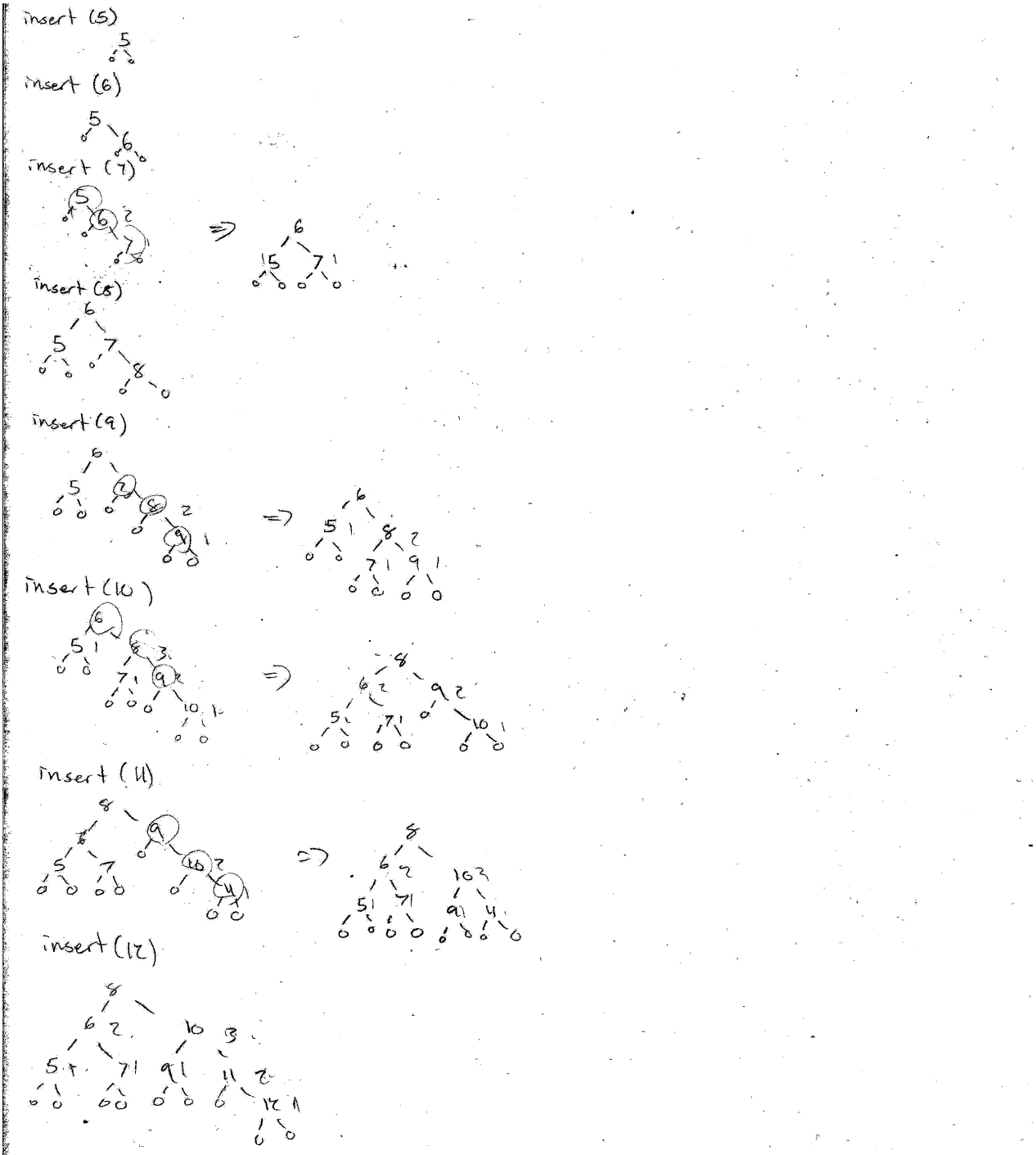
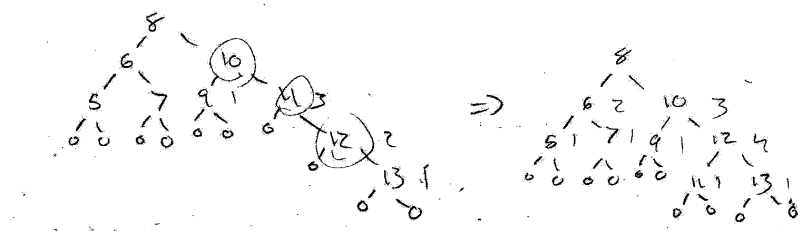
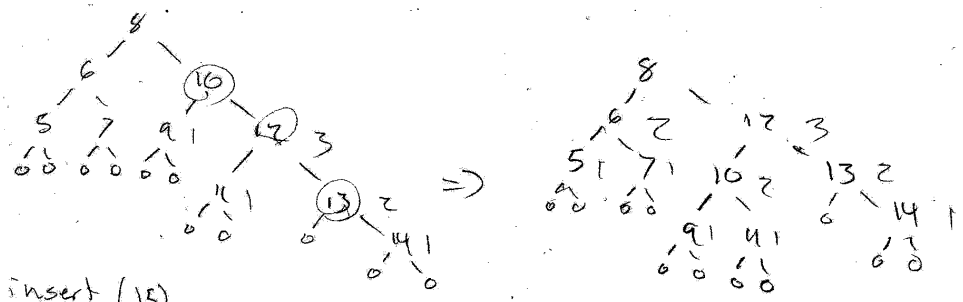
1. Starting with an empty tree, perform the following operations in the AVL tree. Indicate the resulting tree after each step. Also show intermediate steps if any.



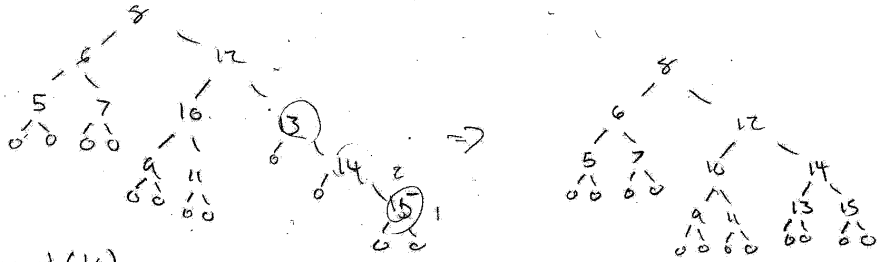
insert(13)



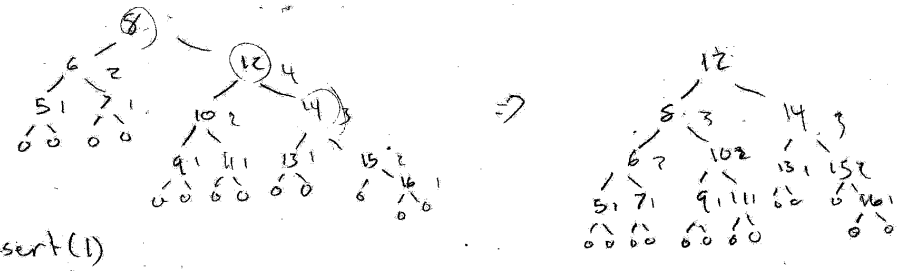
insert(14)



