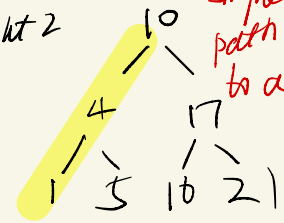
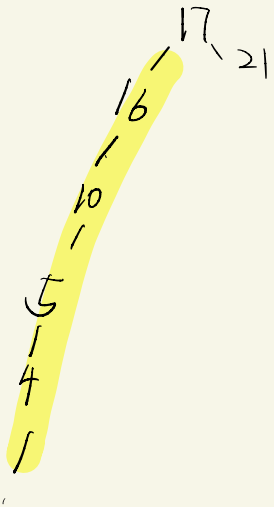
12-1-1 Height of the node:
The number of edges on the longest simple downward path from the node to a leaf.

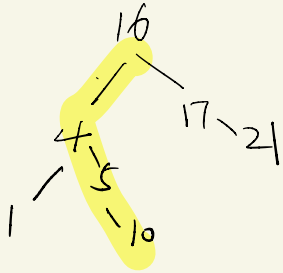
Height 2



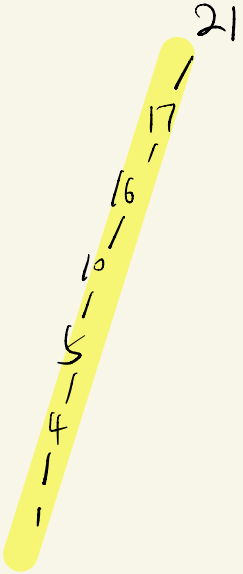
Height 5



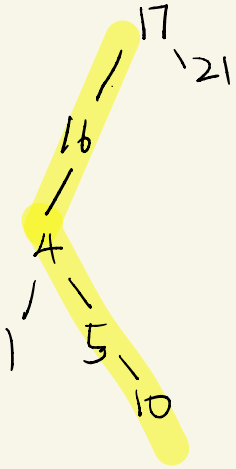
Height 3



Height 6



Height 4



12.1-2

minheap: node $\leq$ left-subtree children
node $\leq$ right-subtree children

Binary-Search-Tree:
node $\geq$ left-subtree children
node $\leq$ right-subtree children

INORDER-WALK takes $\Theta(n)$ and print out n-node in sorted order.
(loop invariant: the printed subtree is sorted)

HEAPSORT takes $O(n \lg n)$

Page 160

12.1-3

Easy non-recursive walk with stack:

```
INORDER-TREE-WALK(x)
  S = new STACK()

  while x ≠ NIL or not STACK-EMPTY(S)
      while x equals NIL
          S.PUSH(x)
          x = x.left

      x = S.pop()
      print x.key
      x = x.right
```

12.1-4

```
PREORDER-TREE-WALK(x)
  if x ≠ NIL
    print x.key
    PREORDER-TREE-WALK(x.left)
    PREORDER-TREE-WALK(x.right)

POSTOREDER-TREE-WALK(x)
  if x ≠ NIL
    POSTOREDER-TREE-WALK(x.left)
    POSTOREDER-TREE-WALK(x.right)
    print x.key.
```

12.1-5

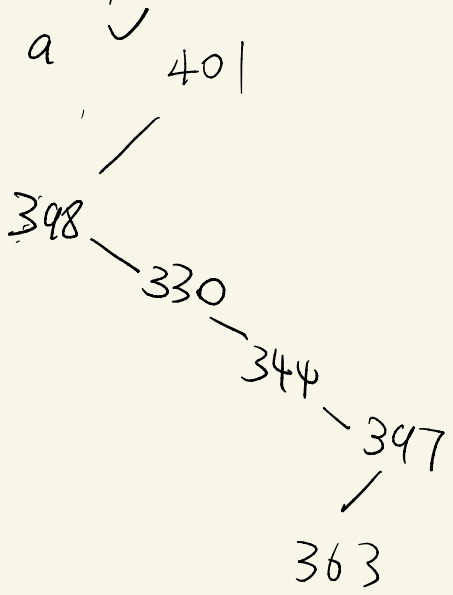
Assume that for an arbitary list of n elements takes $O(n \lg n)$ time in worst case to construct a binary search tree.

