CSE 5311 Homework #1 Solutions

Fall 2002

1 Binary Search Tree

Find the optimal binary search tree for the given keys and probabilities. The optimal subtree for all members of each family is given in Figure 1.

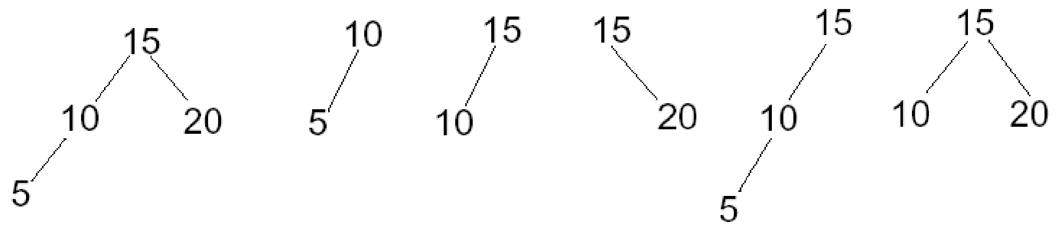

Figure 1: Optimal BST

The code (http://reptar.uta.edu/NOTES5311/obst.c) is as follows:

```
// Optimal binary search tree

#include <stdio.h>

#define MAXKEYS (30)

int n; // number of keys
int key[MAXKEYS+1];
float p[MAXKEYS+1]; // probability of hitting key i
float q[MAXKEYS+1]; // probability of missing between key[i-1] and
key[i]
float c[MAXKEYS+1][MAXKEYS+1]; // cost of subtree
int r[MAXKEYS+1][MAXKEYS+1]; // root of subtree
float w[MAXKEYS+1][MAXKEYS+1]; // accumulated p and q

void opttree()
{
float x,min;
int i,j,k,h,m;

for (i<0;i<=n;i++)
  c[i][i]=0;  // width of tree h=0
for (i=0;i<n;i++) // width of tree h=1
{
  j=i+1;
  c[i][j]=w[i][j];
  r[i][j]=j;
}
for (h=2;h<=n;h++)
  for (i=0;i<=n-h;i++)
  {
    j=i+h;
    printf("Building c(%d,%d) using roots %d thru %d\n",
      i,j,r[i][j-1],r[i+1][j]);
    m=r[i][j-1];
    min=c[i][m-1]+c[m][j];
    for (k=m+1;k<=r[i+1][j];k++)
    {
      x=c[i][k-1]+c[k][j];
      if (x<min)
      {
        m=k;
        min=x;
      }
    }
    c[i][j]=min+w[i][j];
    r[i][j]=m;
  }
}

void prefix(int i,int j)
// prints optimal binary search tree
{
if (i<j)
{
  printf("%d",key[r[i][j]]);
  if (i<r[i][j]-1 && r[i][j]<j)
  {
    printf("(");
    prefix(i,r[i][j]-1);
    printf(",");
    prefix(r[i][j],j);
    printf(")");
  }
  else if (i<r[i][j]-1)
  {
    printf("(");
    prefix(i,r[i][j]-1);
    printf(",");
    printf(")");
  }
  else if (r[i][j]<j)
  {
    printf("(");
    printf(",");
    prefix(r[i][j],j);
    printf(")");
  }
}
}

main()
{
int i,j;

n=4;
q[0]=0.01;
key[1]=5;
p[1]=0.03;
```

```
q[1]=0.02;
key[2]=10;
p[2]=0.16;
q[2]=0.08;
key[3]=15;
p[3]=0.20;
q[3]=0.20;
key[4]=20;
p[4]=0.1;
q[4]=0.2;

for (i=0;i<=n;i++)
{
  w[i][i]=q[i];
  printf("w[%d][%d]=%f\n",i,i,w[i][i]);
  for (j=i+1;j<=n;j++)
  {
    w[i][j]=w[i][j-1]+p[j]+q[j];
    printf("w[%d][%d]=%f\n",i,j,w[i][j]);
  }
}
opttree();
printf("Average probe length is %f\n",c[0][n]/w[0][n]);
printf("trees in parenthesized prefix\n");
for (i=0;i<=n;i++)
  for (j=0;j<=n-i;j++)
  {
    printf("c(%d,%d) cost %f ",j,j+i,c[j][j+i]);
    prefix(j,j+i);
    printf("\n");
  }
}
```

The outputs are as follows:

```
w[0][0]=0.010000
w[0][1]=0.060000
w[0][2]=0.300000
w[0][3]=0.700000
w[0][4]=1.000000
w[1][1]=0.020000
w[1][2]=0.260000
w[1][3]=0.660000
w[1][4]=0.960000
w[2][2]=0.080000
w[2][3]=0.480000
w[2][4]=0.780000
w[3][3]=0.200000
w[3][4]=0.500000
w[4][4]=0.200000
Building c(0,2) using roots 1 thru 2
Building c(1,3) using roots 2 thru 3
Building c(2,4) using roots 3 thru 4
Building c(0,3) using roots 2 thru 3
Building c(1,4) using roots 3 thru 4
Building c(0,4) using roots 3 thru 3
```

```
Average probe length is 1.860000
trees in parenthesized prefix
c(0,0) cost 0.000000
c(1,1) cost 0.000000
c(2,2) cost 0.000000
c(3,3) cost 0.000000
c(4,4) cost 0.000000
c(0,1) cost 0.060000 5
c(1,2) cost 0.260000 10
c(2,3) cost 0.480000 15
c(3,4) cost 0.500000 20
c(0,2) cost 0.360000 10(5,)
c(1,3) cost 0.920000 15(10,)
c(2,4) cost 1.260000 20(15,)
c(0,3) cost 1.060000 15(10(5,),)
c(1,4) cost 1.720000 15(10,20)
c(0,4) cost 1.860000 15(10(5,),20)
```

2 Proof of Probe Length

Prove that the optimal static list for n elements with Zipf's distribution has an average probe length of $\frac{n}{H_n}$.

$$\sum_{i=1}^{n} p_i c_i = \sum_{i=1}^{n} i p_i = \sum_{i=1}^{n} i \frac{1}{iH_n} = \frac{1}{iH_n} \sum_{i=1}^{n} 1 = \frac{n}{H_n}$$

3 Upper Bound by Substitution

Find an asymptotic upper bound on the recurrence $T(n) = T(\frac{n}{5}) + n$ by using substitution.