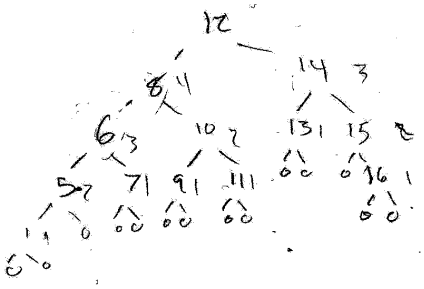
insert(15)



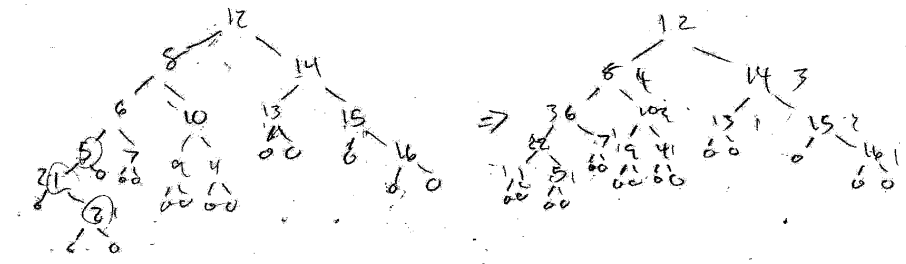
insert(16)



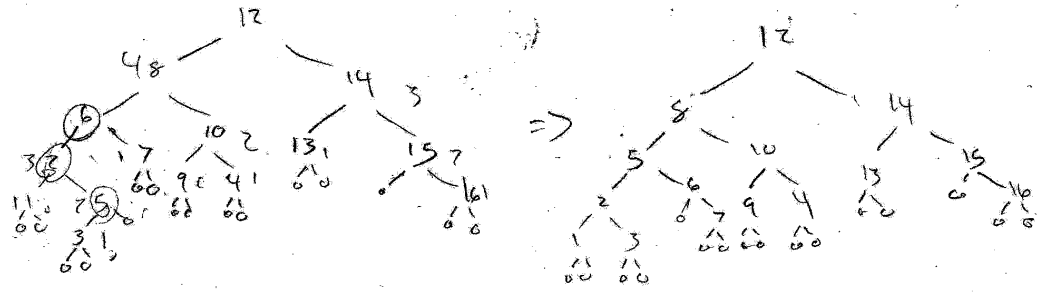
insert(1)



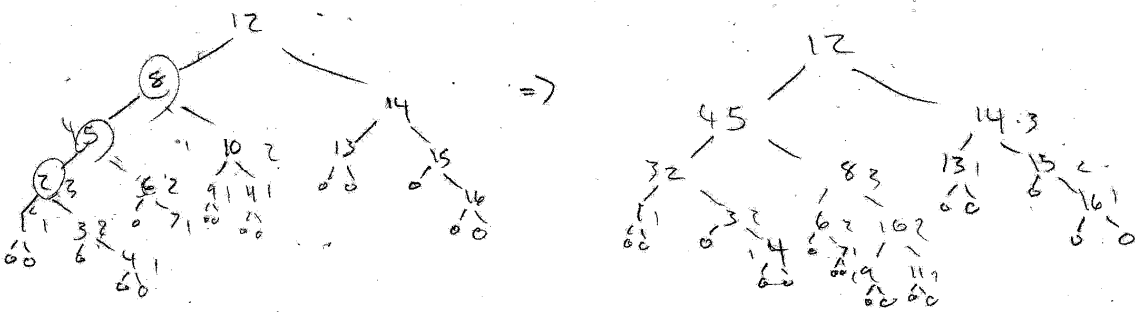
insert(2)



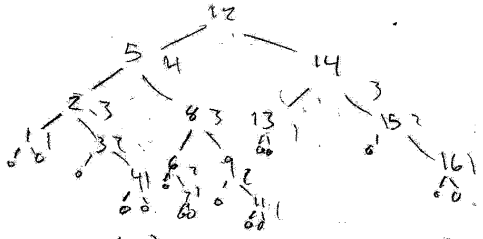
insert(3)



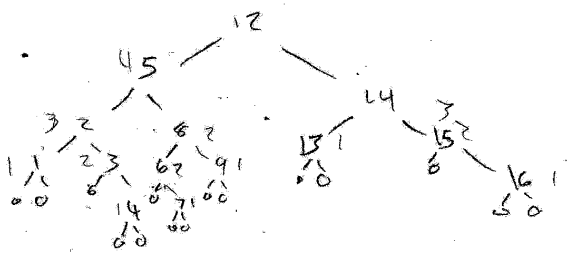
insert (4)



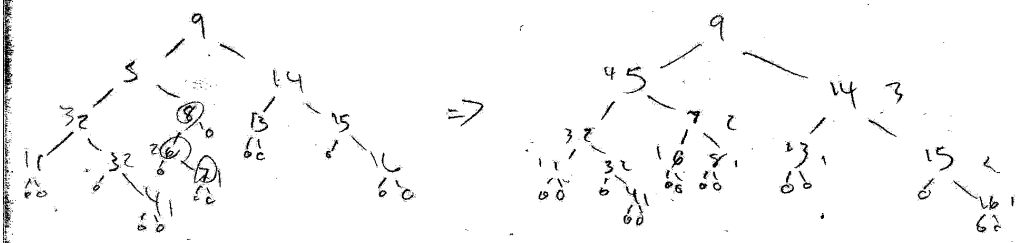
remove (10)



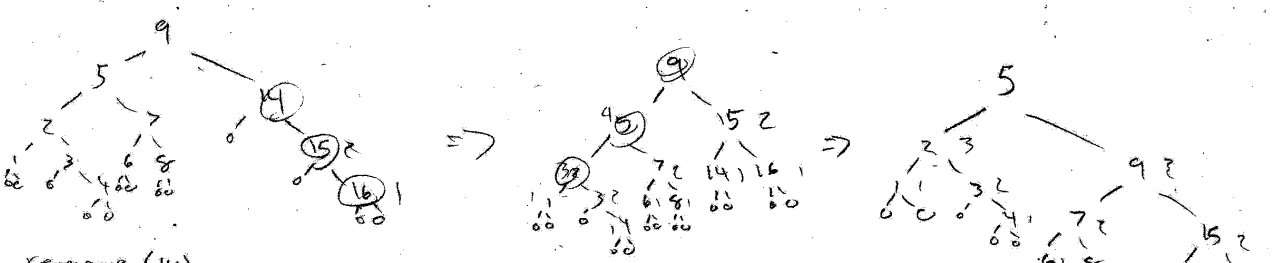
remove (4)