Then it takes $O(n)$ to get a sorted arrary from this binary search tree.

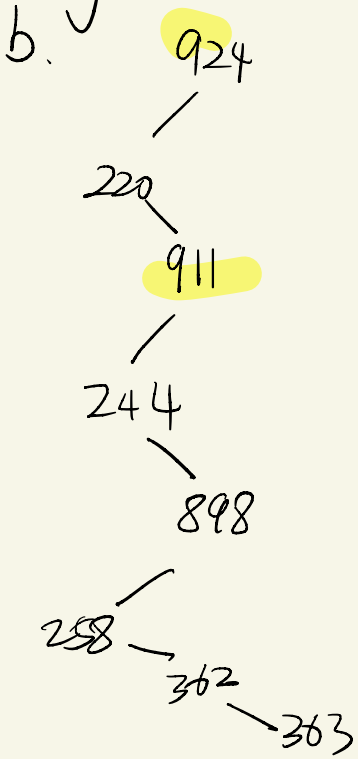
In this way, it tapes $O(n \lg n + n) = O(n \lg n)$ time to soot an arrary in worst case by a comparison model. This contradicts the lower bound for sorting with comparison model, which is $\Omega(n \lg n)$.

12.2-1

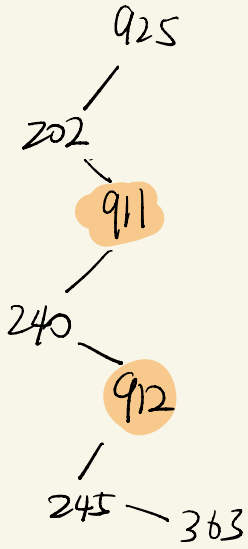
a. ✓


401
398
330
344
347
363

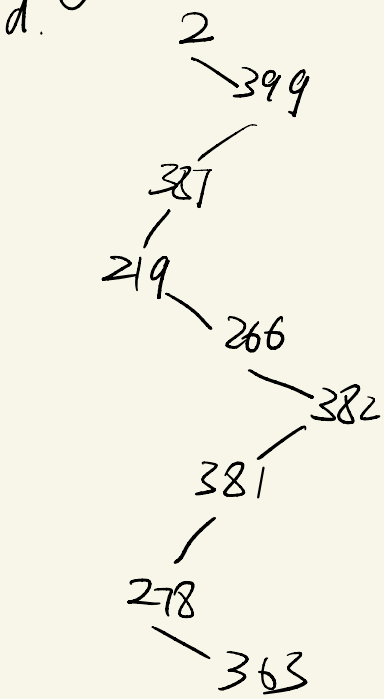
b. ✓


924
220
911
244
898
258
362
363

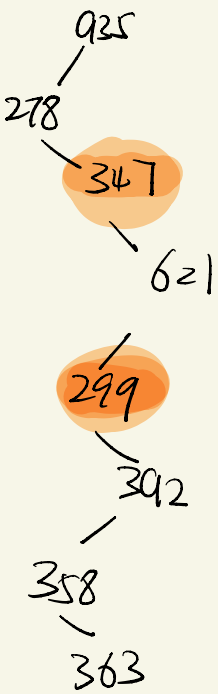
c.