Guess: $T(n) = O(n)$. So we need to prove that $T(n) \leq cn$ for some $c > 0$. Assume this hold for $T(\frac{n}{5})$ (ie. $T(\frac{n}{5}) \leq c\frac{n}{5}$). Substituting into the recurrence yields

$$\begin{aligned} T(n) &\leq c\tfrac{n}{5} + n \\ &= (\tfrac{c}{5} + 1)n \\ &= (c - \tfrac{4c}{5} + 1)n \\ &\leq cn \text{ if } c \geq \tfrac{5}{4} \end{aligned}$$

Therefore we have $T(n) = O(n)$.

4 Upper Bound by Iteration

Find an asymptotic upper bound on the recurrence $T(n) = 3T(\frac{n}{4}) + \frac{1}{2}n$ by using iteration.

$$\begin{aligned} T(n) &= \tfrac{n}{2} + 3T(\tfrac{n}{4}) \\ &= \tfrac{n}{2} + 3\left[\tfrac{n}{8} + 3T(\tfrac{n}{16})\right] \\ &= \tfrac{n}{2} + 3\left[\tfrac{n}{8} + 3\left[\tfrac{n}{32} + 3T(\tfrac{n}{64})\right]\right] \\ &= \tfrac{n}{2} + \tfrac{3n}{8} + \tfrac{9n}{32} + 27T(\tfrac{n}{64}) \end{aligned}$$

5 Upper and Lower Bound by Substitution

Find an asymptotic upper and lower bound on the recurrence $T(n) = 3T(\frac{n}{3}) + 1$ by using substitution.

1. Show that $T(n)$ is in $\Omega(n)$ (ie. $T(n) \geq cn$ for some $c > 0$.

   Assume $T(\frac{n}{3}) \geq c\frac{n}{3}$. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\geq 3c\tfrac{n}{3} + 1 \\ &= cn + 1 \\ &\geq cn \end{aligned}$$

   Thus $T(n)$ is in $\Omega(n)$.

2. Show that $T(n)$ is in $O(n)$ (ie. $T(n) \leq cn$ for some $c > 0$.

   Assume $T(\frac{n}{3}) \leq c\frac{n}{3}$. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\leq 3c\tfrac{n}{3} + 1 \\ &= cn + 1 \end{aligned}$$

   which does not imply $T(n) \leq cn$ for any choice of c. This difficulty can be overcome by subtracting a lower-order term from our previous guess. Our new guess is $T(n) \leq cn - b$ where $b \geq 0$ is a constant. We now assume $T(\frac{n}{3}) \leq c\frac{n}{3} - b$. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\leq 3(c\tfrac{n}{3} - b) + 1 \\ &= cn - 3b + 1 \\ &\leq cn - b \text{ as long as } b \geq \tfrac{1}{2} \end{aligned}$$

   Thus $T(n)$ is in $O(n)$.

Combining these two parts gives $T(n)$ is in $\Theta(n)$.

6 Upper Bound by Iteration

Use the iteration method to show that $f(n) = n$ is an asymptotic upper bound on $T(n) = 3T(\frac{n}{3}) + 1$.

$$\begin{aligned} T(n) &= 1 + 3T(\tfrac{n}{3}) \\ &= 1 + 3\left[1 + 3T(\tfrac{n}{9})\right] \\ &= 1 + 3\left[1 + 3\left[1 + 3T(\tfrac{n}{27})\right]\right] \\ &= 1 + 3 + 9 + 27T(\tfrac{n}{27}) \end{aligned}$$

The ith term is 3^i. The iteration stops when $\frac{n}{3^i} = 1$ or $i = \log_3 n$ (assumming $T(1) = \Theta(1)$). Thus, we have

$$T(n) = \underbrace{1 + 3 + 9 + \cdots}_{\log_3 n \text{ terms}} = \sum_{i=0}^{\log_3 n} 3^i = \frac{\log_3 n + 1}{3 - 1} = \frac{3n - 1}{2} = O(n)$$

7 Red-Black Tree Insertion

Figure 2 shows the modifications to the tree resulting from the addition of 160. Nodes which have modified R/B values are depicted by a dashed outline. An empty square represents an unchanged subtree.