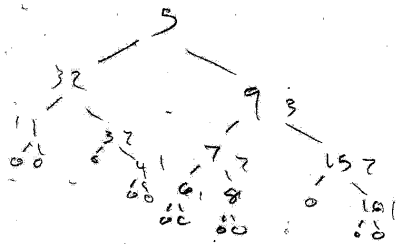
remove (12)



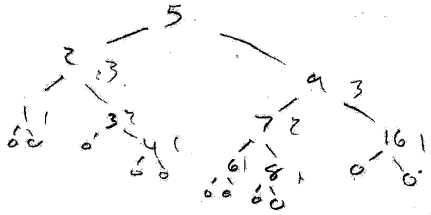
remove (13)



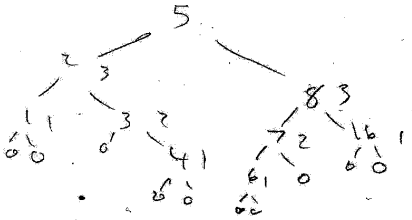
remove (14)



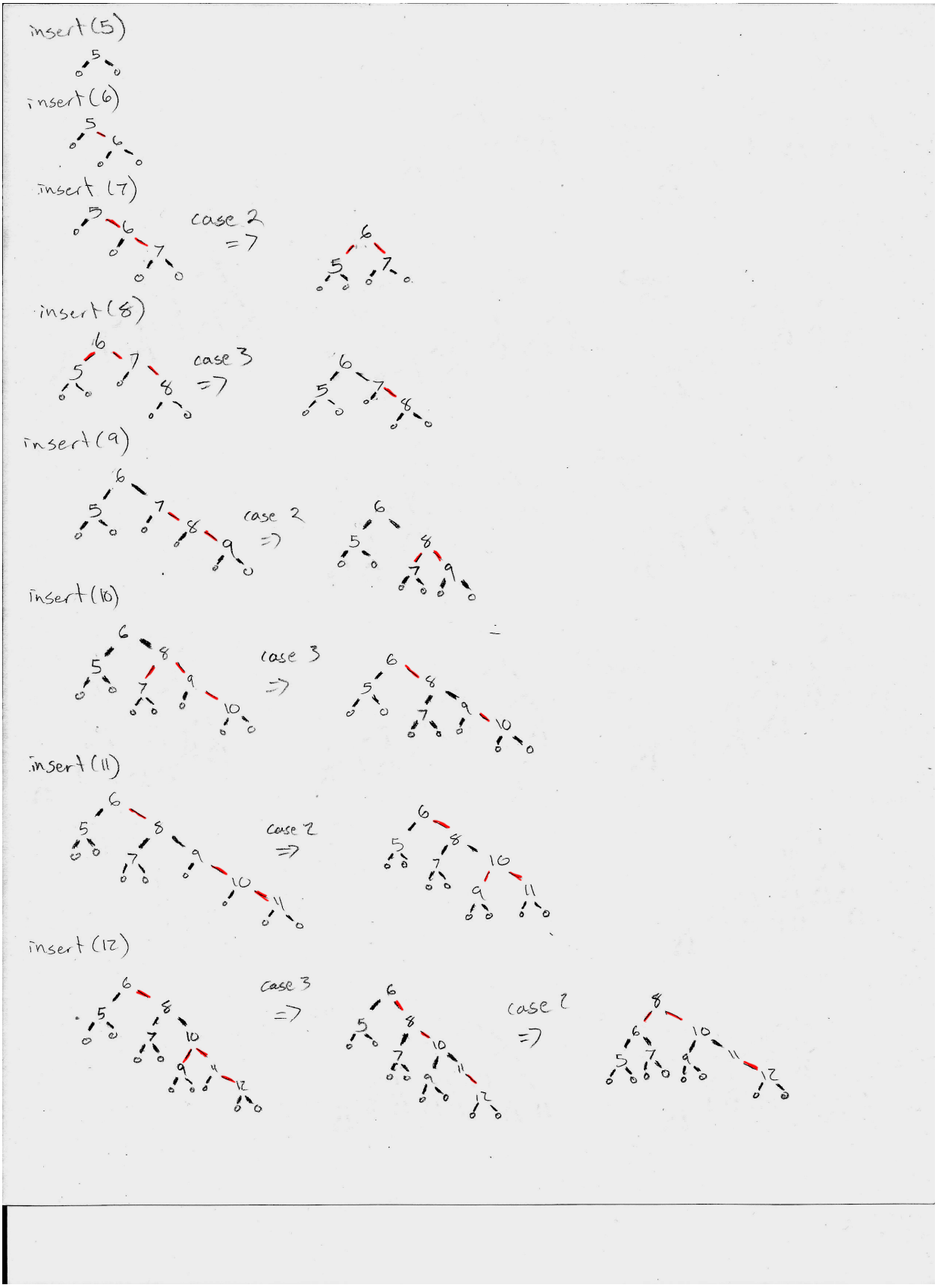
remove (15)



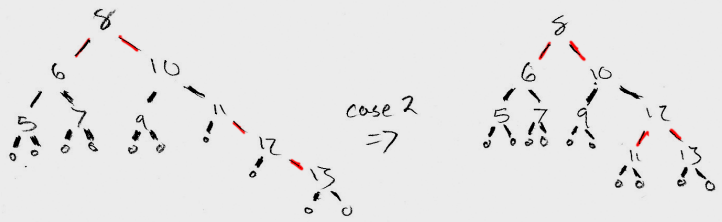
remove (9)



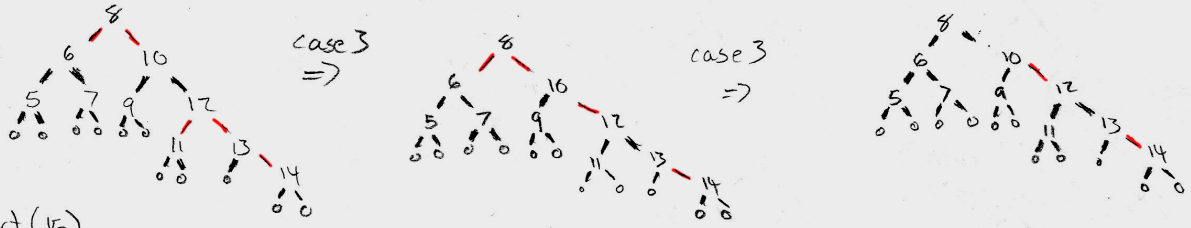
2. Starting with an empty tree, perform the operations from 1. in a red-black tree. Indicate the resulting tree after each step. Also show intermediate steps if any.