925
202
911
240
912
245
363

d. ✓


2
399
387
219
266
382
381
278
363

e. ✓

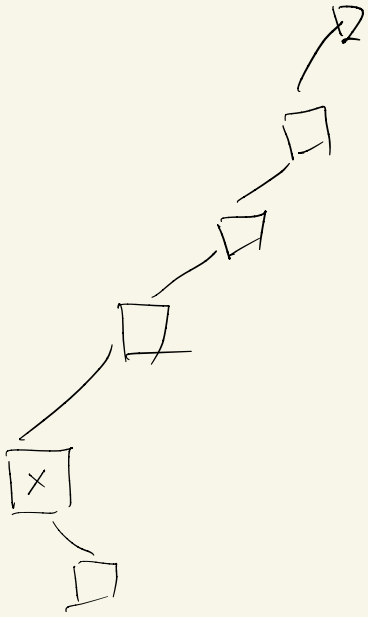

925
278
347
621
299
392
358
363

12.2-2

```
TREE-MINIMUM(x)
    if x.left ≠ NIL
        return TREE-MINIMUM(x.left)

    else
        return x
```

```
TREE-MAXIMUM(x)
    if x.right ≠ NIL
        return TREE-MAXIMUM(x.right)
    else
        return x
```

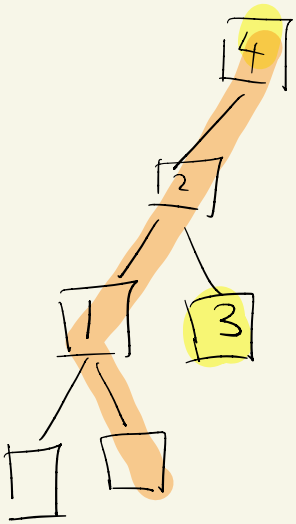


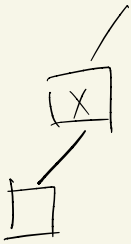
12.2-3

```
TREE-PREDECESSOR(x)
 if x.left ≠ NIL
    return TREE-MAXIMUM(x.left)
 y = x.p
 while y ≠ NIL and x == y.left
        x = y
        y = x.p

return y
```

12.2-4





$y \geq x$

when the ancestor of x whose right child is an ancestor of x, this ancestor is less than x.

So every ancestor whose left child is an ancestor of x is greater than x.

The higher such ancestors get, the greater they get.

So. — — — .

12.2-5

successor is the minimum of the right subtree, which is a leaf.

predecessor. —

12.2-6

The successor is the element whose key the minimum element greater than x. key.

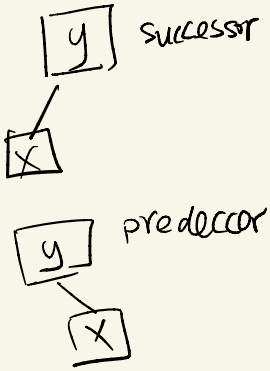
12.2-7

No redundant work is done.

12.2-8

$O(k+h)$?

12.2-9



12.3-1

```
TREE-INSERT(parent, z)
if parent == NIL
    z.p = NIL
    parent = z
```

EMPTY TREE

```
elif parent.key > z.key
    if parent.left == NIL
        z.p = parent
        parent.left = z
    else
        TREE-INSERT(parent.left, z)

else
    if parent.right == NIL
        z.p = parent
        parent.right = z
    else
        TREE-INSERT(parent.right, z)
```

12.3-2

SEARCH and TREE-INSERT takes the same steps except that SEARCH will examine one node more since the value is inserted.

12.3-3 worst case running times:

INSERT takes $O(h)$

$O(1+2+\cdots+n+n)$ → INORDER-WALK

$= O(n^2)$

Best case running times

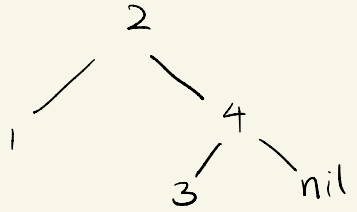
$O(\lg 1 + \lg 2 + \cdots \lg n + n)$

$= O(\lg n! + n)$

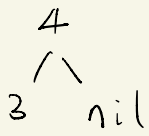
$= O(n \lg n)$

12.3-4

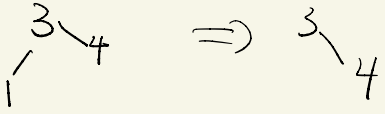
Not Communiratie



1 => 2



2 => 1



12.3-5

The left child and right child pointers remain the same.

