1. Consider the $B^+$-tree $T$ illustrated below. Its B-tree is of order three, which means that each of its nodes must have either two or three children. Its leaf nodes are large enough to hold four entries and, to ensure no worse than 50% storage utilization, we insist that each one holds at least two entries.

For each insert operation in the list (a) through (e), show the $B^+$-tree that results from performing that operation on $T$. (Each operation is to be applied to the tree $T$ illustrated below, not to the tree resulting from applying all the previous operations to $T$.) You need not draw the entire tree each time—just show that portion of it that was changed in carrying out the operation, as well as a little surrounding context.

Assume, in carrying out the insertions, that redistribution is used whenever possible. That is, split an overflowing node only if all of its adjacent/immediate siblings are full. Follow the algorithm presented in class, which corresponds to that described on the relevant web page. Keep in mind that redistribution involves two adjacent siblings and their parent, not three or more siblings and not first (or second, etc.) cousins (i.e., nodes with a common grandparent, great-grandparent, etc.).

(a) insert 24 gorn  
(b) insert 38 gorn  
(c) insert 16 gorn  
(d) insert 58 gorn  
(e) insert 72 gorn
2. Consider the $B^+$-tree $T$ illustrated below. Its $B$-tree is of order three, which means that each of its nodes must have either two or three children. Its leaf nodes are large enough to hold four entries and, to ensure no worse than 50% storage utilization, we insist that each one holds at least two entries.

For each delete operation in the list (a) through (e), show the $B^+$-tree that results from performing that operation on $T$. (Each operation is to be applied to the tree $T$ illustrated below, not to the tree resulting from applying all the previous operations to $T$.) You need not draw the entire tree each time—just show that portion of it that was changed in carrying out the operation, as well as a little surrounding context.

Assume, in carrying out the deletions, that redistribution is used whenever possible. That is, concatenate/merge two nodes only if all adjacent siblings of the underflowing node are on the verge of underflowing. Follow the algorithm presented in class, which corresponds to that described on the relevant web page. Keep in mind that redistribution involves two adjacent siblings and their parent, not three or more siblings and not first (or second, etc.) cousins (i.e., nodes with a common grandparent, great-grandparent, etc.).

(a) delete 3   (c) delete 44   (e) delete 35
(b) delete 52   (d) delete 77

```
| 40 |
+----+----+
| 20 |
+----+----+
| 12 |
+----+----+
| 0 bee| 12 let| 23 jan| 35 arf| 40 sum| 52 car| 60 bob| 69 why| 77 and|
| 15 my | 125 hat| 36 par| 44 bet| 55 but| 61 tar| 71 who| 80 not|
| 19 of | 17 orc| 63 low| 65 hit| 65 hit|
```

2