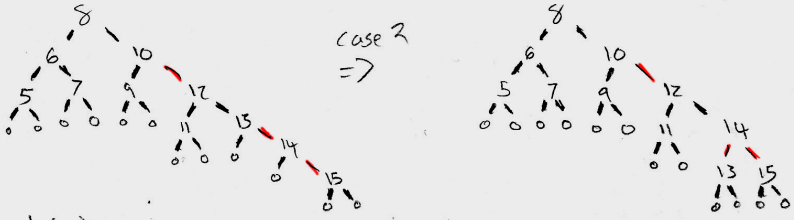
insert (13)



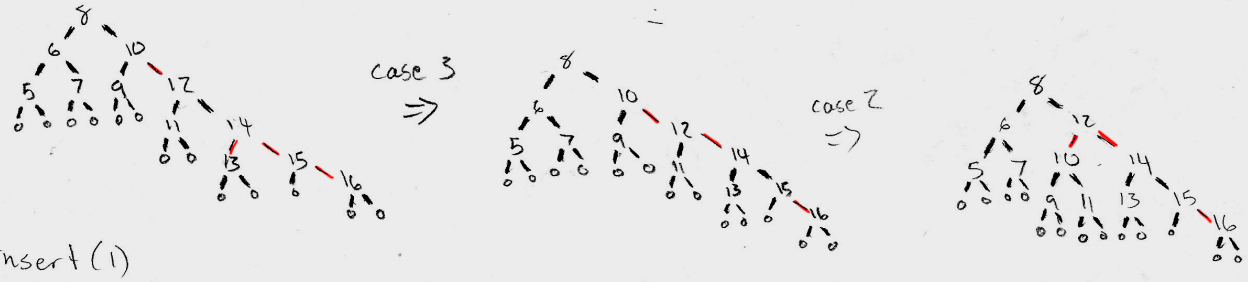
insert (14)



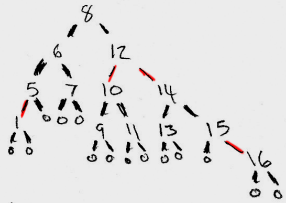
insert (15)



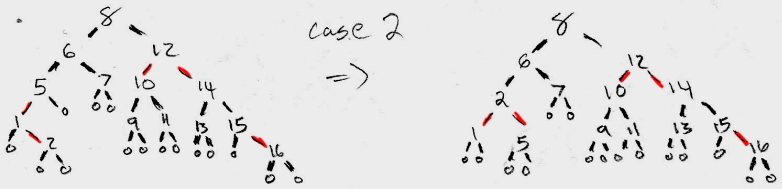
insert (16)



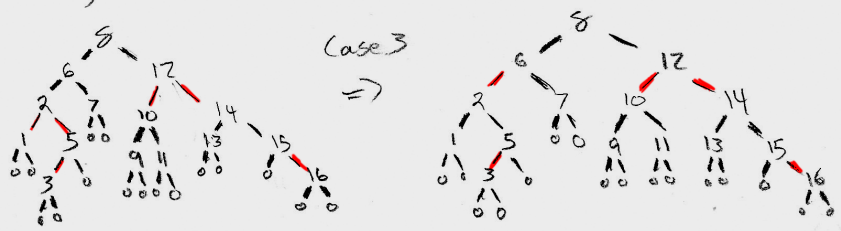
insert (1)



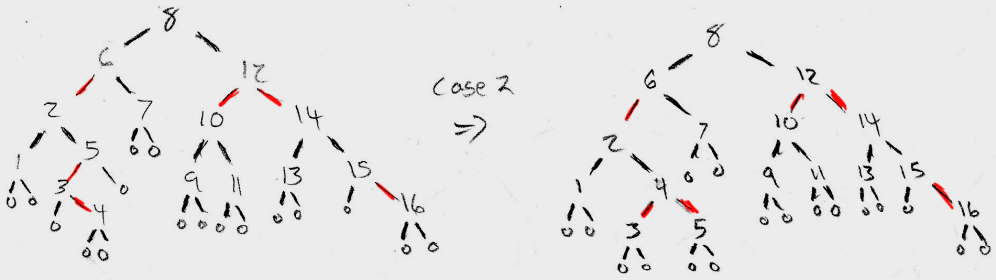
insert (2)



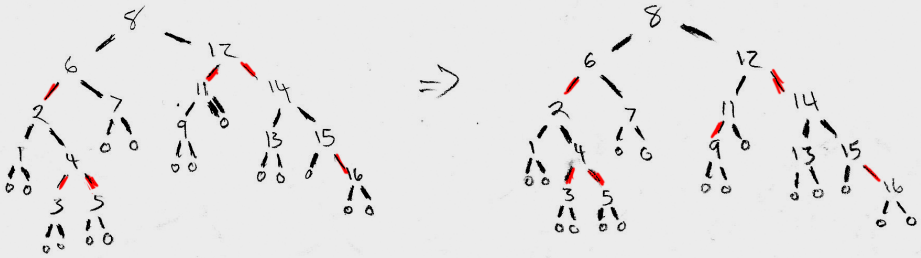
insert(3)



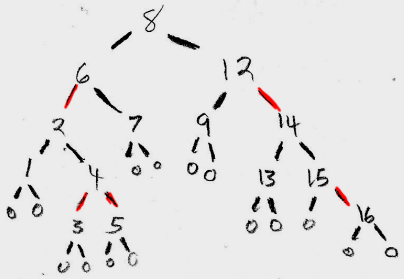
insert(4)



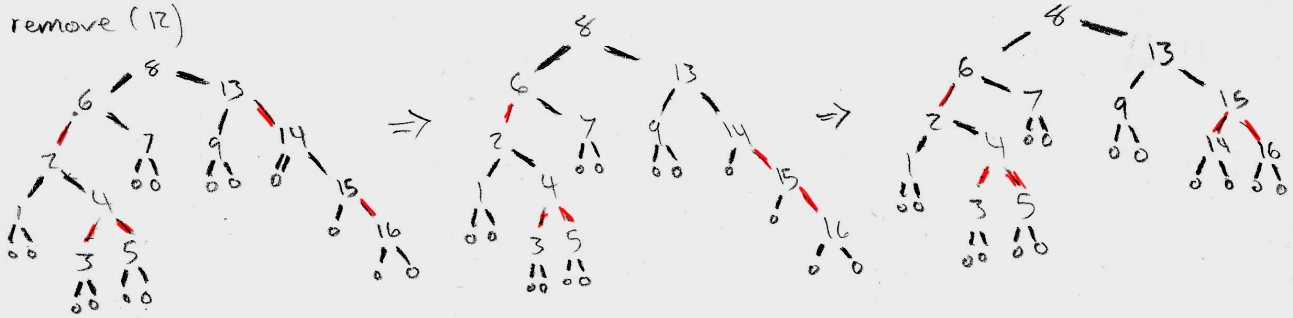
remove (10)



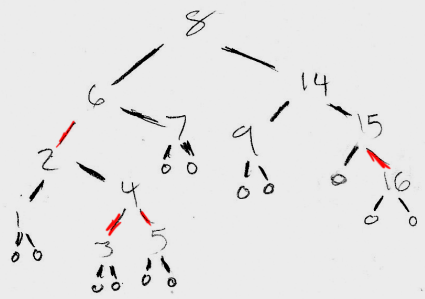
remove (11)



