1. 1-1

	1 Second	1 minute	1 Hour
lg n	$2^{\wedge(10\wedge6)}$	$2^{(6 \times 10\wedge7)}$	$2^{(3.6x10\wedge9)}$
√n)	10^{12}	3.6×10^{15}	1.3×10^{19}
n	10^{6}	$6x10^{7}$	3.6×10^{9}
n lg n	6.3×10^{4}	2.8×10^{6}	1.3×10^{8}
n^2	10^{3}	7.7×10^{3}	6×10^{4}
n^3	10^{2}	$10^{(2*6)}$	$10^{(3 * 1)}$
2^n	20	26	31
n!	9	11	12

2. 2.2-3. Average number A(n) of elements to check in order to find an element in the array of length n can be obtained as:

$$A(n) = \sum_{i=1}^{n} P_i T_i$$

Where P_i is probability of the element to be in position i, and T_i is number of comparisons to make for a element in position i.

If we assume that an array contains distinct elements, and the element to be searched is equally likely to be in any position in the array, then $P_i = \frac{1}{n}$ and $T_i = i$. This yields

$$A(n) = \frac{1}{n} \sum_{i=1}^{n} i = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

Thus, on average, about half the array will be checked. Running time for this algorithm is $\frac{n+1}{2} = \Theta(n)$. In the worst case, the whole array has to be checked, i.e. n comparisons are to be made. Worst case running time is $n = \Theta(n)$.

3. 2.3-1. Merge sort of the array A = {3, 41, 52, 26, 38, 57, 9, 49}

3	41	52	26	38	57	9	49	Initial Array
3	41	26	52	38	57	9	49	Step 1
3	26	41	52	9	38	49	57	Step 2
3	9	26	38	41	49	52	57	Sorted Array

4. 3.1-4.

Is $2^{n+1} = O(2^n)$? **YES**

Is $2^{2n} = O(2^n)$? **NO** $2^{2n} = O(4^n)$

5. 3.2-2. Prove $a^{\log_b n} = n^{\log_b a}$

$\log_b a \bullet \log_b n = \log_b n \bullet \log_b a$ commutativity of $\bullet$

$\log_b a^{\log_b n} = \log_b n^{\log_b a}$ $(\log x)y = \log x^y$ (both sides)

$a^{\log_b n} = n^{\log_b a}$ $x^{\log_x y} = y$ (both sides)

6. 3-2.

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ	YES	YES	No	No	No
n^k	c^n	YES	YES	No	No	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No	No	No
2^n	$2^{n/2}$	No	No	YES	YES	No
$n^{\lg c}$	$c^{\lg n}$	YES	No	YES	No	YES
$\lg(n!)$	$\lg(n^n)$	YES	No	YES	No	YES

7. 3-4.

a.) $f(n) = O(g(n))$ implies $g(n) = O(f(n))$ **FALSE**

Let $f(n) = n$ and $g(n) = n^2$.

n^2 is an upper bound on n, but n is not an upper bound on n^2

b.) $f(n) + g(n) = \Theta(\min(f(n), g(n)))$. **FALSE**

Let $f(n) = n$, $g(n) = n^2$. So for $f(n) + g(n)$, $\min(f(n), g(n)) = n$.

Thus, $n + n^2 = \Theta(n)$ does not hold

c.) $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \geq 0$ and $f(n) \geq 1$ for all sufficiently large n. **TRUE**

$f(n) = O(g(n))$ means $\exists n_0, c_1$ such that $0 \leq f(n) \leq c_1 g(n)$ for all $n > n_0$ (1)

Taking lg gives $0 \leq \lg(f(n)) \leq \lg c_1 + \lg(g(n))$ for all $n > n_0$ (2)

For sufficiently large n in (2), $0 \leq \lg(f(n)) \leq \lg c_1 + \lg(g(n)) \leq 2\lg(g(n))$.