The search algorithms remain the same.

```
TREE-SEARCH(x, k)
    if x == nil
        return

    elif x.key == k
        return x

    elif x.key > k
        return TREE-SEARCH(x.left, k)

    else
        return TREE-SEARCH(x.right, k)
```

```
TREE-INSERT(T, z)
  y = NIL
  x = T.root
  pred = NIL
  while x ≠ NIL
      y = x
      if x.key > z.key
          x = x.left
      else
          pred = x
          x = x.right
  if y == NIL
      T.root = z
      z.successor = NIL
  else if y.key > z.key
      y.left = z
      z.successor = y
      if pred ≠ NIL
          pred.successor = z
  else
      y.right = z
      z.successor = y.successor
      z.pred = y
```

pred is not the predecessor of the current node but the predecessor of the last node, which is a leaf.

when pred remains NIL, we keep going to the left. Then z is the minimum of the tree, the pred of z is NIL.

Case #0: Empty tree

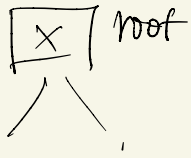
Case #1: z is the left child

Case #1.1 1.2
when pred = NIL
z is the minimum node
Otherwise, maintain the successor single linked list.

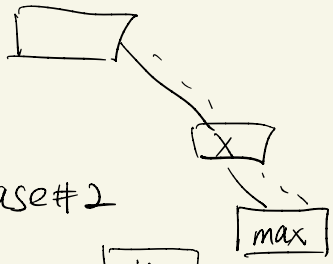
Case #2
z is the right child.

```
PARENT(T, x)
if x == T.root
   return NIL
else
   y = TREE-MAXIUM(x).successor
   if y == NIL
      y = T.root
   else if y.left == x
      return y
   else
      y = y.left
   while y.right ≠ x
      y = y.right
   return y
```

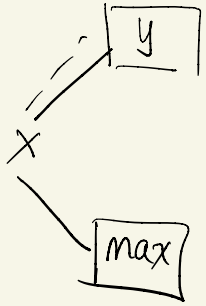
Case #0



Case #1

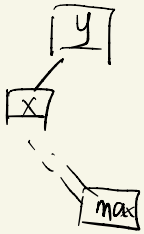


Case #2

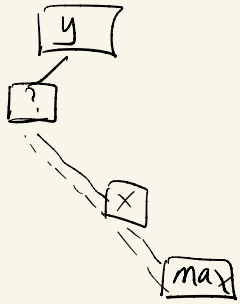


Can turn left once otherwise y will not be the successor of max

Case #2.1



Case #2.2



```
TRANSPLANT (T, U, V)
    parentU = PARENT(T, U)
    if parentU == NIL
        T.root = V
    else if parentU.left == U
            parentU.left = V
    else    parentU.right == U
            parentU.right == V
```

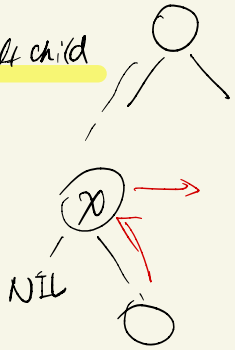
```
TREE-PREDECESSOR(T, x)
if x.left ≠ NIL
    return TREE.MAXIUM(x.left)
else
    y = T.root
    pred = NIL
    while y != x and y != NIL
        if y.key > x
            y = y.left
        else if y.key < x
            pred = y
            y = y.right
    if y == NIL
        ERROR("No key is found")
    else return pred.
```