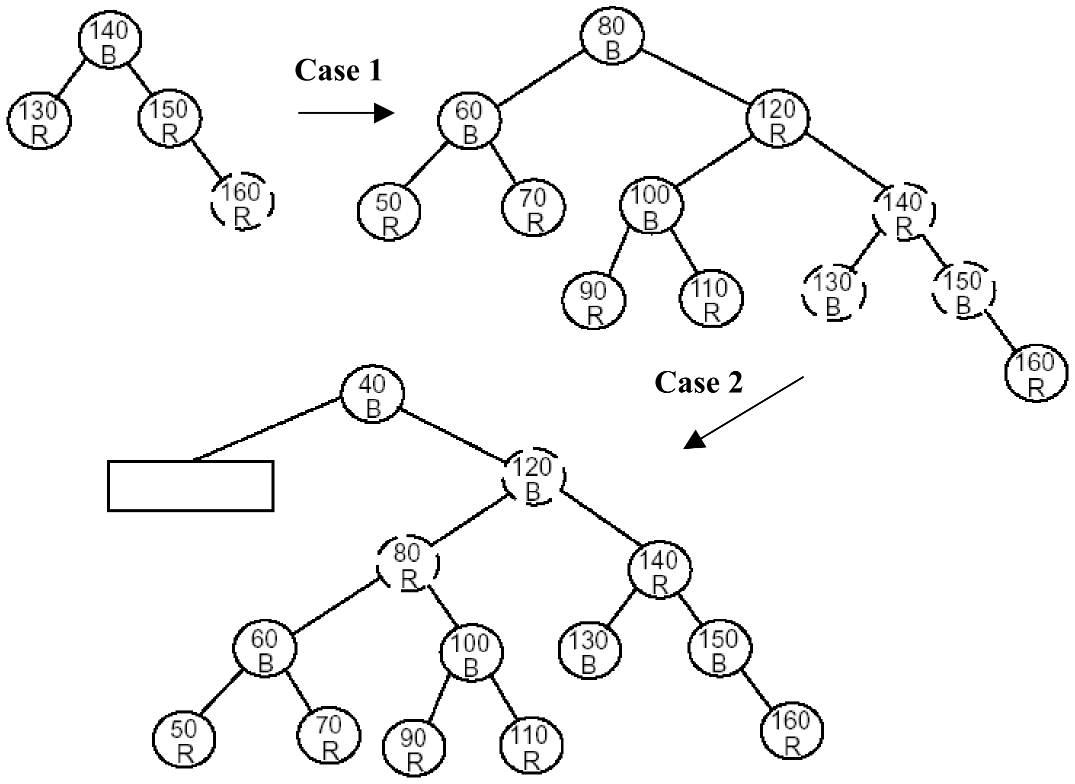

Figure 2: Insertion of 160 into Red-Black Tree

8 Red-Black Tree Insertion

Figure 3 shows the modifications to the tree resulting from the addition of 95. Nodes which have modified R/B values are depicted by a dashed outline. An empty square represents an unchanged subtree.

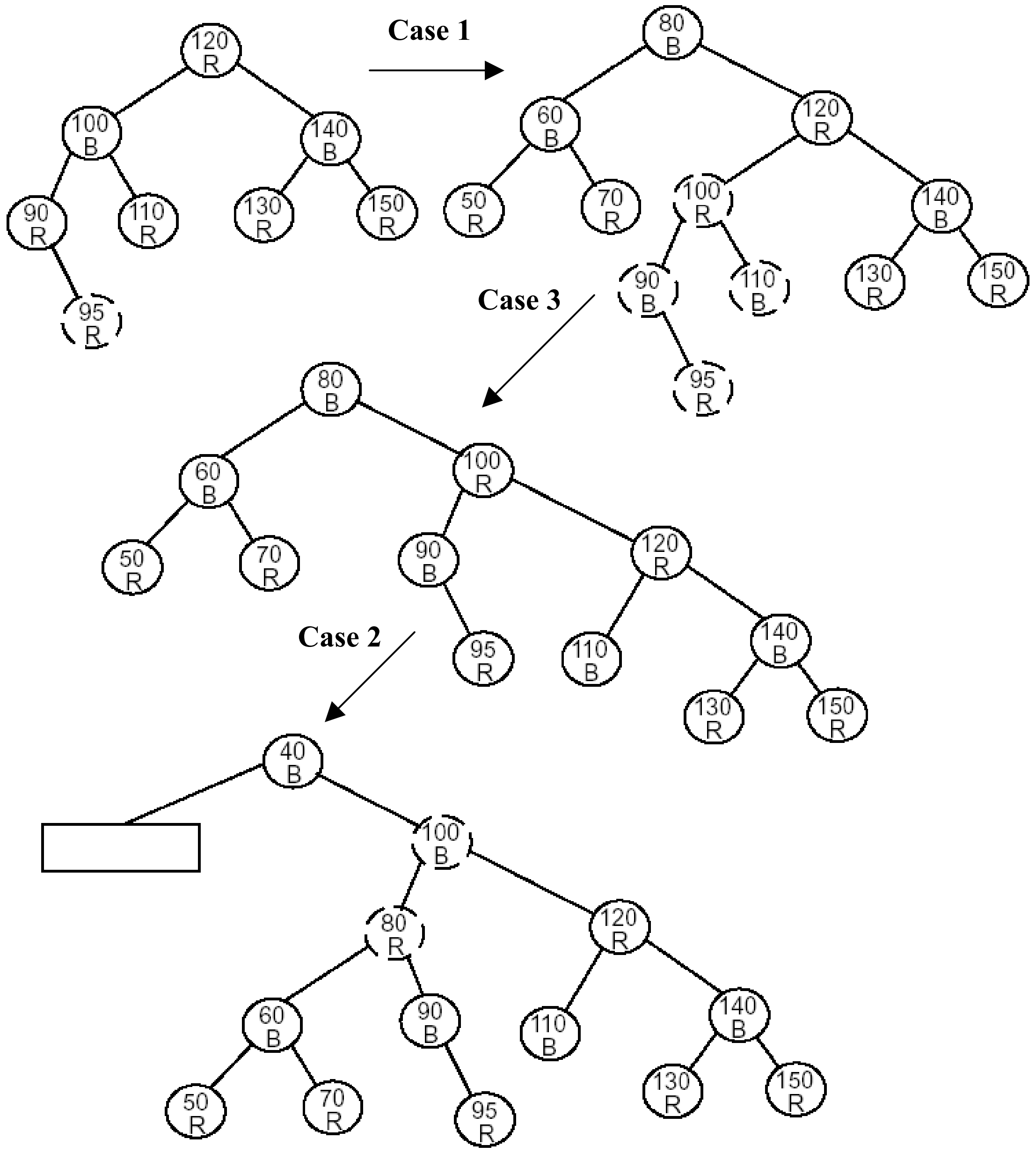

Figure 3: Insertion of 95 into Red-Black Tree

9 Red-Black Tree Deletion

Figure 4 shows the modifications to the tree resulting from the deletion of 80. Nodes which have modified R/B values are depicted by a dashed outline. An empty square represents an unchanged subtree.