d.) $f(n) = O(g(n))$ implies $2^{f(n)} = O\left(2^{g(n)}\right)$ **FALSE**

Let $f(n) = 2n$ and $g(n) = n$. $2^{2n} = 4^n \notin O\left(2^n\right)$

e.) $f(n) = O\left(f(n)^2\right)$ **FALSE**

Let $f(n)$ be $\frac{1}{n}$.

f.) $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$ **TRUE**

By transpose symmetry, $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

g.) $f(n) = \Theta\left(f\left(\frac{n}{2}\right)\right)$ **FALSE**

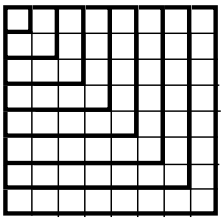
Let $f(n) = 2^n$. $2^n \notin O\left(2^{\frac{n}{2}}\right)$

h.) f(n) + o(f(n)) = Θ(f(n)) **TRUE**

For whatever $g(n) = o(f(n))$ is chosen, $\exists n_0, c_1$ such that $g(n) \le cf(n)$ when $n > n_0$.
From this and exercise 3.1-1 (in Notes 2), $\max(g(n), cf(n)) = cf(n) = \Theta(f(n))$.

8. A.1-1. Simplify the expression $\sum_{k=1}^{n}(2k-1)$

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



9. A.2-2. Find an asymptotic upper bound on $\sum_{k=0}^{\lfloor \lg n \rfloor}\left\lceil \frac{n}{2^k} \right\rceil$.

$$\sum_{k=0}^{\lfloor \lg n \rfloor}\left\lceil \frac{n}{2^k} \right\rceil \le \sum_{k=0}^{\lg n}\left(\frac{n}{2^k}+1\right) = \lg n + 1 + n\sum_{k=0}^{\lg n}\frac{1}{2^k} \le \lg n + 1 + n\frac{1}{1-\frac{1}{2}} = \lg n + 1 + 2n = \mathrm{O}(n)$$

10. A.2-3. Show that the nth harmonic number is $\Omega(\lg n)$ by splitting summations.

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \geq \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j}$$

$$\geq \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{j=0}^{2^i - 1} \frac{1}{2^{i+1}}$$

$$= \frac{1}{2} \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{j=0}^{2^i - 1} \frac{1}{2^i} = \frac{1}{2} \sum_{i=0}^{\lfloor \lg n \rfloor - 1} 1 = \frac{1}{2} \lfloor \lg n \rfloor$$

11. A.2-4. Approximate $\sum_{k=1}^{n} k^3$ with an integral.

$$\int_0^n k^3 dk \leq \sum_{k=1}^{n} k^3 \leq \int_0^{n+1} k^3 dk$$

$$\left[\frac{k^4}{4} \right]_0^n \leq \sum_{k=1}^{n} k^3 \leq \left[\frac{k^4}{4} \right]_1^{n+1}$$

$$\frac{n^4}{4} \leq \sum_{k=1}^{n} k^3 \leq \frac{(n+1)^4}{4} - \frac{1}{4}$$

12. 4.3-2. Show that the solution of $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ is $O(\lg n)$. (Assume base 2 logarithms.)

Must show that $T(n) \leq c \lg n$ for some $c > 0$.

Assume $T\left(\left\lceil \frac{n}{2} \right\rceil\right) \leq c \lg\left(\frac{n}{2}\right)$ and substitute.

$T(n) \leq c \lg\left(\frac{n}{2}\right) + 1 = c \lg n - c + 1 \leq c \lg n$ if $c \geq 1$

13. 4.3-3. Show that the solution of $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ is $\Omega(n \lg n)$ and conclude that the solution is $\Theta(n \lg n)$.

Must show that $T(n) \geq cn \lg n$ for some $c > 0$.

