



# B-Trees B+ Trees

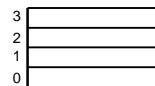
Dr. Thomas Hicks  
Computer Science Department  
Trinity University

1

```
# define M 5
```

B Trees Nodes

1] Contain M - 1 Records

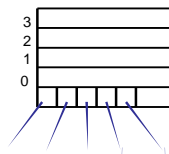


```
# define M 5
```

B Trees Nodes

1] Contain M - 1 Records

2] Contain M Pointers To Other B Tree Nodes



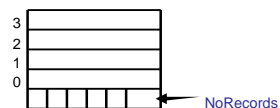
```
# define M 5
```

B Trees Nodes

1] Contain M - 1 Records

2] Contain M Pointers To Other B Tree Nodes

3] Contain NoRecords Counter



```
# define M 5
```

```
template <class InfoType>
```

```
class BTreeNode
```

```
{
```

```
public:
```

```
    BTreeNode(void);
```

```
private:
```

```
    InfoType
```

```
        Info[M-1];
```

```
    NodePtr
```

```
        Ptrs[M];
```

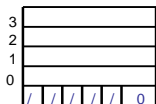
```
    long int
```

```
        NoRecords;
```

```
};
```

Assume that sizeof(InfoType) = 9,996

sizeof(BTreeNode) = **40,008**



```
# define M 11
```

```
template <class InfoType>
```

```
class BTreeNode
```

```
{
```

```
public:
```

```
    BTreeNode(void);
```

```
private:
```

```
    InfoType
```

```
        Info[M-1];
```

```
    NodePtr
```

```
        Ptrs[M];
```

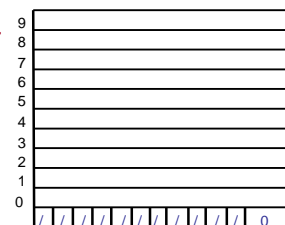
```
    long int
```

```
        NoRecords;
```

```
};
```

Assume that sizeof(InfoType) = 9,996

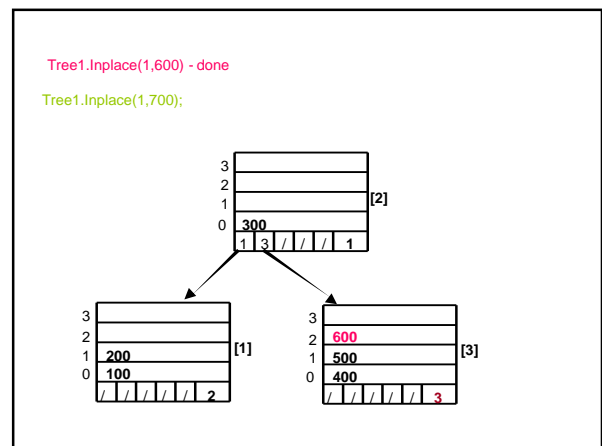
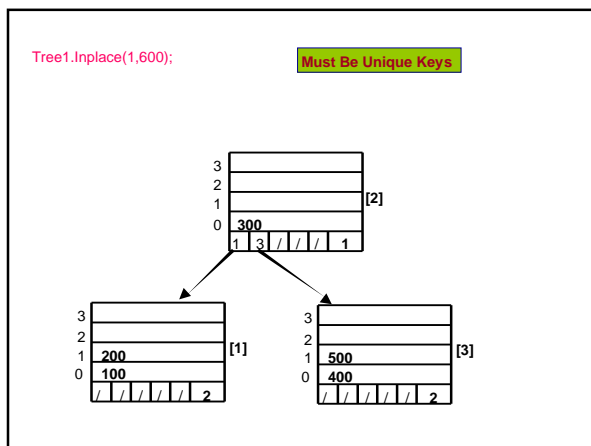
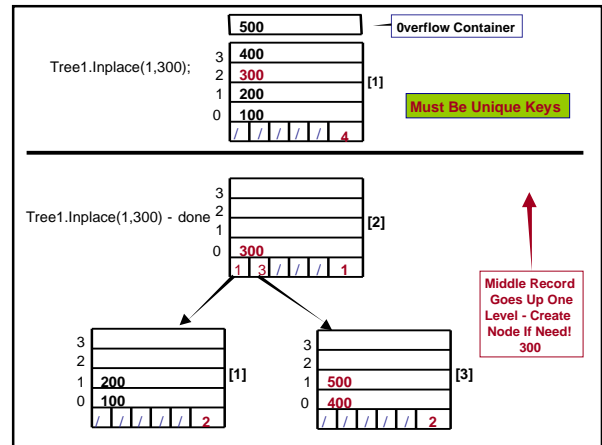
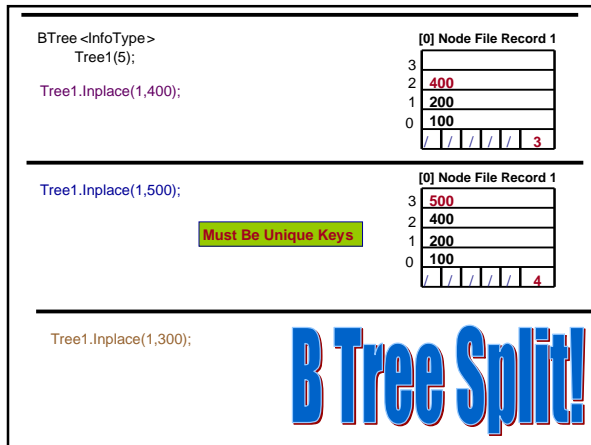
sizeof(BTreeNode) = **100,008**