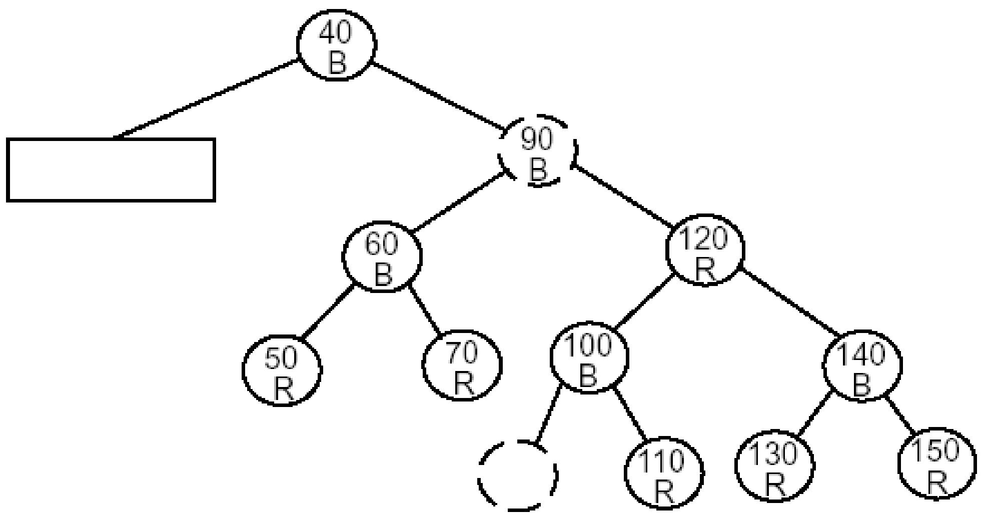

Figure 4: Deletion of 80 from Red-Black Tree

10 Red-Black Tree Deletion

Figure 5 shows the modifications to the tree resulting from the deletion of 30. Nodes which have modified R/B values are depicted by a dashed outline. An empty square represents an unchanged subtree.

11 AVL Tree Insertion and Deletion

Figures 6 and 7 show various stages of the insertions. Figure 8 shows the results of the specified deletions.