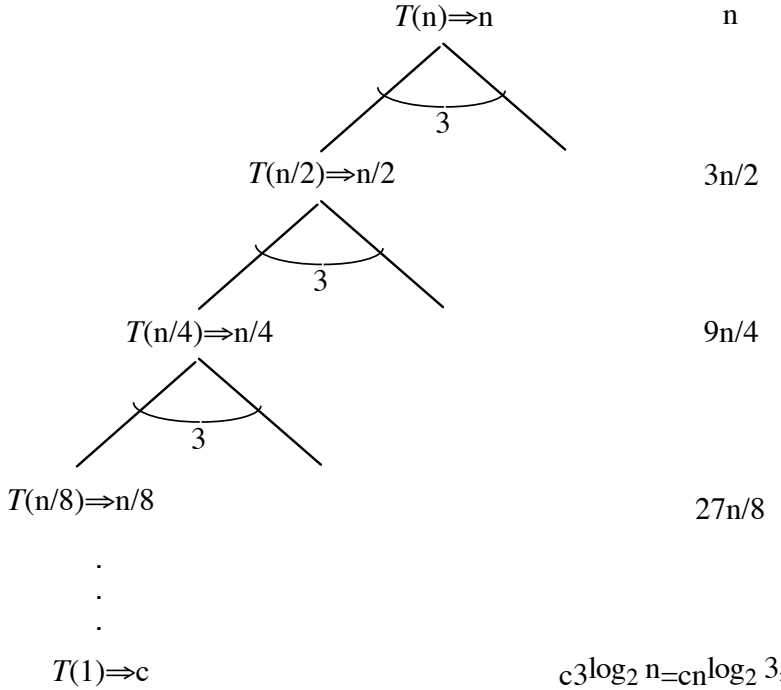
Assume that $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \geq c \frac{n}{2} \lg \frac{n}{2}$. (Base 2 logarithms)

$T(n) \geq 2c \frac{n}{2} \lg \frac{n}{2} + n$

$= cn \lg n - cn + n$

$\geq cn \lg n$ if $0 < c \leq 1$

14. 4.4-1. Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$. Use the substitution method to verify your answer.



$$cn\sum_{i=0}^{\log_2 n-1}\left(\frac{3}{2}\right)^i + cn^{\log_2 3} = cn\frac{\left(\frac{3}{2}\right)^{\log_2 n}-1}{\frac{3}{2}-1} + cn^{\log_2 3}$$

$$= 2cn\left(n^{\log_2 \frac{3}{2}}-1\right) + cn^{\log_2 3}$$

$$= 2cn\left(n^{.585}-1\right) + cn^{1.585}$$

$$= 3cn^{1.585} - 2cn$$

Substitution method: Show $T(n) = O\left(n^{\log_2 3}\right)$

Assume $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \le c\left(\frac{n}{2}\right)^{\log_2 3} - c\frac{n}{2}$. Show $T(n) \le cn^{\log_2 3} - cn$.

$$T(n) \le 3\left(c\left(\frac{n}{2}\right)^{\log_2 3} - c\frac{n}{2}\right) + n = 3c\frac{n^{\log_2 3}}{2^{\log_2 3}} - 3c\frac{n}{2} + n$$

$$= cn^{\log_2 3} - 3c\frac{n}{2} + n$$

$$= cn^{\log_2 3} - cn - \frac{1}{2}cn + n$$

$$\le cn^{\log_2 3} - cn \text{ if } c \ge 2$$

15. 4.4-6. Argue that the solution to the recurrence $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$ is $\Omega(n \lg n)$ by appealing to the recursion tree.

From Figure 4.6, the least depth of complete levels is $\log_{\frac{3}{2}} n$, and each level adds *n* to the algorithm's running time.