pred can be NIL

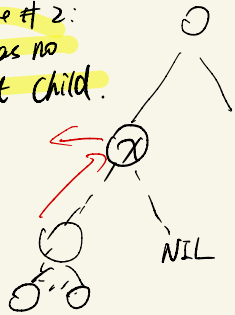
```
DELETE(T, x)
if x.left == NIL
    TRANSPLANT(T, x, x.right)
else if x.right == NIL
        TRANSPLANT(T, x, x.right)
else
    if PARENT(x.successor) != x
        TRANSPLANT(T, x.successor, x.successor.right)
        x.successor.right = x.right
    TRANSPLANT(T, x, x.successor)
    x.successor.left = x.left
pred = TREE-PREDECESSOR(x)
if pred != NIL
    pred.successor = x.successor
x.successor = NIL.
```

// Maintain the successor property

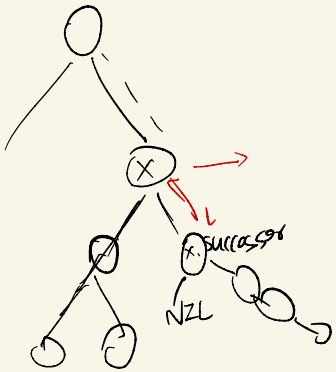
Case #1:
x has no left child



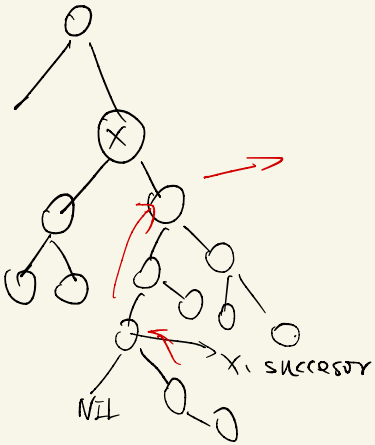
Case # 2:
x has no right child.



case # 3



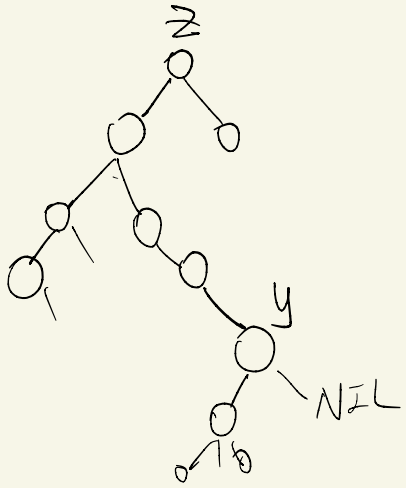
case # 4

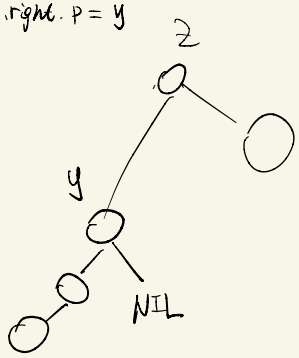


12.3-6

```
TREE-DELETE(T, z)
  if z.left == NIL
     TRANSPLANT(T, z, z.right)
  else if z.right == NIL
     TRANSPLANT(T, z, z.left)
  else
     y = TREE-MAXIMUM(z.left)
     if y.p ≠ z
        TRANSPLANT(T, y, y.left)
        y.left = z.left
        y.left.p = y
     TRANSPLANT(T, y, z)
     y.right = z.right
     y.right.p = y
```

Using a random number generator to switch between 2 algorithm





12.4.1

$$\sum_{i=0}^{n-1} \binom{i+3}{3} = \binom{n+3}{4}$$

Base case $n=1$

$$\text{LHS} = \sum_{i=0}^{0} \binom{i+3}{3} = \binom{3}{3} = 1$$

$$\text{RHS} = \binom{4}{4} = 1$$

LHS = RHS

Inductive case $n>1$

$n=k$

$$\sum_{i=0}^{k-1} \binom{i+3}{3} = \binom{k+3}{4}$$

$n=k+1$

$$\text{LHS} = \sum_{i=0}^{k} \binom{i+3}{3} = \sum_{i=0}^{k-1} \binom{i+3}{3} + \binom{k+3}{3}$$