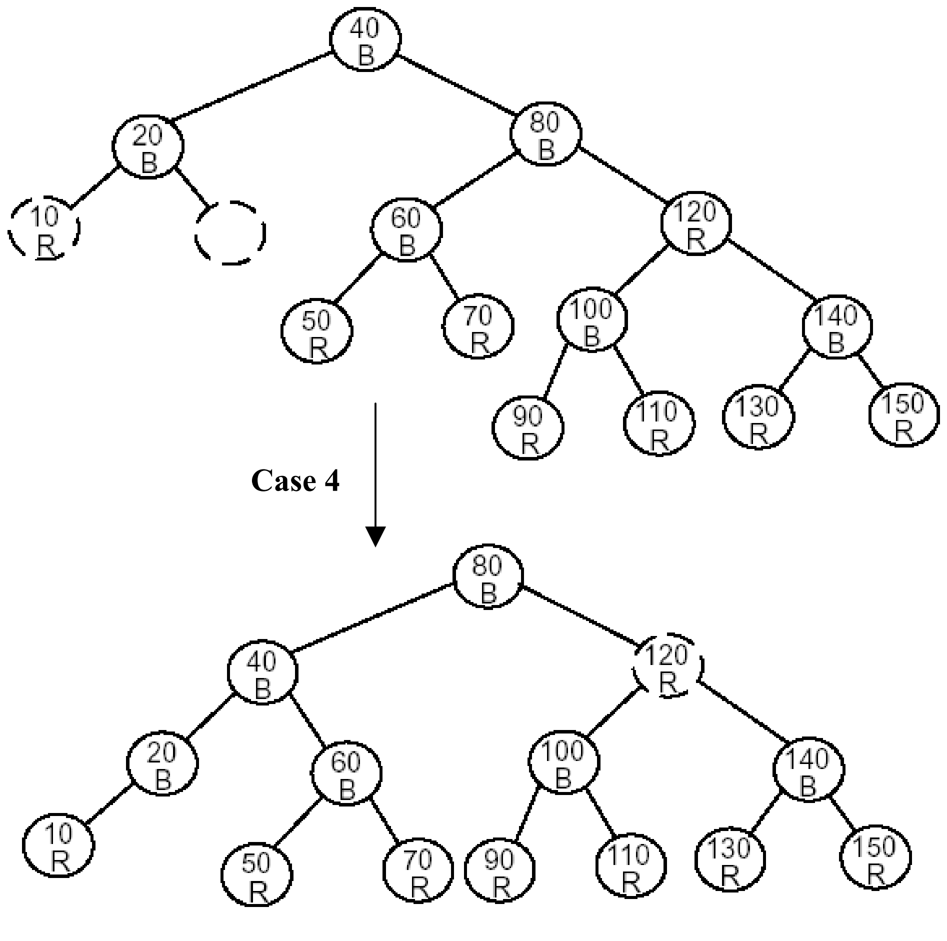

Figure 5: Deletion of 30 from Red-Black Tree

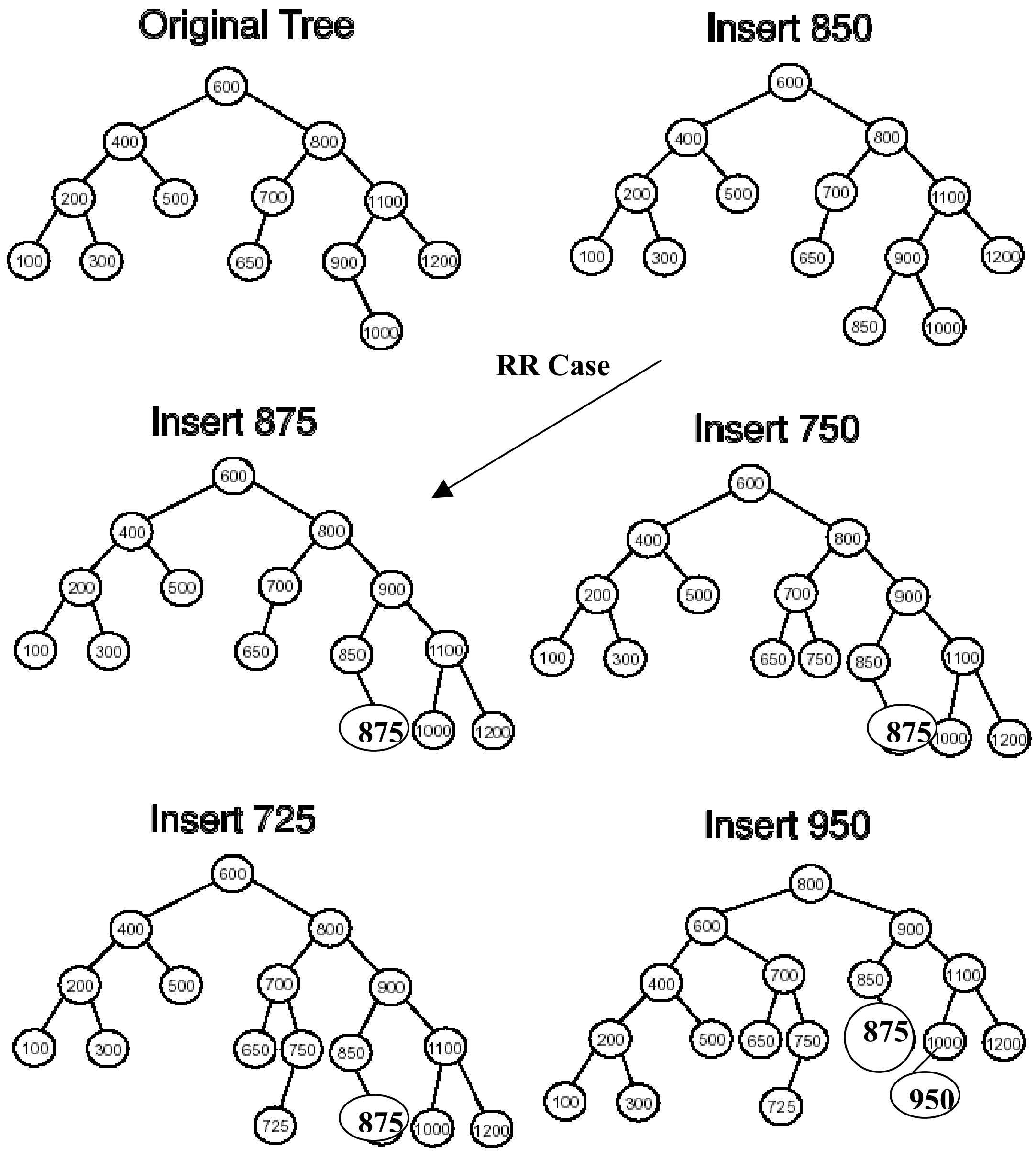

Figure 6: AVL Insertion of 850 thru 950

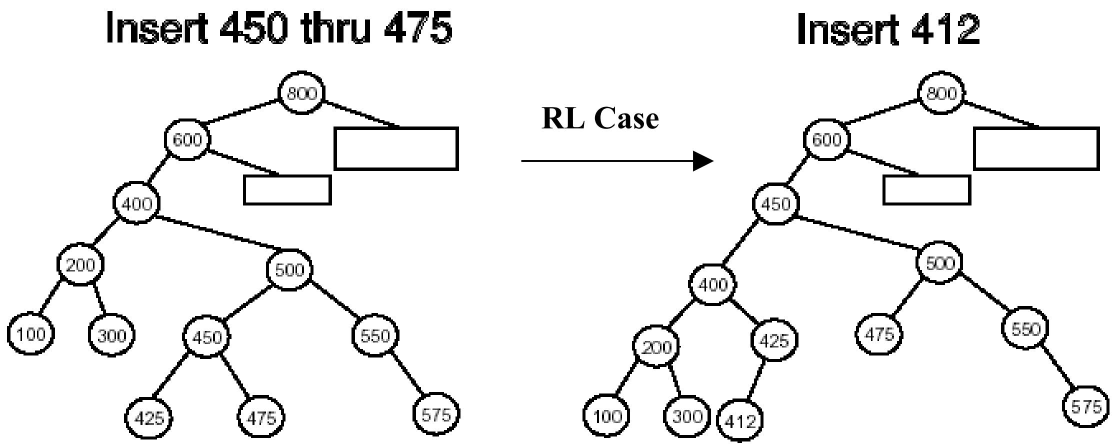

Figure 7: AVL Insertion of 450 thru 412

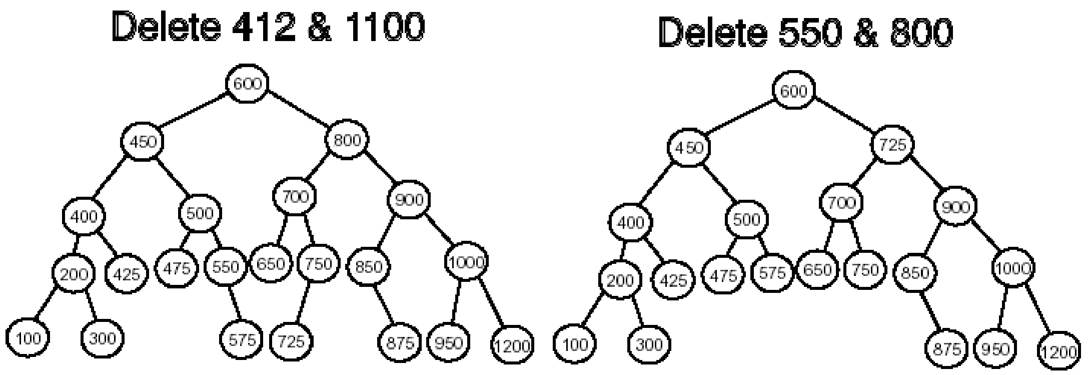

Figure 8: AVL Deletion of 412 thru 480

12 Problem 4.3-1

a)

$$T(n) = 4T(\tfrac{n}{2}) + n$$
$$a = 4, b = 2, f(n) = n$$
$$n^{\log_b a} = n^{\log_2 4} = n^2$$
$$f(n) = n = O(n^{\log_b a - \epsilon}), \epsilon(n^2) \text{ case } 1$$
$$\text{Thus } T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