16. C.3-1. Expectation of the sum

Sum	**Number of ways we can get by throwing the dice**	**Value**
2	1	2
3	2	6
4	3	12
5	4	20
6	5	30
7	6	42
8	5	40
9	4	36
10	3	30
11	2	22
12	1	12
Total		252

Since the probability of all events is equal, P = 1/36 . The expectation = 253 /36 = 7

Expectation of the maximum

Maximum	**Number of ways we can get by throwing the dice**	**Value**
1	1	1
2	3	6
3	5	15
4	7	28
5	9	45
6	11	66
Total		161

Since the probability of all events is equal, P = 1/36 . The expectation = 161 /36 = 4.47

17. C.3-2

The expectation of the index of the **maximum** element in the array A is,

$$\text{Expectation} = \sum_{i=1}^{n} \frac{i}{n} = \frac{1}{n} \sum_{i=1}^{n} i = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

The probability of the maximum element in any of the n positions is $\frac{1}{n}$. Similarly E for the **minimum** element is also $\frac{n+1}{2}$.

18. C.3-3. There are four possible outcomes,

1. The person loses a dollar or
2. He gains 1 dollar or
3. He gains 2 dollars or
4. He gains 3 dollars

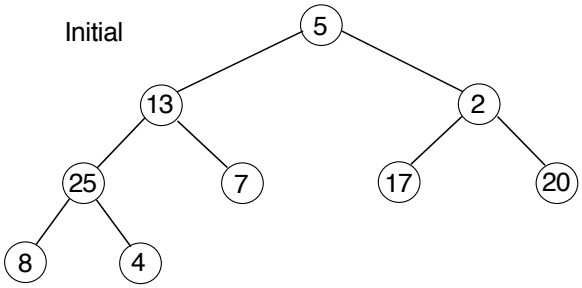
Outcome	Probability	Value Lost/Gained
-1	(5/6 * 5/6 * 5/6)	-1
+1	(1/6 * 5/6 * 5/6)*3	1
2	(1/6 * 1/6 * 5/6)*3	2
3	(1/6 * 1/6 * 1/6)	3

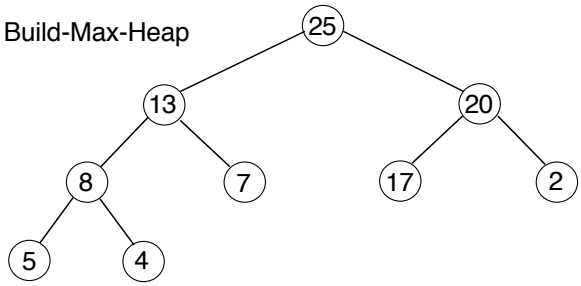
Expectation = -(125 / 216) + (75 / 216) + (30 / 216) + (3 / 216)
= -(17/216)

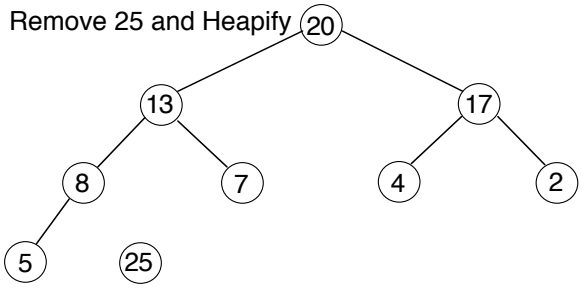
19. 6.2-5**.** lterative Heapify

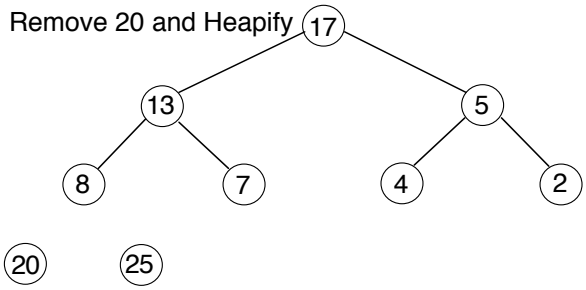
```
Heapify (A, i)
{
        do
        {
                p = i
                l = left(i)
                r = right(i)

                if (l <= heapsize(A) and A[l] > A[i])
                then
                        largest = l
                else
                        largest = i


                if (r <= heapsize(A) and A[r] > A[largest])
                then
                        largest = r

                if (largest < > i)
                then
                        Exchange(A[i],A[Largest])
                        i=largest
        }

while (p < > i)
}
```