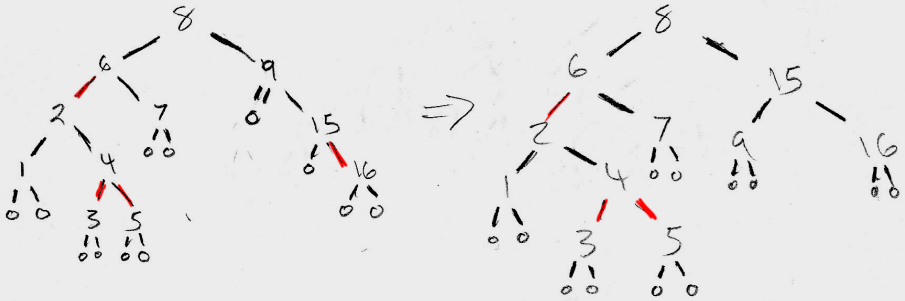
remove (12)



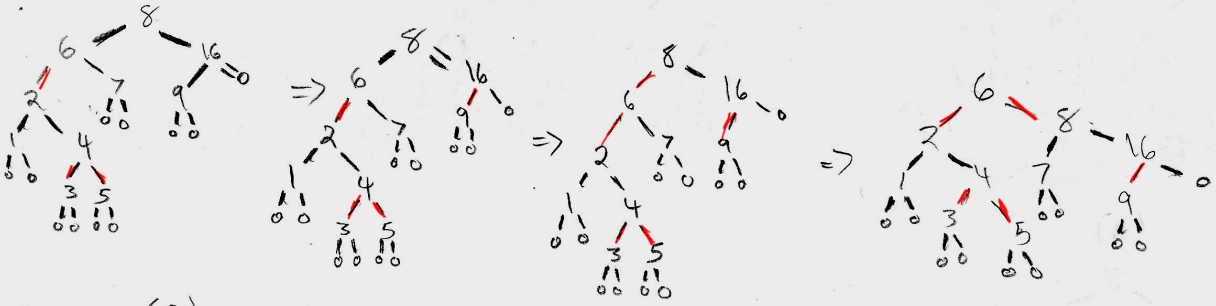
remove (13)



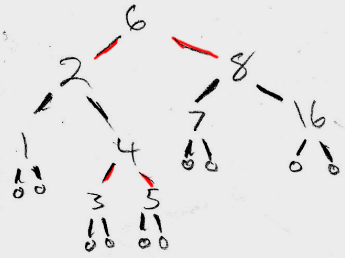
remove (14)



remove (15)



remove (9)



3. Given the sequence { 10, 6, 9, 2, 3, 11, 8, 15, 1, 4} sort the numbers using Merge Sort. Write the intermediate steps.

{10, 6, 9, 2, 3, 11, 8, 15, 1, 4}
{10, 6, 9, 2, 3} {11, 8, 15, 1, 4}
{10, 6, 9} {2, 3} {11, 8, 15} {1, 4}
{10, 6} {9} {2, 3} {11, 8} {15} {1, 4}
{10} {6} {9} {2} {3} {11} {8} {15} {1} {4}
{6, 10} {9} {2, 3} {8, 11} {15} {1, 4}
{6, 9, 10} {2, 3} {8, 11, 15} {1, 4}
{2, 3, 6, 9, 10} {1, 4, 8, 11, 15}
{1, 2, 3, 4, 6, 8, 9, 10, 11, 15}

4. Sort the same numbers using Quick Sort. Write the intermediate steps.

{10, 6, 9, 2, 3, 11, 8, 15, 1, 4}
{1, 6, 9, 2, 3, 11, 8, 15, 10, 4}
{1, 3, 9, 2, 6, 11, 8, 15, 10, 4}
{1, 3, 2, 9, 6, 11, 8, 15, 10, 4}
{1, 3, 2} {4} {9, 6, 11, 8, 15, 10}
{1, 3, 2} {4} {9, 6, 8, 11, 15, 10}
{1} {2} {3} {4} {9, 6, 8} {10} {11, 15}
{1} {2} {3} {4} {6, 9, 8} {10} {11, 15}
{1} {2} {3} {4} {6} {8} {9} {10} {11} {15}
{1, 2, 3, 4, 6, 8, 9, 10, 11, 15}

5. Sort the same numbers using Radix Sort. Use 4 bits. Write the intermediate steps.

Decimal	Binary	Decimal	Binary
10	1010	11	1011
6	0110	8	1000
9	1001	15	1111
2	0010	1	0001
3	0011	4	0100

0. { 1010, 0110, 1001, 0010, 0011, 1011, 1000, 1111, 0001, 0100 }
1. { 1010, 0110, 0010, 1000, 0100, 1001, 0011, 1011, 1111, 0001 }
2. { 1000, 0100, 1001, 0001, 1010, 0110, 0010, 0011, 1011, 1111 }
3. { 1000, 1001, 0001, 1010, 0010, 0011, 1011, 0100, 0110, 1111 }
4. { 0001, 0010, 0011, 0100, 0110, 1000, 1001, 1010, 1011, 1111 }
   { 1, 2, 3, 4, 6, 8, 9, 10, 11, 15 }