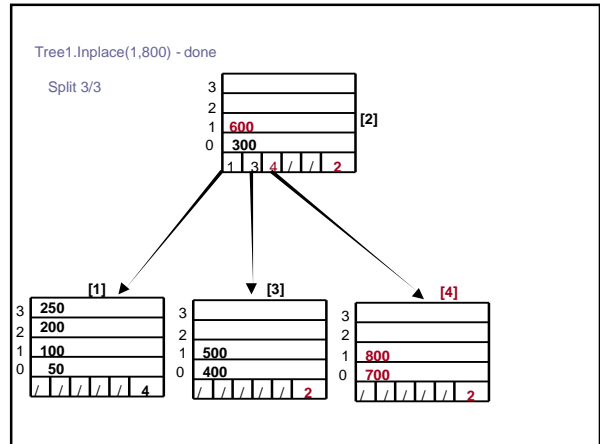
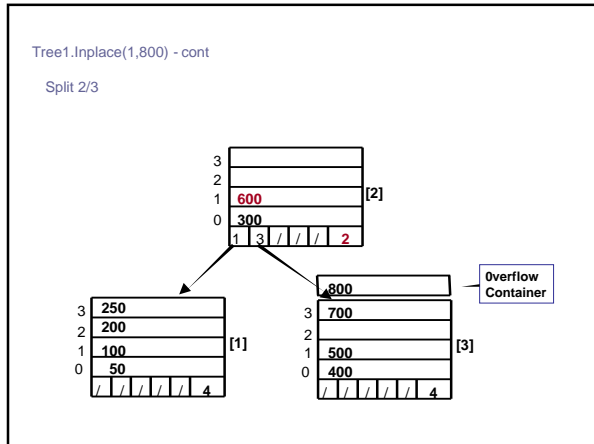
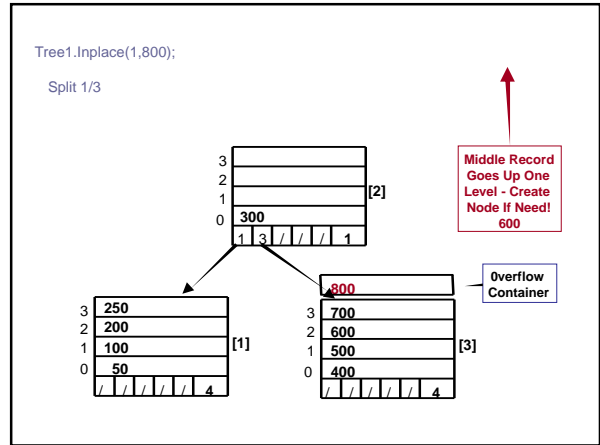
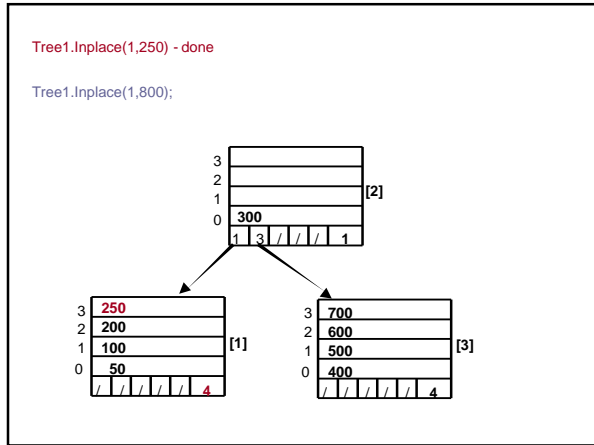
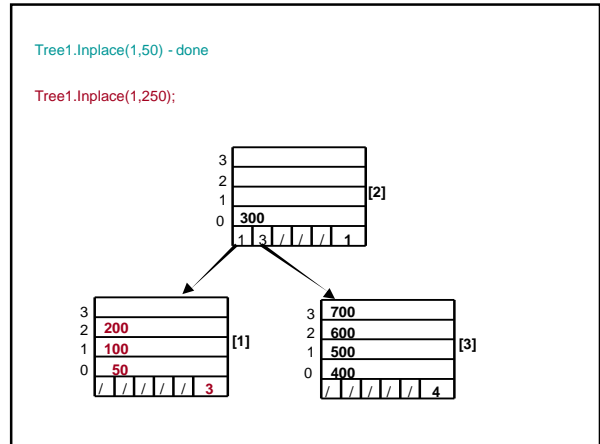
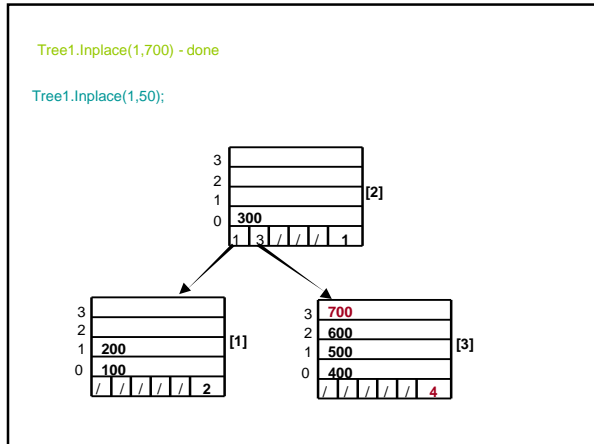