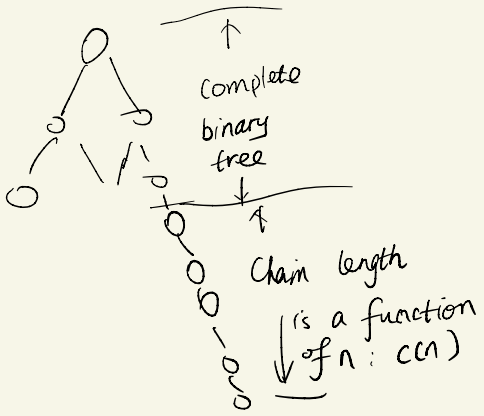
$$= \binom{k+3}{4} + \binom{k+3}{3}$$

$$= \frac{(k+3)(k+2)(k+1)\cdot k}{4!} + \frac{(k+3)(k+2)(k+1)}{3!}$$

$$= \frac{(k+3)(k+2)(k+1)}{3!}\left(\frac{k+4}{4}\right) = \frac{(k+4)(k+3)\cdots(k+1)}{4!}$$

$$= \frac{(k+4)}{4!} \quad \text{q.e.d.}$$

12.4-2



$h = \lg(n - c(n)) + c(n)$

when $c(n) = \omega(\lg n)$

$h = \omega(\lg n)$

Average Depth of each node: (the length of)

Depth of the complete binary tree

Depth of the chain

$$\bar{D} = \frac{1}{n}\left[\sum_{i=1}^{n-c(n)} \lg i + \left(\lg(n-c(n)) + 1\right) + \cdots + \left[\lg(n-c(n)) + c(n)\right]\right)$$

$$= \frac{1}{n}\left[(n-c(n))\lg(n-c(n)) + c(n)\cdot\lg(n-c(n)) + \frac{(c(n)-1)c(n)}{2}\right)$$

$$= \frac{1}{n}\left[n\cdot\lg(n-c(n)) + \frac{c^2(n)-c(n)}{2}\right)$$

$$= \lg(n-c(n)) + \frac{c^2(n)-c(n)}{2n}$$

when $c(n) = \sqrt{n}$

$$\bar{D} = \Theta\left(\lg(n-c(n)) + \frac{c^2(n)-c(n)}{2n}\right)$$

$$= \Theta\left(\lg(n-\sqrt{n}) + \frac{1}{2\sqrt{n}}\right) = \Theta(\lg(n))$$

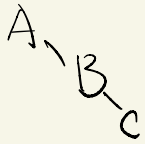
12.4-3 $A < B < C$

$n=3$

permutation

Binary Trees

A B C



A C B

B C A

B A C

C A B

C B A

Six permutations 5 binary Tree.

12.4-4

$$y = 2^x = (e^{\ln 2})^x = e^{\ln 2 x}$$

$$y' = 2^x \cdot \ln 2$$

$$y'' = 2^x (\ln 2)^2 > 0.$$

So $y = 2^x$ is convex.

12.4-5

12-1

```
TREE-INSERT(T, z)
y=NIL
x=T.root
while x != NIL
    y=x
    if x.key > z.key
        x=x.left
    else if x.key < z.key
        x=x.right
    else
        x=STRATEGY(x)
z.p=y
if y==NIL
    T.root=x
else
    if y.key > x
        y.left=x
    else if y.key < x
        y.right=x
    else
        y=STRAGGY(y)
        y=z
```

```
STRATEGY1(x)
if x.b
    x=x.left
    x.b=False
else
    x=x.right
    x.b=True
return x
```

a.