6. Using bucket sort sort the numbers { 1,2,2,6,6,6,6,6,6,3,3,3,2,2,1,1,9,9,2,2,}

{ 1,2,2,6,6,6,6,6,6,3,3,3,2,2,1,1,9,9,2,2,}

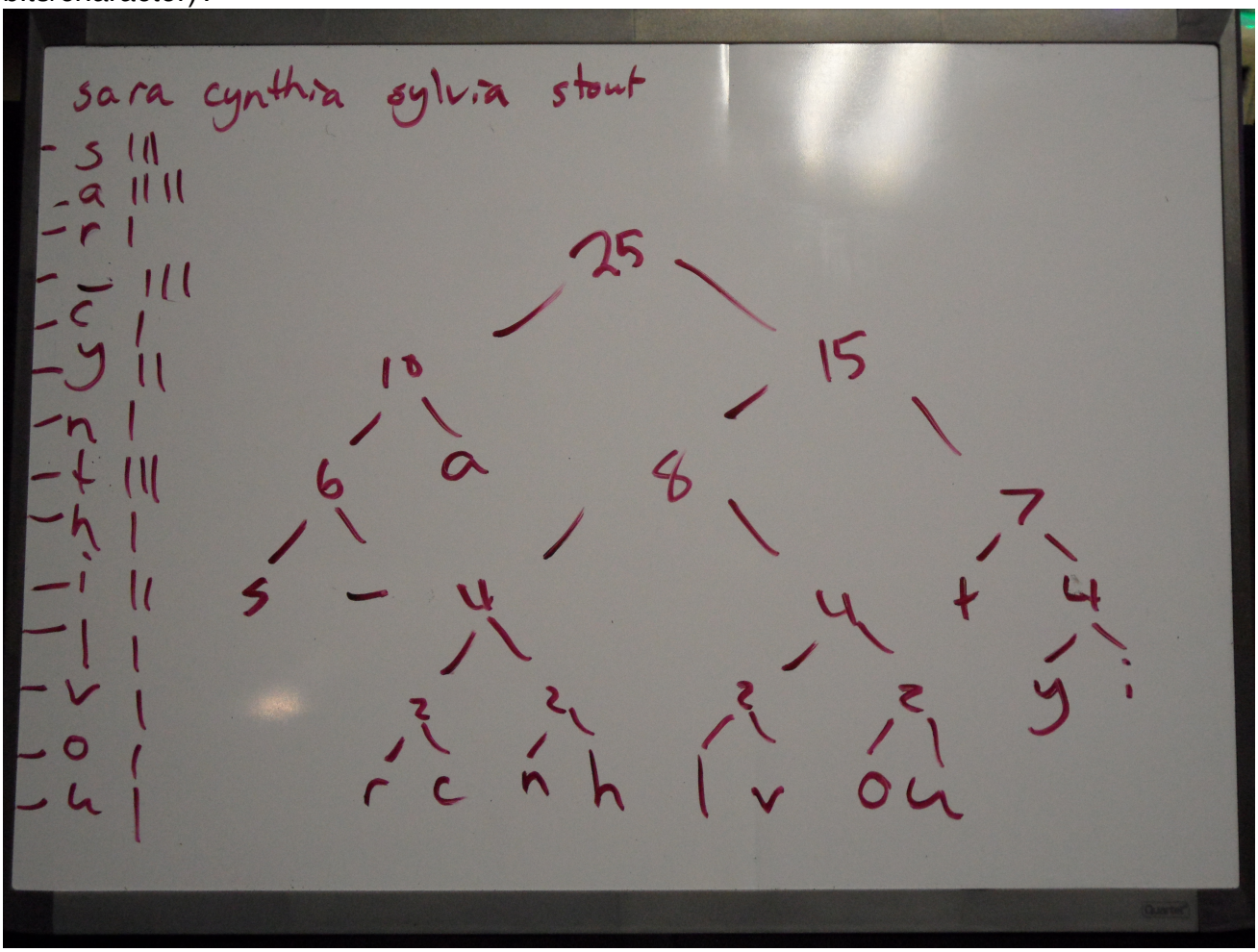
1 lll
2 llll- l
6 llll- l
3 lll
9 ll

{ 1,1,1,2,2,2,2,2,2,3,3,3,3,3,3,6,6,6,9,9 }

7. Write the complexities (O(?)) of Merge Sort, Quick Sort, Radix Sort, and Bucket Sort.

Merge sort:	O(nlogn)	
Quick sort:	O(nlogn)	Average
	$O(n^2)$	Worst
Radix sort:	O(nb)	
Bucket sort:	O(n+m)	

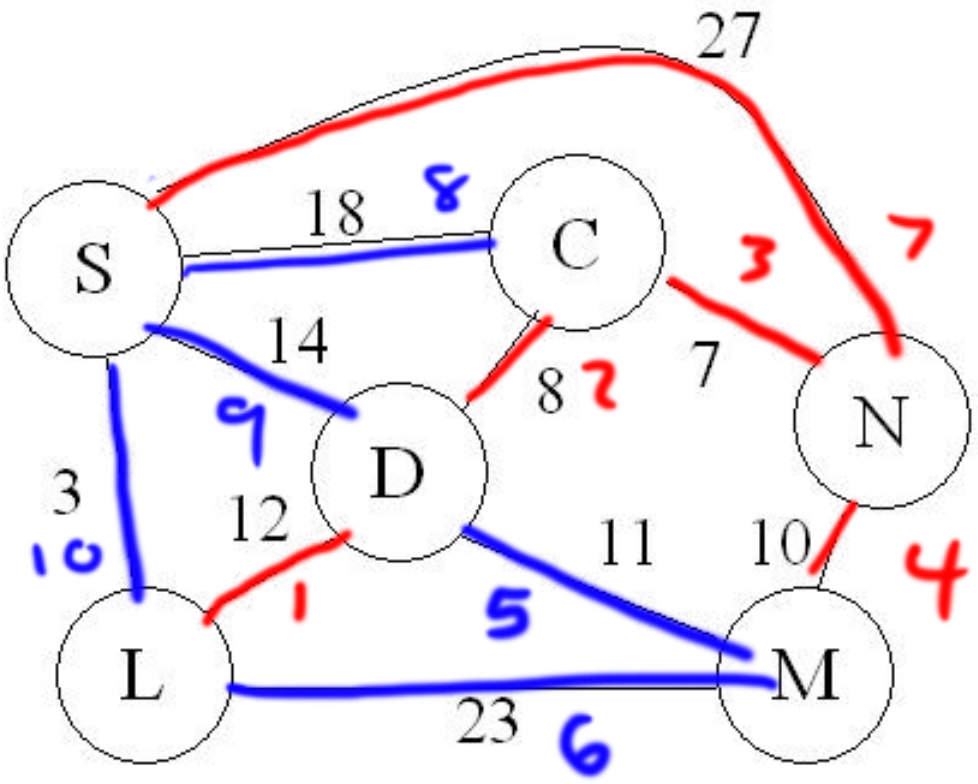
8. Find an optimal Huffman encoding trie for the sentence: "sara cynthia sylvia stout". How many bits are needed in the compressed and in the original sentence in ascii (assume Ascii encoding neds 8 bits/character)?

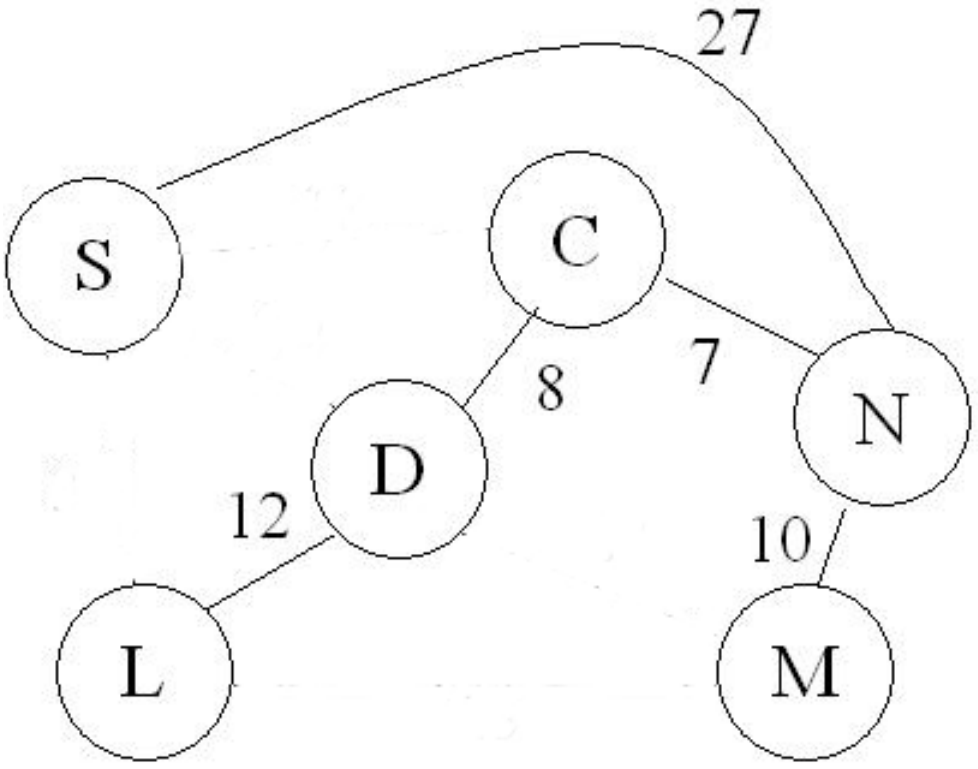


Character	Bitcode	Character	Bitcode
s	01	t	110
a	000	h	10011
r	10000	i	1111
space	001	l	10100
c	10001	v	10101
y	1110	o	10110
n	10010	u	10111