# define M 5

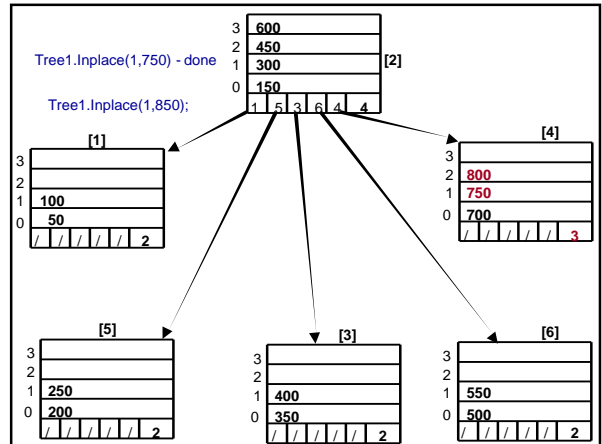
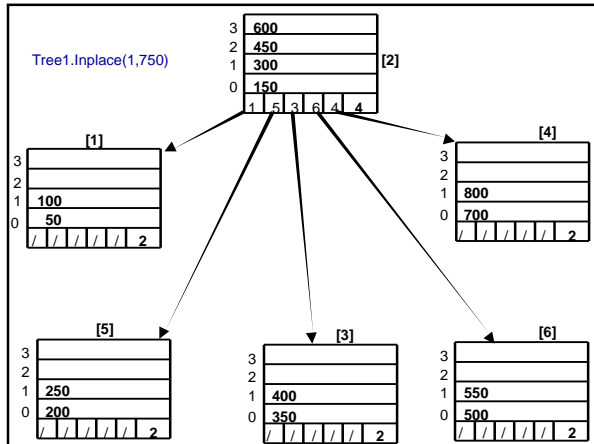
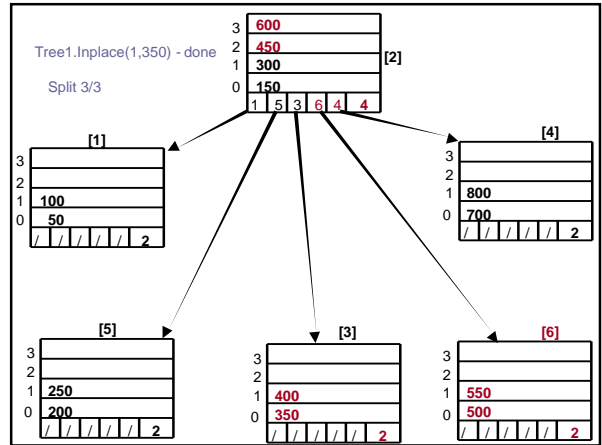
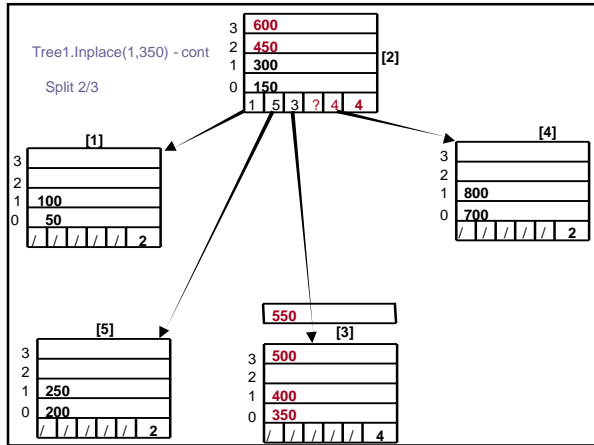
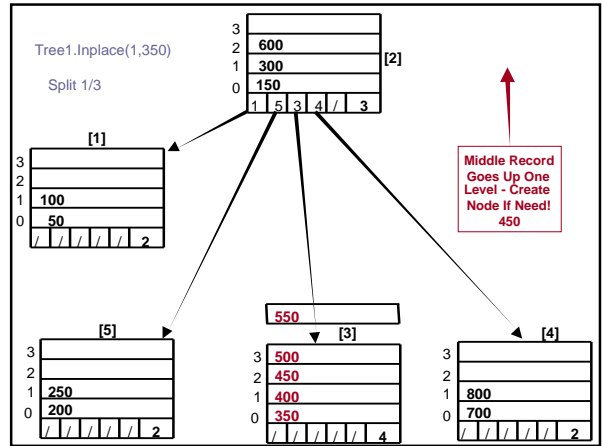
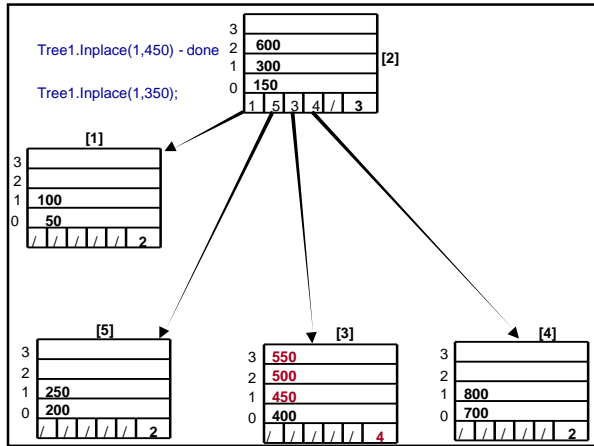
### About B Trees