b)

$$T(n) = 4T(\tfrac{n}{2}) + n^2$$
$$a = 4, b = 2, f(n) = n^2$$
$$f(n) = n^2 = \Theta(n^{\log_2 4}) \text{ case } 2$$
$$\text{Thus } T(n) = \Theta(n^2 \lg n)$$

c)

$$T(n) = 4T(\tfrac{n}{2}) + n^3$$
$$a = 4, b = 2, f(n) = n^3$$
$$f(n) = \Omega(n^{\log_2 4 + \epsilon}), \epsilon = 1 > 0 \text{ case } 3$$
$$af(\tfrac{n}{b}) \leq cf(n) \text{ for some } c < 1$$
$$\text{or } 4\tfrac{n^3}{8} \leq cn^3$$
$$\text{or } \tfrac{1}{2} \leq c \text{ which is true for } \tfrac{1}{2} \leq c < 1$$
$$\text{Thus } T(n) = \Theta(n^3)$$

13 Problem 4.3-2

$$
\begin{aligned}
&T(n) = 7T(\tfrac{n}{2}) + n^2\\
&a = 7, b = 2, f(n) = n^2\\
&n^{\log_b a} = n^{\log_2 7} \sim n^{2.8}\\
&f(n) = n^2 = O(n^{\log_2 7 - 0.8}), \epsilon = 0.8 > 0 \text{ case } 1\\
&\text{Thus } T(n) = \Theta(n^{\log_2 7})
\end{aligned}
$$

$$
\begin{aligned}
&T'(n) = aT'(\tfrac{n}{4}) + n^2\\
&b = 4, f(n) = n^2\\
&n^{\log_b a} = n^{\log_4 a}\\
&n^{\log_4 a} = n^{\log_2 7} \text{ or } n^{\log_4 a} = n^{\log_2 7}\\
&\text{Thus } a = 4^{\log_2 7} \approx 48.50293
\end{aligned}
$$

So, the answer is $a = 48$.

14 Problem C-1

a) For every ball we have a choice of putting it in one of b bins.

$$\text{Thus } \overbrace{b \times b \times \cdots \times b}^{n} = b^n$$

.

b) Using the hint, $b-1$ sticks and n balls can be arranged in $(n+b-1)!$ ways. Since the sticks are identical the unique number of ways are:

$$\frac{(n+b-1)!}{(b-1)!}$$

c) Since the n balls are identical, $n!$ permutations must be removed which gives:

$$\frac{(n+b-1)!}{(b-1)!}\frac{1}{n!} = \binom{n+b-1}{n}$$

.

d) Here the assumption is that the number of balls are less than the number of bins. This gives us $\binom{b}{n}$ as the number of ways to place the n balls.

e) Since the bins are not to be empty and the balls are identical, the problem shifts to placing $(n-b)$ identical balls into b bins in x ways. Using part c) we get:

$$x = \binom{b+(n-b)-1}{b-1} = \binom{n-1}{b-1}$$

15 Problem 9.3-1

This solution is taken nearly verbatim from the printed solution.

- For groups of 7, the algorithm still works in linear time. The number of elements greater than x (and less than x) is at least $4(\lceil\frac{1}{2}\lceil\frac{n}{7}\rceil\rceil-2) \geq \frac{2n}{7}-8$, and the recurrence becomes

  $$T(n) \leq T(\lceil n/7\rceil) + T(5n/7+8) + O(n)$$

  which can be shown to be $O(n)$ by substitution.

- For groups of 3, however, the algorithm no longer works in linear time. The recurrence becomes

  $$T(n) \leq T(\lceil n/3\rceil) + T(2n/3+4) + O(n)$$

  which does not have a linear solution. This can be proved by showing that the worst-case time for groups of 3 is $\Omega(n \lg n)$, which can be done by deriving a recurrence for a particular case that takes $\Omega(n \lg n)$ time.

  In counting up the number of elements greater than x, consider the particular case in which there are exactly $\lceil\frac{1}{2}\lceil\frac{n}{3}\rceil\rceil$ groups with medians $\geq x$ and in which the "leftover" group contributes 2 elements greater than x. Then the number of elements greater than x is excactly $2(\lceil\frac{1}{2}\lceil\frac{n}{3}\rceil\rceil - 1) + 1$ (the -1 discounts x's group and the $+1$ is contributed by x's group) $= 2\lceil n/6\rceil - 1$, and the recursive step for elements $\leq x$ has $n - (2\lceil n/6\rceil - 1) \geq \frac{2n}{3} - 1$ elements. This give the recurrence

  $$T(n) \geq T(\lceil n/3\rceil) + T(2n/3-1) + \Theta(n) \geq T(n/3) + T(2n/3-1) + \Theta(n)$$

  from which you can show that $T(n) \geq cn \lg n$ by substitution. You can also see that $T(n)$ is non-linear by noticing that each level of the recursion tree sums to n.

- (In fact, any odd group size ≥ 5 work in linear time.)

16 Problem 11.4-1

Table 1 gives the results of each method. Note: In this case, because of the order of the inserts, Brent's rehash ends up with the same tables as double hashing. If you insert key 31 after key 15, you will notice that the hash table remains unchanged for double hashing but changes quite a bit for Brent's rehash.

17 Problem 14.2-2

Yes. Since the black-height of a node is the number of black nodes in one of its subtrees (left of right–they have the same number of black nodes), this problem is very similar to the one given in the text on maintaining as a field the size of a subtree rooted at a node (page 282–285). The black-heights can be maintained as fields for both insertion and deletion without affecting the asymptotic running times of either operation because the updates will only be local.

	linear probing		quadratic probing		double hashing		Brent's rehash
0	22	0	22	0	22	0	22
1	88	1	1	1	1	1	1
2	1	2	88	2	59	2	59
3	1	3	17	3	17	3	17
4	4	4	4	4	4	4	4
5	15	5	1	5	15	5	15
6	28	6	28	6	28	6	28
7	17	7	59	7	88	7	88
8	59	8	15	8	1	8	1
9	31	9	31	9	31	9	31
10	10	10	10	10	10	10	10

Table 1. Solutions to Problem 11.4-1

18 Problem 14.2-3

No. When the depth of a node is maintained as a field, a rotation requires changing the depth information of all the nodes in the subtrees involved. For example, a rotation at the root will require the depth field of all n nodes in the RB-tree to be updated which cannot be done without affecting the asymptotic ($O(\log n)$) performance of the RB-tree operations.