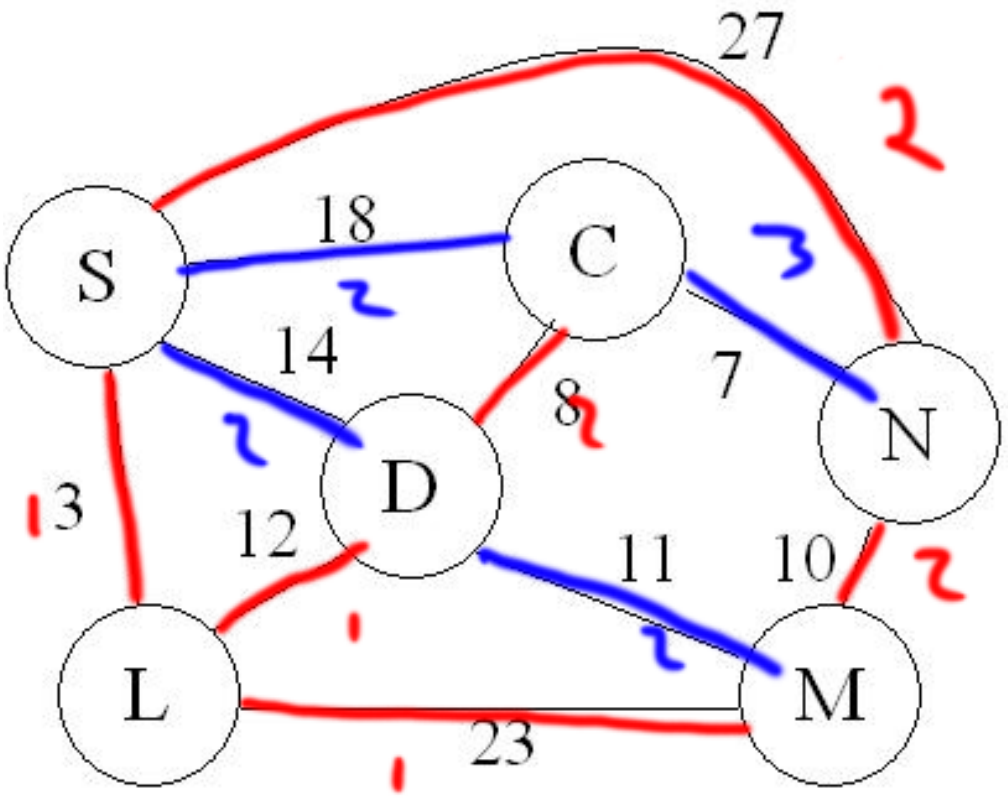
Uncompressed size: 25 characters * 8 bits/character = 200 bits
Compressed size: 90 bits

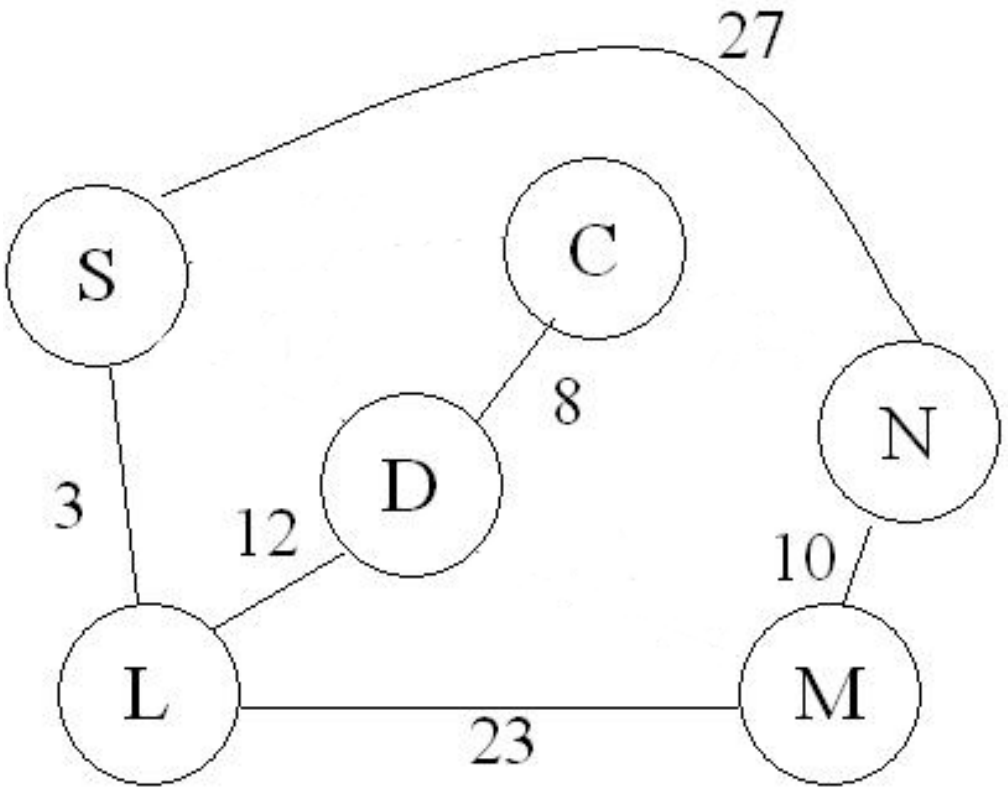
9. Given the following graph, find the Depth First Search Tree of the following graph starting at node L. The DFS tree includes only the discovery edges and the vertices. Assume that the vertices appear in alphabetical order in the adjacency list used in the traversal.



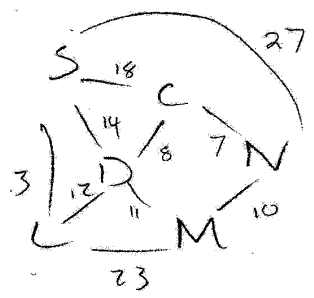


10. Using the graph in question 8, find the Breadth First Search Tree of the graph in 9. starting at node L. The BFS tree includes only the discovery edges and the vertices.





