```
B-tree-Insert (T, k)
1.  r ← root[T]
2.  if n[r] = 2t-1
3.      then s ← allocate-node()
4.           root[T] ← s
5.           leaf[s] ← False
6.           n[s] ← 0
7.           c1[s] ← r
8            B-tree-Split-Child (s, 1, r)
9.           B-tree-Insert-Nonfull (s, k)
10       else B-tree-Insert-Nonfull (r, k)

B-tree-Insert-NonFull (x, k)
1.  i ← n[x]
2.  if leaf[x]
3.      then while i ≥ 1 and k < key_i[x]
4.                do key_{i+1}[x] ← key_i[x]
5.                   i ← i-1
6.           key_{i+1}[x] ← k
```

The median key $key_t[y]$ is moved up to y's parent.

— To reduce # of disk operations, whenever we travel down th tree searching for the position to insert, we split each ful node we visit along the way.

B-tree-Split-child (x, i, y) // $c_i[x] = y$ is full

1. $z \leftarrow$ allocate-node()
2. $leaf[z] \leftarrow leaf[y]$
3. $n[z] \leftarrow t-1$
4. for $j \leftarrow 1$ to $t-1$ — set up the new n
5. do $key_j[z] \leftarrow key_{j+t}[y]$
6. if not $leaf[y]$
7. then for $j \leftarrow 1$ to t
8. do $c_j[z] \leftarrow c_{j+t}[y]$
9. $n[y] \leftarrow t-1$
10. for $j \leftarrow n[x]+1$ down to $i+1$
11. do $c_{j+1}[x] \leftarrow c_j[x]$ — update the
12. $c_{i+1}[x] \leftarrow z$ — parent node x
13. for $j \leftarrow n[x]$ down to i — after move up
14. do $key_{j+1}[x] \leftarrow key_j[x]$ — $key_t[y]$
15. $key_i[x] \leftarrow key_t[y]$
16. $n[x] \leftarrow n[x]+1$
17. Disk-Write (y)
18. Disk-Write (z)
19. Disk-write (x)

$\Theta(t)$ running time

$O(1)$ disk operations.

Fig 18.5 illustrate this operation.

```
8                B-tree-split-child(S, 1, r)
9.               B-tree-Insert-Nonfull(S, K)
10       else    B-tree-Insert-Nonfull(r, k)
```

```
B-tree-Insert-NonFull(x, K)
1.  i ← n[x]
2.  if leaf[x]
3.      then while i ≥ 1 and K < key_i[x]
4.                do key_{i+1}[x] ← key_i[x]
5.                   i ← i-1
6.           key_{i+1}[x] ← K
7.           n[x] ← n[x] + 1
8.           Disk-Write(x)
9.      else while i ≥ 1 and K < key_i[x]
10.               do i ← i-1
11.          i ← i+1
12.          Disk-Read(c_i[x])
13.          if n[c_i[x]] = 2t-1
14.              then B-tree-Split-Child(x, i,
15.                   if K > key_i[x]
16.                       then i ← i+1
17.          B-tree-Insert-Nonfull(c_i[x], K)
```

Fig 18.7 on page 448 shows various cases of ins