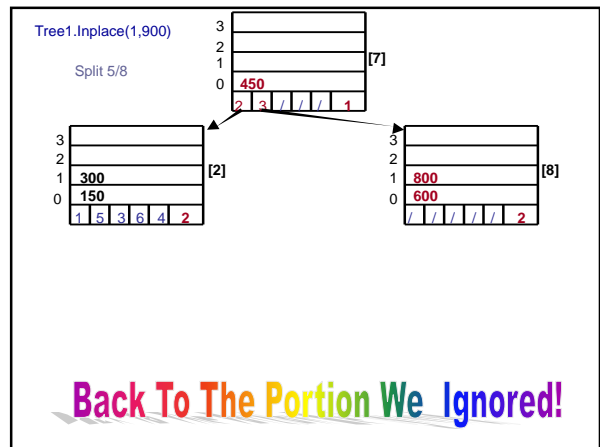
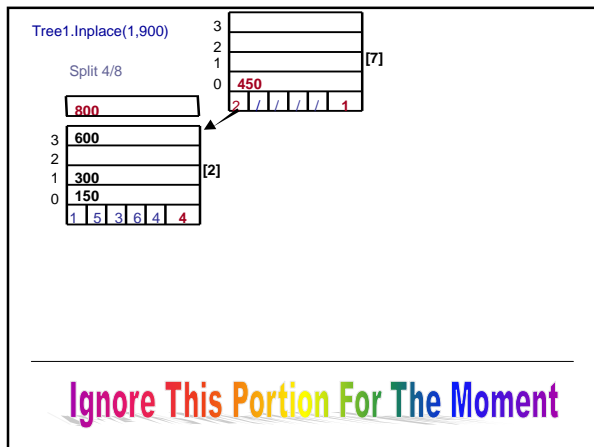
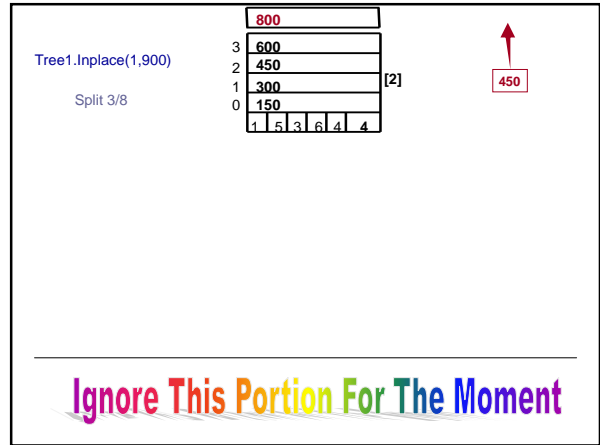
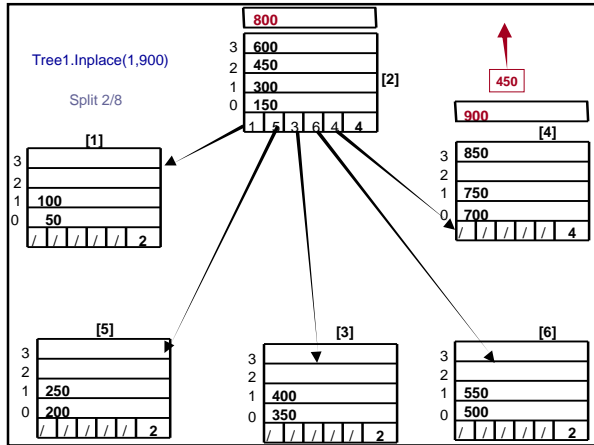
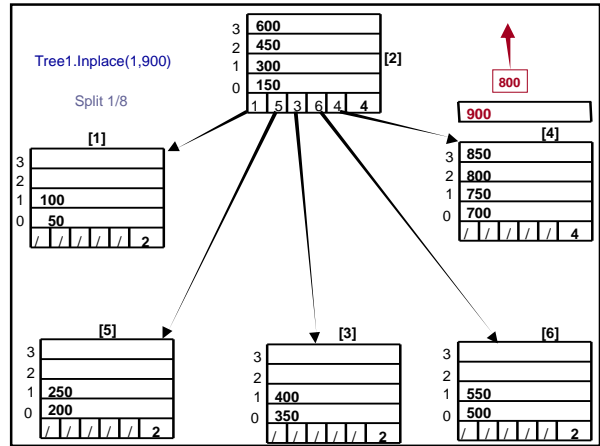
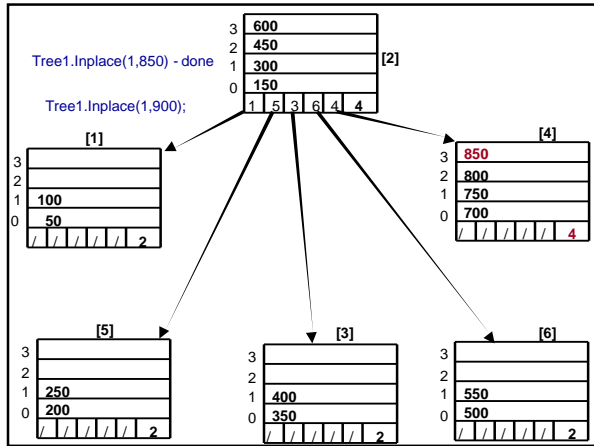
- 1] Nodes Contain M - 1 Records
- 2] Nodes Contain M Pointers To Other B Tree Nodes
- 3] Nodes Contain NoRecords Counter
- 4] Every Node, Except The Root, Shall Always Be At Least Half Full

