HW 1~~7~~ 9.4 Discrete Mathematics

P8) The body of the last for loop executes $n-1$ times the first time, $n-2$ the second time, and so on. This time dominates, so the worst-case time is

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=\Theta(n^2).$$

P12) Suppose that the weight of each edge in K_n is equal 2. Suppose that some algorithm does not examine edge e. Let T denote the minimum spanning tree output by algorithm. If e is in T, alter the input by changing the weight of e to 3. If e is not in T, alter the input by changing the weight of e to 1. Return the algorith Notice that since the algorithm does not examine e, it will still output T. However, for the modified input, T is not a minimal spanning tree. This is a contradiction. Therefore every minimal spanning tree algorithm examine every edge in K_n

---

P18) In Algorithm 9.4.3, change ∞ in line 6 to $-\infty$ and change $<$ to $>$ in line 1

P24) The algorithm picks one 10-cent and six 1-cent stamps to make 16 cents postage, but two 8-cent stamps is optimal.

P26) $a_1=11$, $a_2=5$. For $n=15$, the greedy method gives 11, 1, 1, 1, 1, but 5, 5, 5 is better

---

P32) There might be a non-greedy solution for n that does not use a 6-cent stamp. In this case, the greedy algorithm might be optimal for $n-6$, but not for n (Consider $n=10$).

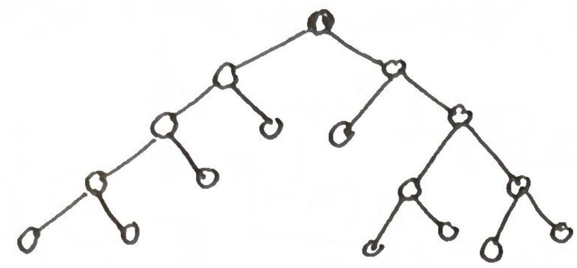
D.M.

HW 18 | 9.5 | Discrete Mathematics | page

P2) 2^{64}, which has $\lfloor 1 + \log_{10} 2^{64} \rfloor = 20$ digits

---

P10)



P14) Input: A word w to insert in a binary search tree T

output: The updated binary search tree T

```
bst_recurs (w, T)
  if (T == null) {
    let T be the tree with one vertex, root
    store w in root
    return T
  }
  s = word in T's root
  if (w < s)
    if (T has no left child)
      give T a left child and store w in it
    else {
      left = left child of T
      bst_recurs (w, left)
    }
  else
    if (T has no right child)
      give T a right child and store w in it
    else {
      right = right child of T
      bst_recurs (w, right)
    }
  return T
```

9.5

p.16) Input: The root $\underline{root}$ of a nonempty binary tree in which data are stored

output: true, if the binary tree is a binary search tree;

false, " " " " is not " " "

If the binary tree is a binary search tree, the algorith sets $\underline{small}$ to the smallest value in the tree and $\underline{large}$ to the largest value in the tree.

```
is_bst(root, small, large) {
  if (root has no children) {
    small = value of root
    large = " " "
    return true
  }
  lchild = left child of root
  rchild = right " " "
  if (is_bsT(l child, small_left, large_left) ∧ is_bsT(
  rchild, small_right, larg_right)) {
     val = value of root
   if (large_left > val ∨ small_right < val)
    return false
    small = small_left
    large = larg_right
    return true
  }
  else return false
}
```

~~p.19~~ ~~Not balanced~~

4

HW18 $\boxed{95}$

P 26) we prove that $n < 2^{h+1}$ using induction on n.

Basic step $n = 1$ true

Assume that the result is true for binary trees with less than n vertices.

Let T be an n-vertex binary tree.

Let n_L be the number of vertices in T's left subtree, and let n_R be ~~the height of T's right subtree~~ the number of vertices in T's right subtree. Let h_L be the height of T's left subtree, and let h_R be the height of T's right subtree. Note that $1 + h_L \le h$ and $1 + h_R \le h$.

By the induction assumption, $n_L < 2^{h_L + 1}$ and $n_R < 2^{h_R + 1}$. Now

$$n = 1 + n_L + n_R < 1 + 2^{h_L + 1} + 2^{h_R + 1} \le 1 + 2^h + 2^h$$

$$= 1 + 2 \cdot 2^h = 1 + 2^{h+1}$$

(P.19) Not balanced