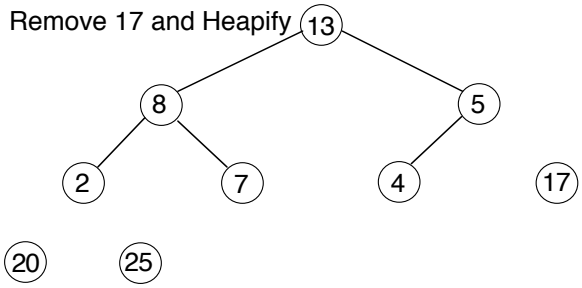
20. 6.4-1.

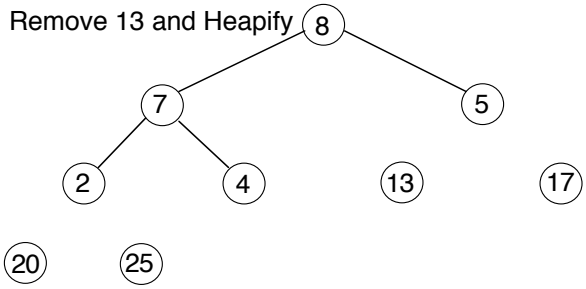

Initial
5
13
2
25
7
17
20
8
4

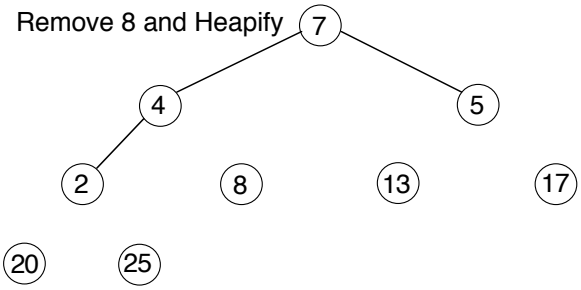

Build-Max-Heap
25
13
20
8
7
17
2
5
4

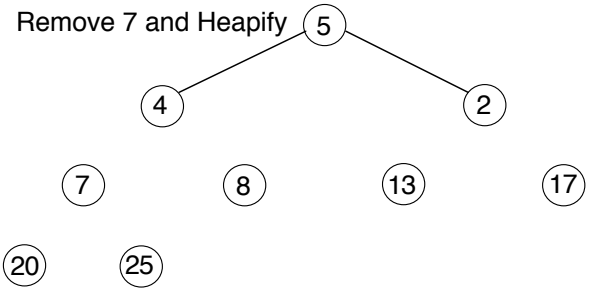

Remove 25 and Heapify
20
13
17
8
7
4
2
5

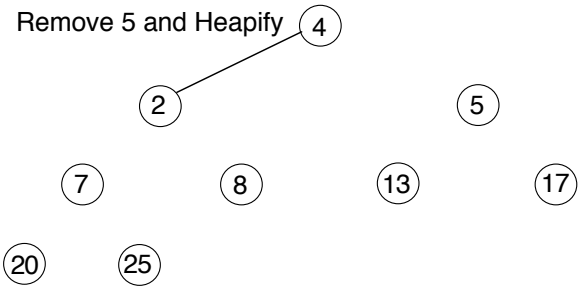

Remove 20 and Heapify
17
13
5
8
7
4
2

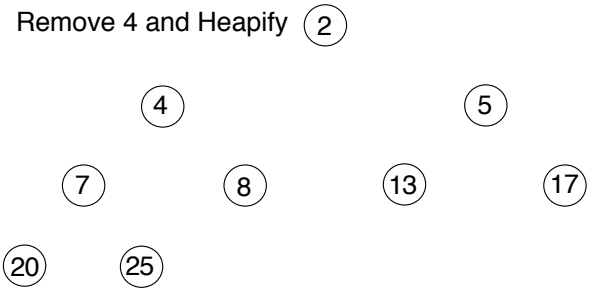

Remove 17 and Heapify
13
8
5
2
7
4


Remove 13 and Heapify
8
7
5
2
4


Remove 8 and Heapify
7
4
5
2

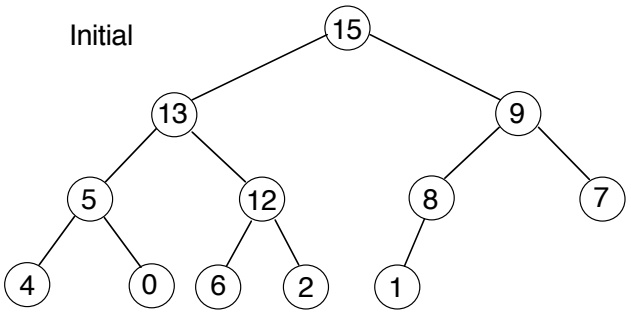

Remove 7 and Heapify
5
4
2


Remove 5 and Heapify
4
2
5

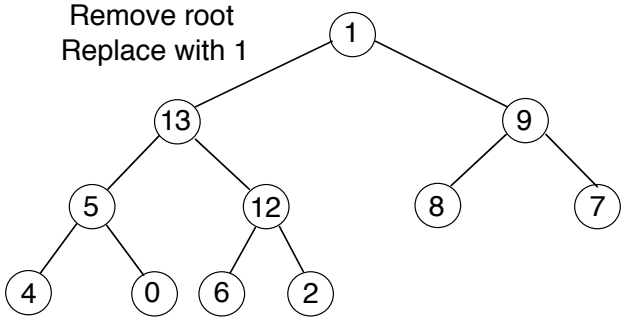

Remove 4 and Heapify
2
4
5

21. 6.5-1.