19 Binomial Queue Question

Why does insertion into a binomial queue take $O(1)$ amortized time?

It can be observed that half of the insertions (every other insertion) require no merging at all, and for the other half of the inserts there are a very limited number of operations. Actually, this problem is similar to that of incrementing a binary counter as given in the text book (pages 358–360) and the sequence of operations and the subsequent analysis is the same as the for the binary counter. Thus, insertion into a binomial queue takes $O(1)$ amortized time.

20 Binomial Queue Merge

Figure 9 shows the steps involved in merging the two binomial queues.

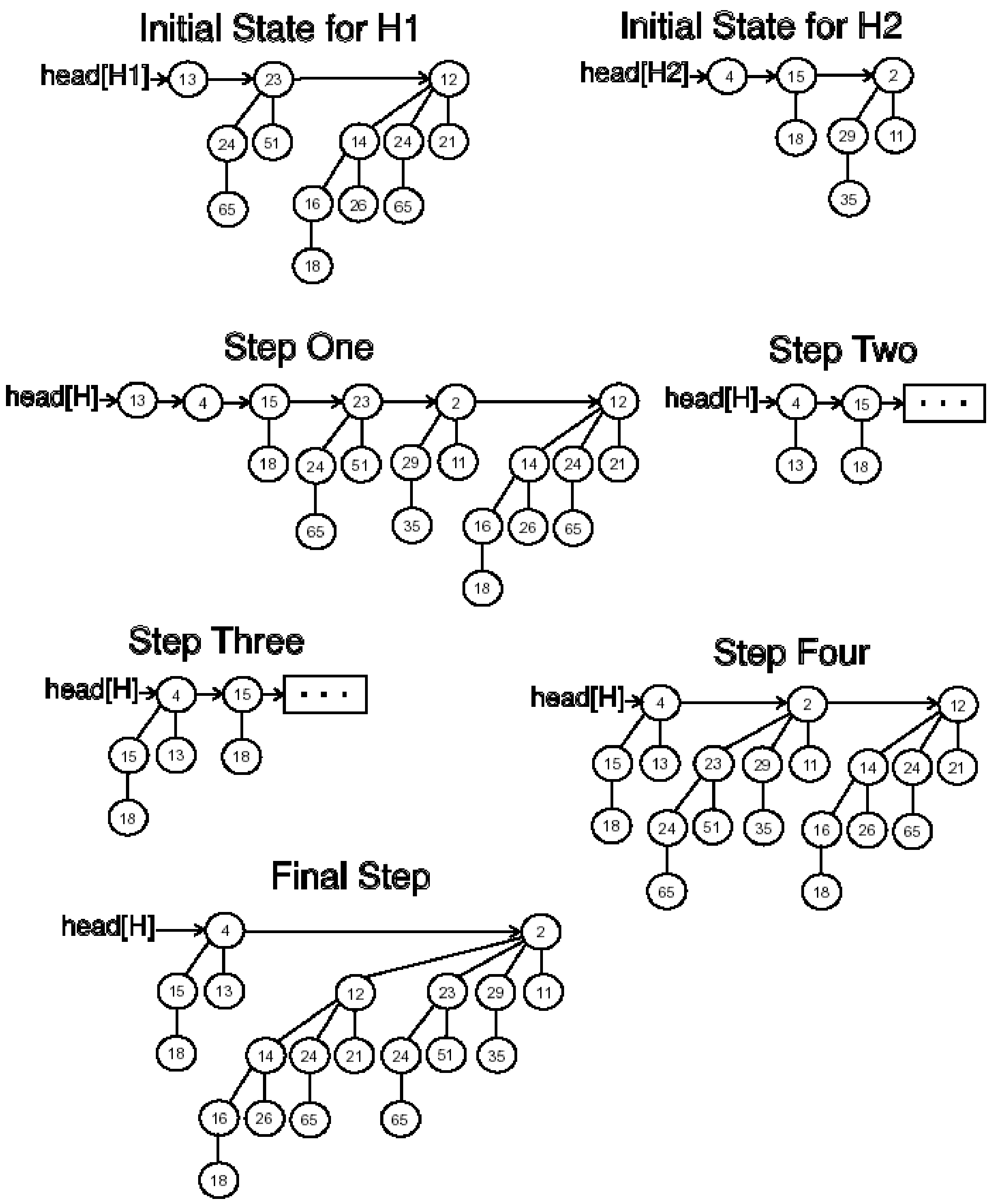

Figure 9: Merge of Binomial Queues

21 AVL Tree Deletion

Figure 10 shows the results of the special deletion. Empty boxes are used to denote unchanged subtrees.