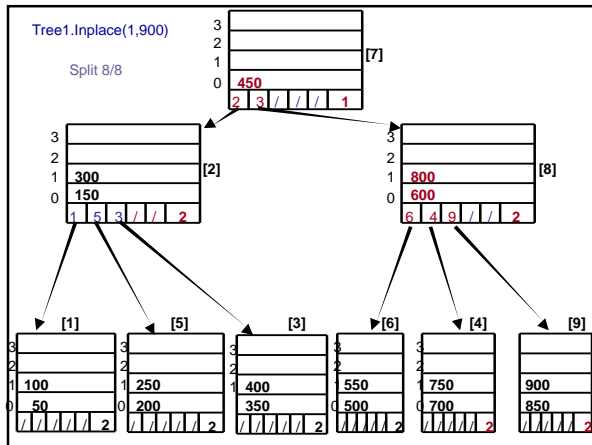
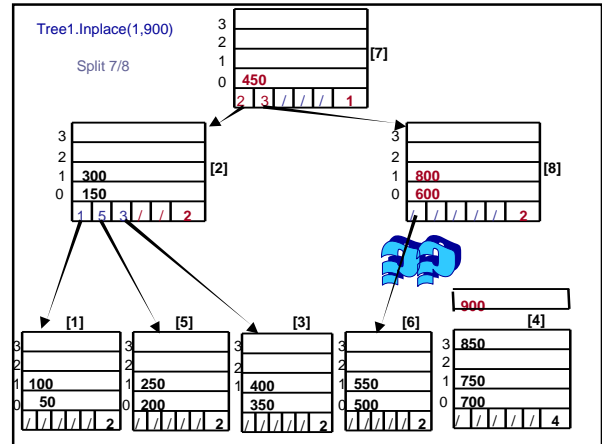
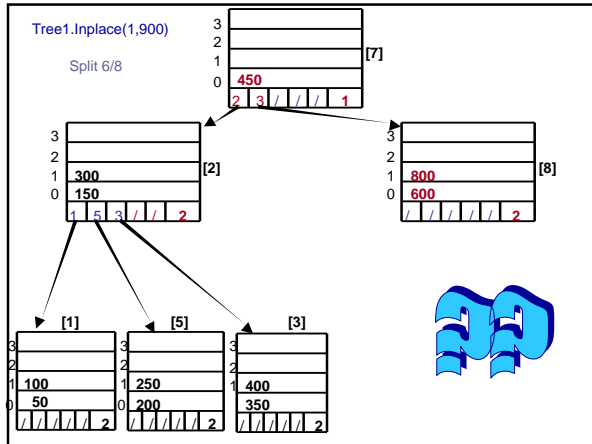












# define M 5

B+Trees - also called M-Way Trees - A balanced search tree in which every *node* has between M-1 and 2M-1 *children*, where M is an arbitrary constant.

This is a good structure if much of the tree is in slow memory (disk), since the *height*, and hence the number of accesses, can be kept small, say one or two, by picking a large t.

### B-Tree

```

B-Tree-Search(x, k)
i <- 1
while i <= n[x] and k > keyi[x]
  do i <- i + 1
if i <= n[x] and k = keyi[x] then
  return (x, i)
if leaf[x] then
  return NIL
else
  Disk-Read(ci[x])
  return B-Tree-Search(ci[x], k)

```

<http://www.public.asu.edu/~peterjn/btree/>

### Build B/B+Tree - BRIEF SKETCH - M = 5

A G F B

Inplace the following

K

D H M

J

E S I R

X

C L N T U

P