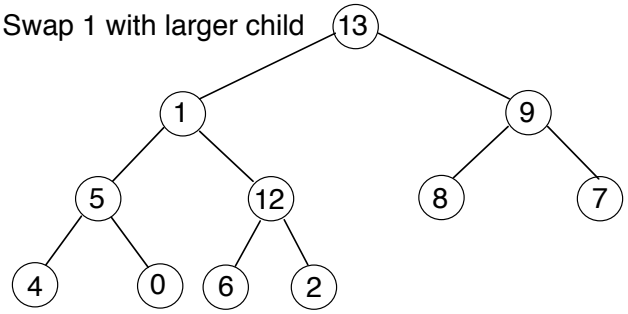
Initial



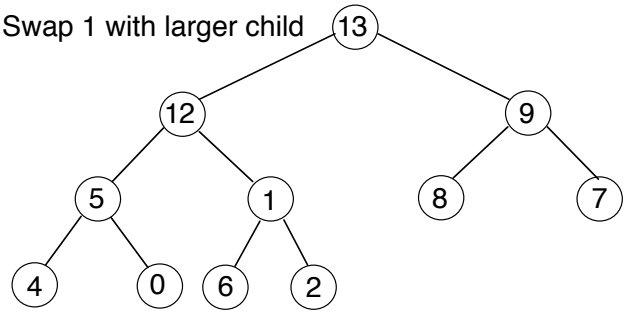
Remove root
Replace with 1



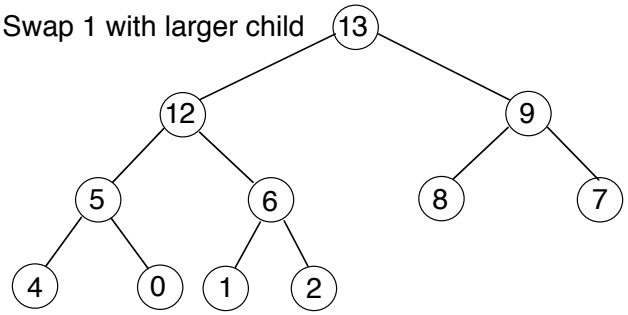
Swap 1 with larger child



Swap 1 with larger child

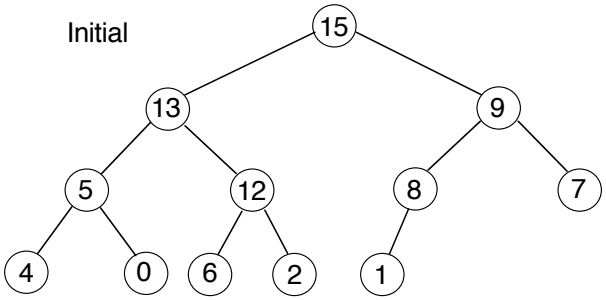


Swap 1 with larger child

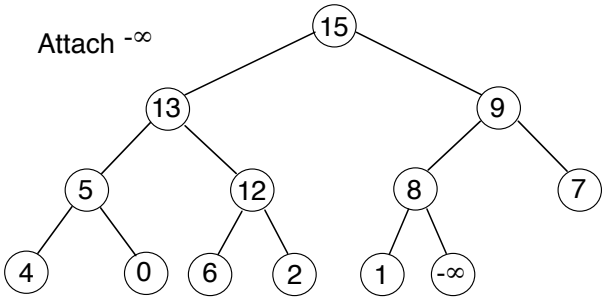


22. 6.5-2.

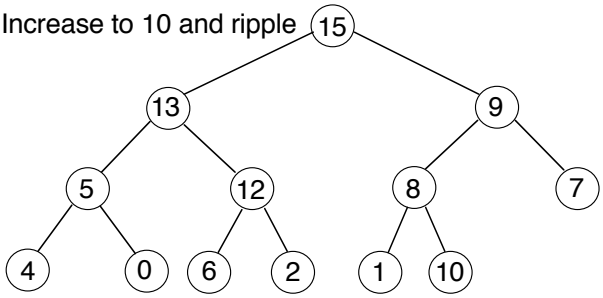
Initial

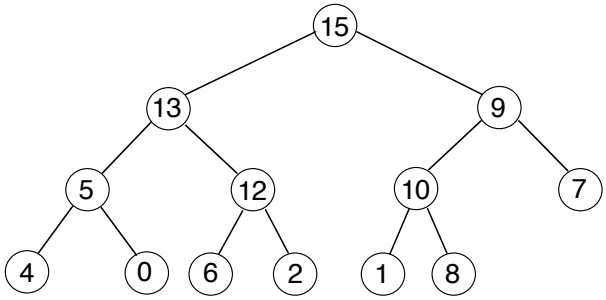


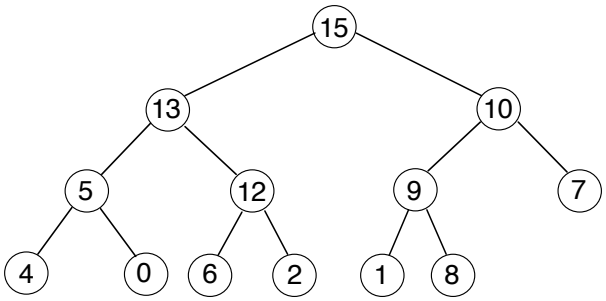
Attach $-\infty$



Increase to 10 and ripple







23. 15.2-1

Finding the optimal parenthesization of a matrix-chain product whose sequence of dimensions is <5, 10, 3, 12, 5, 50, 6>

The m table

0	150	330	405	1655	2010
0	0	360	330	2430	1950
0	0	0	180	930	1770
0	0	0	0	3000	1860
0	0	0	0	0	1500
0	0	0	0	0	0

The s table

0	1	2	2	4	2
0	0	2	2	2	2
0	0	0	3	4	4
0	0	0	0	4	4
0	0	0	0	0	5
0	0	0	0	0	0

The optimal parenthesization

(A1 * A2) * ((A3 *A4) * (A5 * A6)))