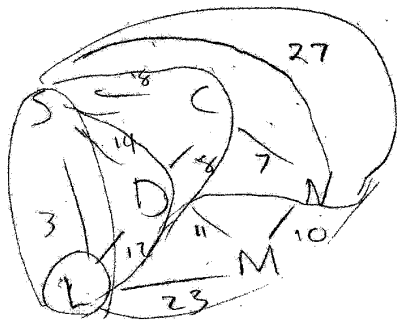
11 Using Dijkstra's algorithm find the shortest path tree of the graph in question 9 starting at vertex L.



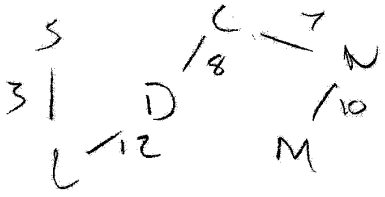
S C D N M L

S ∞	3	3	3	3	3	3
C ∞	∞	21	20	20	20	20
D ∞	12	12	12	12	12	12
N ∞	∞	30	30	27	27	27
M ∞	23	23	23	23	23	23
L 0	0	0	0	0	0	0
	L	S	D	C	M	N

12.Using the Prim-Jarnick algorithm, find the minimum spanning of the graph in 9. starting at vertex L.



⇒



1. (L, S)
2. (L, D)
3. (D, C)
4. (C, N)
5. (N, M)

13. Implement in "C" the method

```
#define MaxVertices 50
void shortest_path(int v, int n,
          int weight[MaxVertices][MaxVertices],
          int aDistance[MaxVertices],
          int parent[MaxVertices] )
```

that returns in aDistance[i] the shortest distance from vertex v to vertex i and in parent[i] the previous vertex in the shortest path from v to i. n is the number of vertices and weight[i][j] is the weight in the edge (i,j). If edge (i,j) does not exist,. then weight[i][j] is -1.

```
#define MaxVertices 50
void shortest_path(int v, int n,
           int weight[MaxVertices][MaxVertices],
           int aDistance[MaxVertices],
           int parent[MaxVertices] ) {

  int i;

  int [] visited = new int[n];

  int MaxDistance = 1000000000;
  for (i=0; i < n; i++) {
     aDistance[i]=MaxDistance;
     visited[i] = 0;
  }
  aDistance[v]=0;

  parent[v]=v;

  while (1) {

     int u = -1;
     for (i=0; i < n; i++) {
          if (!visited[i]) {
               if (u<0 || aDistance[i] < aDistance[u]) {
                    u=i;
               }
          }
     }

     if (u == -1) {
          return;
     }

     visited[u]=1;

     int z;
     for (z = 0; z < n; z++) {

          if (weight[u][z]) {
               if (aDistance[z] > aDistance[u]+weight[u][z]) {
                    aDistance[z] = aDistance[u]+weight[u][z];
```

```
                        parent[z]=u;
                    }
                }
            }
        }
    }
```

14. Write the implementation of the remove(int key) method of the AVL tree that removes the node with that key. You may base your implementation with the one given in class. Also include the implementation of any auxiliary procedures you use.

```
// Removes one element from the dictionary
bool
AVLTree::removeElement(KeyType key)
{

  bool found = false;

  AVLNode * n = _root;
  while (n)     {
      int cmp = strcmp(n->_key,key);
      if (cmp < 0) {
            n = n->_right;
      } else if (cmp > 0) {
            n = n->_left;
      } else {
            found = true;
            break;
      }
  }

  if (found) {
      int side = 0;
      int maxh = 0;
      // find the highest child, if any
      if (n->_left) {
            side = 1;
            maxh = n->_left->_height;
      }
      if (n->_right && n->_right->_height > maxh) {
            side = 2;
      }
      AVLNode * c = NULL;                        // substitution node
      AVLNode * fix = n->_parent;// node from which to restructure
      // traverse child subtree
      if (side != 0) {
            // find leaf closest to n's key
            c = (side == 1) ? n->_left : n->_right;
            while ((side == 1) ? c->_right : c->_left) {
                  c = (side == 1) ? c->_right : c->_left;
```

```
            }

            // figure out node from which to update height
            if (c->_parent != n) {
                  fix = c->_parent;
            } else {
                  fix = c;
            }

            // adjust c's parent linkage
            if (c->_parent) {
                  if (c->_parent->_left == c) {
                        c->_parent->_left = c->_right;
                  } else {
                        c->_parent->_right = c->_left;
                  }
            }
            c->_left = n->_left;
            c->_right = n->_right;
            c->_parent = n->_parent;
            updateHeight(c);
      }
      // adjust parent linkage
      if (n->_parent) {
            if (n->_parent->_left == n) {
                  n->_parent->_left = c;
            } else {
                  n->_parent->_right = c;
            }
      } else {
            _root = c;
      }

      if (c && c->_left) {
            c->_left->_parent = c;
      }
      if (c && c->_right) {
            c->_right->_parent = c;
      }
      delete n;
      // Restructure code is given in notes, so I won't retype it.
      // The only difference is that z isn't initially set to the
      // parent of the passed-in node
      restructure(fix);
  }
  return found;
}


// Update the height of an AVLNode
void
updateHeight(AVLNode * n){
  int maxh = 0;
```

```
  if (n->_left) {
     maxh = n->_left->_height;
  }
  if (n->_right && maxh < n->_right->_height) {
     maxh = n->_right->_height;
  }
  n->_height = maxh+1;
}
```

15. Write the code for quicksort that sorts a sequence of n integers stored in array a. Include the code of any auxiliary functions.

```
void quicksort(int * a, int n) {
  if (n < 2) {
     return;
  }
  int x = a[n-1];
  int l = 0;
  int r = n-2;
  while (l < r) {
     while (l < r && a[l] < x) {
          l++;
     }
     while (l < r && a[r] > x) {
          r--;
     }
     if (l < r) {
          int temp = a[l];
          a[l] = a[r];
          a[r] = temp;
     }
  }
  int temp = a[l];
  a[l] = a[n-1];
  a[n-1] = temp;
  quicksort(a,l);
  quicksort(a+l+1,n-l-1);
}
```

16. Write the code for mergesort that sorts a sequence of n integers stored in array a. Include the code of any auxiliary functions.

```
void mergesort(int * a,int n) {
  if (n < 2){
     return;
  }
  int n1 = n/2;
  int n2 = n-n1;
  int * s1 = new int[n1];
  int * s2 = new int[n2];
  int i;
```

```
  for (i = 0; i < n1; i++) {
     s1[i] = a[i];
  }
  for (i = 0; i < n2; i++) {
     s2[i] = a[i+n1];
  }
  mergesort(s1,n1);
  mergesort(s2,n2);
  i = 0;
  int i1 = 0, i2 = 0;
  while (i1 < n1 && i2 < n2) {
     if (s1[i1] < s2[i2]) {
          a[i++] = s1[i1++];
     } else {
          a[i++] = s2[i2++];
     }
  }
  while (i1 < n1) {
     a[i++] = s1[i1++];
  }
  while (i2 < n2) {
     a[i++] = s2[i2++];
  }
  delete [] s1;
  delete [] s2;
  return;
}
```