Insert n items

requires $\Theta(n^2)$

$\Theta(1+2+3+\cdots(n-))$

b. $\Theta(\lg 1+\lg 2+\cdots\lg(n-1))$

$=\Theta(n\lg n)$

c. $O(n)$

d. worst case

$\Theta(n^2)$

Expected.

$\Theta(n\lg n)$

12-2

```
PREORDER-WALK(x)
PRINT-NONE-NIL-KEY(x)

PRINT-NONE-NIL-KEY(x)
if x.key ≠ NIL
    print x.key

if x.left ≠ NIL
    PRINT-NONE-NIL-KEY(x)
    PREORDER-WALK(x.left)
else if x.right ≠ NIL
    PRINT-NONE-NIL-KEY(x)
    PREORDER-WALK(x.right)
```

$x < x.left < x.right$

lexicographically.

$O(n)$

n is the total sum of bit of strings.

12-3

a. By the definition of $P(T) = \sum_{x \in T} d(x, T)$

b.

$$P(T) = P(T_L) + P(T_R) + n-1$$

Since we have $n-1$ nodes in the T_L and T_R who need one level depth for compensation.

i is the number of nodes is the left subtree.

c.

$$RHS = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n-i-1) + n-1)$$

$$= \frac{1}{n} \sum_{i=0}^{n-1} (P(T_L) + P(T_R) + n-1)$$

$$= \frac{1}{n} \sum_{i=0}^{n-1} P(T)$$

$$= P(n)$$

?

permutation is different from the tree.

d. $P(0) = 0$

$$P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n-i-1) + n-1)$$

$$= \frac{1}{n} \sum_{i=0}^{n-1} P(i) + \frac{1}{n} \sum_{i=0}^{n-1} P(n-i-1) + \frac{1}{n} \sum_{i=0}^{n-1} (n-1)$$

$$= \frac{2}{n} \sum_{k=1}^{n-1} P(k) + \Theta(n)$$

7. b

d. upper bound

$\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$

Assume for $n \ge n_0$:

$p(n) \le a n \lg n$

$p(n) = \frac{2}{n} \sum_{k=1}^{n-1} p(k) + \Theta(n)$

$\le \frac{2}{n} \sum_{k=1}^{n-1} a k \lg k + \Theta(n)$

$\le \frac{2}{n} \cdot a \cdot \left(\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \underbrace{\frac{2}{n} \cdot a \cdot 1 \cdot \lg 1}_{0} + \Theta(n)$

$= a n \lg n - \frac{a}{4} n + \Theta(n)$

For sufficient large a

$a n \lg n - \frac{a}{4} n + \Theta(n) \le a n \lg n$. q.e.d

f.

PARTITION(A, P, r)

pivot = A[P]

i = p

for j = p+1 to r

if A[j] ≤ pivot

x = A[j]

y = A[i]

for k = i+1 to j

temp = A[k]

A[k] = y

y = temp

A[i] = x

i = i+1

$O(n^2)$ breaks Quicksort

pivots have to be the roots

Maintain the order of left part and right part as it was before partition

PARTITION(A, p, r)

let left and right be 2 new arrays.

pivot = A[1]

for i = p to r

if A[i] ≤ pivot

left.append(A[i])

else

right.append(A[i])

$O(r-p)$ in time and space.

for i = p to r

if i-p+1 ≤ left.length

A[i] = left[i-p+1]

else

A[i] = right[i-p+1-left.length]

12.4

a. $n=0$

There is no nodes

$b_n = 1$, the empty binary tree.