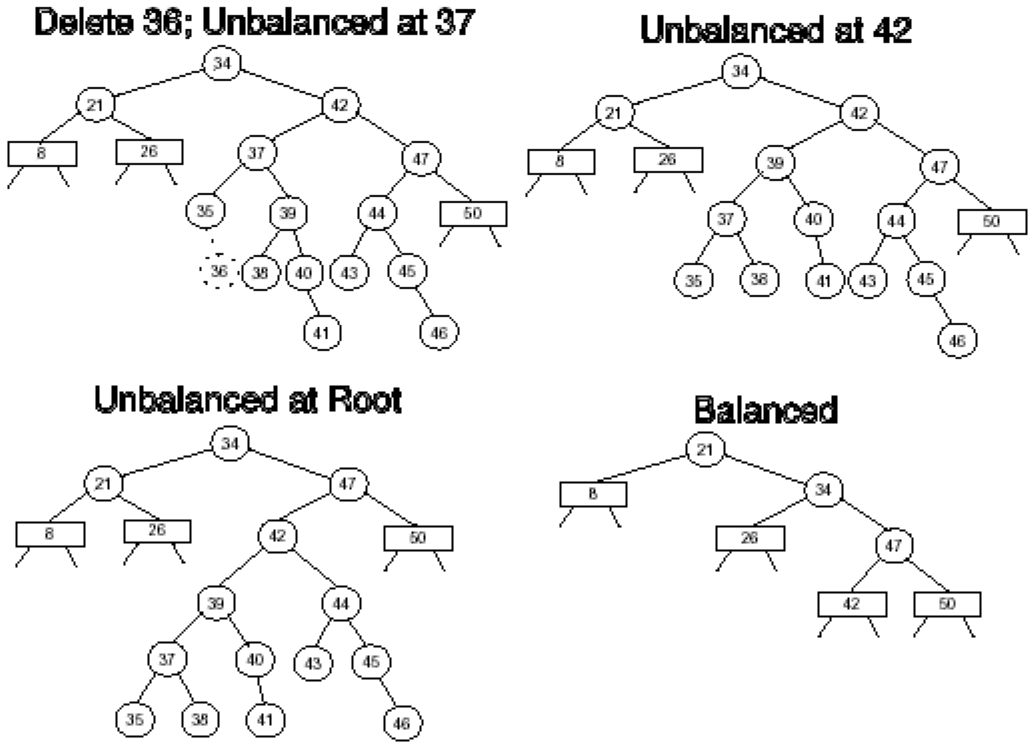

Figure 10: Deletion of 36 from AVL Tree

22 Optimal BST Construction

The answers are as follows:

- building $c(0, 5)$ using roots 2 thru 3
- Average probe length is 2.1500
- $C(0\,5)$ cost 2.1500 20(10,30(,50(40,)))
- Optimal binary search tree is given in Figure 12.

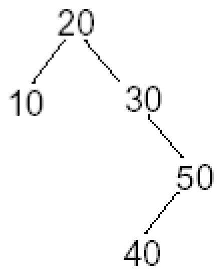

Figure 11: Optimal BST

- cost is 2.15

23 Amortized Complexity Analyses

Potential method: Assuming a sequence of operations, the potential function is $C_i\hat{} = C_i + \Phi(D_i) - \Phi(D_{i-1})$, where C_i = actual cost of ith operation, $C_i\hat{}$ = amortized cost, D_{i-1} = D.S. state before ith operation, D_i = D.S. state after ith operation.

r(x) is defined as the rank of node x, by setting the rank of the root to 0 and applying

$$\left.\begin{array}{l} r(x\hat{}.leftchild) = r(x) - 1 \\ r(x\hat{}.rightchild) = r(x) - 1 \end{array}\right\} \text{(postorder traversal of the tree method)}$$

and

$$\left.\begin{array}{l} r(x\hat{}.leftchild) = r(x) + 1 \\ r(x\hat{}.rightchild) = r(x) + 1 \end{array}\right\} \text{(preorder traversal of the tree method)}$$

Tracing the postorder traversal of the tree, and calculating the amortized complexity of each operation gives Table 2. Table 3 summaries the results for preorder traversal case.

I	Operation	D	$\Phi(D)$	C_i	$C_i\hat{} = C_i + \Phi(D_i) - \Phi(D_{i-1})$

		x=nil	0		
1	x:=PostorderInit(T)	x=I	-4	4	0
2	x:=PostorderSucc(x)	x=F	-4	2	2
3	x:=PostorderSucc(x)	x=E	-3	1	2
4	x:=PostorderSucc(x)	x=D	-2	1	2
5	x:=PostorderSucc(x)	x=H	-1	1	2
6	x:=PostorderSucc(x)	x=A	-2	3	2
7	x:=PostorderSucc(x)	x=C	-2	2	2
8	x:=PostorderSucc(x)	x=B	-1	1	2
9	x:=PostorderSucc(x)	x=G	0	1	2
10	x:=PostorderSucc(x)	x=nil	0	0	0

Table 2: Postorder Traversal Case

I	Operation	D	$\Phi(D)$	C_i	C_i^ = $C_i + \Phi(D_i) - \Phi(D_{i-1})$
		x=nil	0		
1	x:=PreorderInit(T)	x=G	0	0	0
2	x:=PreorderSucc(x)	x=H	1	1	2
3	x:=PreorderSucc(x)	x=D	2	1	2
4	x:=PreorderSucc(x)	x=E	3	1	2
5	x:=PreorderSucc(x)	x=I	4	1	2
6	x:=PreorderSucc(x)	x=F	4	2	2
7	x:=PreorderSucc(x)	x=B	1	5	2
8	x:=PreorderSucc(x)	x=A	2	1	2
9	x:=PreorderSucc(x)	x=C	2	2	2
10	x:=PreorderSucc(x)	x=nil	0	2	0

Table 3: Preorder Traversal Case

24 MST Using Warshall's algorithm

The results are as follows:

1	2	0.14
1	4	0.21
3	4	0.12
4	5	0.11
5	6	0.25
6	7	0.21
6	9	0.23
8	9	0.19

25 23.2-4

When the edge weights are integers in the range 1 to $|V|$, then line 4 (p. 505) of Kruskal's algorithm can be done using the counting sort in $O(E)$ time but the disjoint-set forest operations still take $O(E \lg E)$ time. So the total running time remains $O(E \lg E)$.

When the edge weights are integers in the range 1 to W for some constant W, it doesn't change anything. Running time remains $O(E \lg E)$.

26 23.2-5

If the edge weights are integers in the range 1 to W and W is a very small constant (say 3, i.e., all edge weights are either 1, 2 or 3), then the priority queue can be maintained as a doubly-linked list with the operation Extract-Min taking $O(1)$ time and then the running time or Prim's algorithm becomes $O(|V| + |E|)$.

If the edge weights are integers in the range 1 to $|V|$ it doesn't really change anything except when $|V|$ is very small the the W chosen above.