsorted)	Linked List	Array (sorted)	BST (balanced)
Search(x)	O(n)	O(n)	$O(\log_2 n)$	$O(\log_2 n)$
Insert(x)	O(1)	O(1)	O(n)	$O(\log_2 n)$
Remove(x)	O(n)	O(n)	O(n)	$O(\log_2 n)$

Binary Search Tree (BST)

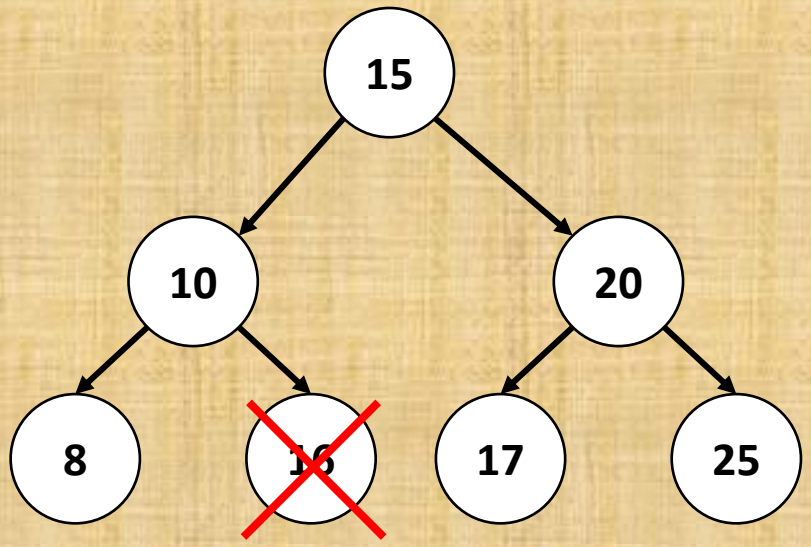
- BST adalah suatu *tree* di mana untuk setiap *node*-nya, nilai dari *left subtree*-nya(cabang sebelah kiri) lebih kecil atau sama dengan, dan nilai dari *right subtree*-nya (cabang sebelah kanan) lebih besar.



Contoh BST

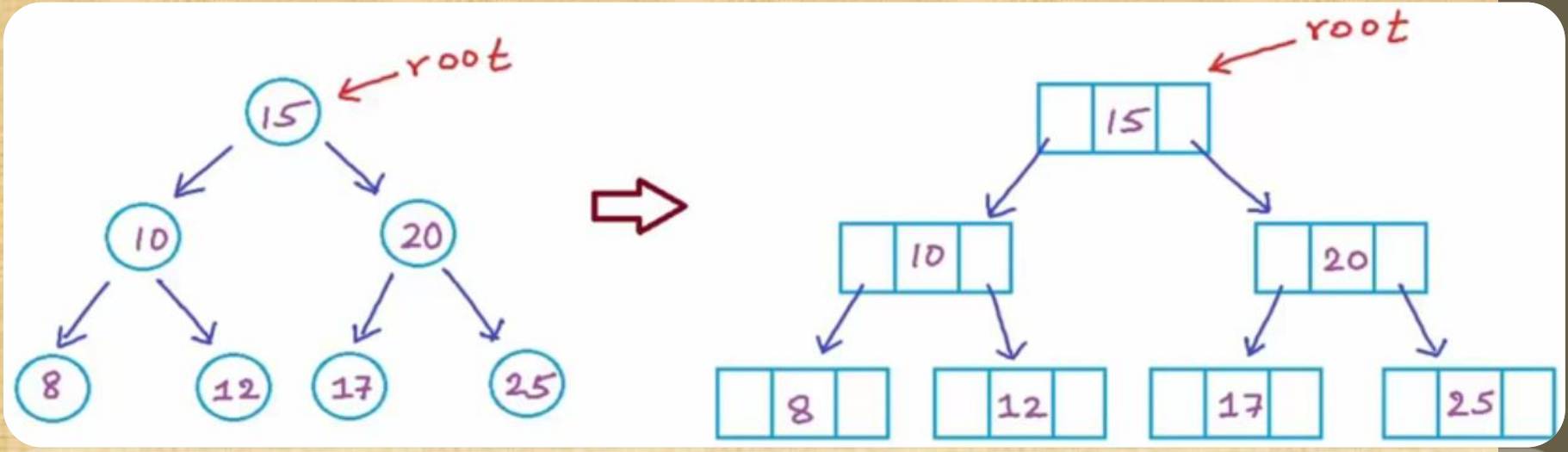


Contoh BST



Implementasi BST

- BST dapat diimplementasikan pada c++, strukturnya sama dengan linked list, tiap node terhubung melalui pointer.




```
struct BstNode {
    int data;
    BstNode* left;
    BstNode* right;
};
```

Deklarasi

```
struct BstNode{
    int data;
    BstNode *left;
    BstNode *right;
};
BstNode *root=NULL;
```

Binary Tree Traversal

- Definisi: Proses mengunjungi (***visiting***) setiap node pada sebuah tree tepat satu kali dengan menggunakan urutan tertentu.
- Visiting → membaca atau memproses data dalam sebuah node.

Binary Tree Traversal

- **Breadth-first**
  - Level order
- **Depth-first**
  - Preorder → <root><left><right>
  - Inorder → <left><root><right>
  - Postorder → <left><right><root>

Hapus Node pada BST

- **Kasus I → Jika node tidak memiliki child**
- **Kasus II → Jika node memiliki 1 child**
- **Kasus III → Jika node memiliki 2 child**