b_n = num of left tree · number of right tree

$$= \sum_{k=0}^{n-1} b_k \, b_{n-1-k}$$

$b_1 = b_0^2$

$b_2 = b_1 b_0 + b_0 b_1$

$b_3 = b_2 b_0 + b_1^2 + b_0 b_2$

$b_4 = b_3 b_0 + b_2 b_1 + b_1 b_2 + b_0 b_3$

$\vdots$

$b_m = b_0 b_{m-1} + \cdots + b_{m-1} b_0$

$b_{m+1} = b_0 b_m + b_1 b_{m-1} + \cdots + b_m b_0$

$b_{m+2} = b_0 b_{m+1} + b_1 b_m + \cdots + b_{m+1} b_0$

b $RHS = x \cdot B(x)^2 + 1$

$LHS = \sum_{n=0}^{\infty} b_n x^n$

$$RHS = x \left[\sum_{n=0}^{\infty} b_n x^n \right]^2 + \overbrace{b_0 x^0}^{1 \cdot 1}$$

$$= x \sum_{n=1}^{\infty} \sum_{k=0}^{n-1} b_k \, b_{n-1-k} \, x^{n-1} + b_0 x^0$$

$$= \sum_{n=1}^{\infty} \sum_{k=0}^{n-1} b_k \, b_{n+k} \, x^n + b_0 x^0$$

$$= \sum_{n=1}^{\infty} b_n x^n + b_0 x^0$$

$$= \sum_{n=0}^{\infty} b_0 x^n = LHS.$$

C. $f(x)=\sqrt{1-4x}$

$f^{(1)}(x)=[(1-4x)^{\frac{1}{2}}]'$

$=\frac{1}{2}(1-4x)^{-\frac{1}{2}}(-4)$

$f^{(2)}(x)=(\frac{1}{2})(-\frac{1}{2})(1-4x)^{-\frac{3}{2}}(-4)^2$

$f^{(k)}(x)=-(1\cdot 3\cdot 5\cdot \cdots\cdot(2k-3))\,2^k\,(1-4x)^{-\frac{2k-1}{2}}$

$f^{(k)}(0)=-(1\cdot 3\cdot 5\cdots(2k-3))\cdot 2^k$

$$f(x)=\sum_{k=0}^{\infty}\frac{f^{(k)}(0)}{k!}(x-a)^k$$

$$=\sum_{k=0}^{\infty}\frac{(2(k-1))!\,2^k}{2^{k-1}(k-1)!k!}x^k$$

$$B(x)=\frac{1}{2x}\left(\sum_{k=1}^{\infty}\frac{(2(k-1))!}{2^{k-1}(k-1)!k!}\cdot 2^k\,x^k\right)$$

$$=\sum_{k=1}^{\infty}\frac{(2(k-1))!}{2(k-1)!\,k!}x^{k-1}$$

$b_n=\frac{1}{n+1}\binom{2n}{n}$

$$=\sum_{n=0}^{\infty}\frac{(2n)!}{n!(n+1)!}\cdot x^n=\sum_{n=0}^{\infty}\frac{1}{n+1}\cdot\frac{2n!}{n!n!}x^n$$

d. $b_n = \frac{1}{n+1}\frac{(2n)!}{(n!)(n!)}$ Stirling approximation

$= \frac{1}{n+1} \cdot \frac{\sqrt{2\pi 2n}\left(\frac{2n}{e}\right)^{2n}\left(1+O\left(\frac{1}{2n}\right)\right)}{2\pi n\left(\frac{n}{e}\right)^{2n}\left(1+O\left(\frac{1}{n}\right)\right)^2}$ page 57

→ when n is sufficiently large

$\approx \frac{1}{n+1} 4^n \cdot \frac{1}{\sqrt{2\pi n}}$

$= \frac{4^n}{\sqrt{2\pi}\, n^{\frac{3}{2}}}\left[\frac{n}{n+1}\right]$

$= \frac{4^n}{\sqrt{2\pi}\, n^{\frac{3}{2}}}\left[1-\frac{1}{n+1}\right] = \frac{4^n}{\sqrt{2\pi}\, n^{\frac{3}{2}}}\left(1+O\left(\frac{1}{n}\right)\right)$

q.e.d.