24. 15.2-2

```
MATRIX-CHAIN-MULTIPLY (A, s, i, j)
{
    if i =j
       C ← Ai
   else
      A ← MATRIX-CHAIN-MULTIPLY (A, s, i, s[i, j])
      B ← MATRIX-CHAIN-MULTIPLY (A, s, s[i,j]+1, j)
      C ←MATRIX-MULTIPLY (A, B)
  return (C)
}
```

25. 15.2-5

We can find the sum by making a note of the access pattern for the m table

Example: For n = 5

	1	2	3	4	5
1	4	3	2	1	0
2	X	4	3	2	1
3	X	X	4	3	2

4	X	X	X	4	3
5	X	X	X	X	4

X – Don't Care; The numbers in the cells indicate the number of accesses to the cell

It can be seen that

$$\sum_{i=1}^{n} \sum_{j=i}^{n} R(i,j) = \sum_{i=1}^{n} i(i-1)$$

$$= \sum_{i=1}^{n} i(i^2-1)$$

$$= n/6\ ((n+1)\ (2n+1)) - n/2\ (n+1)$$

$$= (n^3 - n)\ /3$$

26. 15.4-1

x = <1,0,0,1,0,1,0,1> y = <0,1,0,1,1,0,1,1,0>

		0	1	2	3	4	5	6	7	8	9
	y_j		0	1	0	1	1	0	1	1	0
0	x_i	0	0	0	0	0	0	0	0	0	0
1	1	0	↑0	↖1	←1	↖1	↖1	←1	↖1	↖1	←1
2	0	0	↖1	↑1	↖2	←2	←2	↖2	←2	←2	↖2
3	0	0	↖1	↑1	↖2	↑2	↑2	↖3	←3	←3	↖3
4	1	0	↑1	↖2	↑2	↖3	↖3	↑3	↖4	↖4	←4
5	0	0	↖1	↑2	↖3	↑3	↑3	↖4	↑4	↑4	↖5
6	1	0	↑1	↖2	↑3	↖4	↖4	↑4	↖5	↖5	↑5
7	0	0	↖1	↑2	↖3	↑4	↑4	↖5	↑5	↑5	↖6
8	1	0	↑1	↖2	↑3	↖4	↖5	↑5	↖6	↖6	↑6

27. 15.4-2

Print LCS without the b table

```
print_lcs (i,j)
{
        int i,j;                        /* i, j are the lengths of the two lists */
        if ((i = = 0) || (j = = 0))
            return;
        if(x[i]  = = y [j])             /* x, y are the two lists whose LCS is to be found */
        {
            print_lcs (i -1, j -1);
            printf("%d", x[i]);
        }
        else if (c[i-1][j] >= c[i][j])  /* c is the c table as in the algorithm */
        {
            print_lcs (i -1, j);
        }
        else
        {
            print_lcs (i, j-1);
        }
        return;
}
```

28. 15.4-5

Algorithm

- a[1 ..n] is the input sequence
- length[1..n] contains the length of the monotonically increasing subsequences up to a[i]= { i = 1 ..n}
- lms is the length of the longest monotonically increasing subsequence

```
for i = 2 to n do
        Begin
            for j = 1 to i -1 do
                Begin
                    Search for the j such that length[j] is the largest and a[i] can be
                    included in the subsequence it represents.
                End
                length[i] = length[j] + 1
                if  lms < length[i] then lms = length[i]
        End
```

29. 15-3

Bitonic TSP

Points $P_0 \ldots P_{n-1}$ are sorted by increasing X- Coordinate

C(i, j) = Cost of achieving optimal pair of paths such that are paths ends with P_i, the other with P_j (i < j)

Base Case

C(0,1) = dist (0, 1)

General Case

C(i-1, i) = min {C(j, i-1) + dist(j, i)}

$0 \le j < i-1$

$C(i, j) = C(i, j-1) + dist(j-1, j)$ where $i < j-1$

Final solution

$$\min_{0 \le i < n-1} \{C(i, n-1) + dist(i, n-1)\}$$

30. 16.1-1

```
/* f [1…n] contains finishing times (sorted) of activities
   s [1…n] contains the starting times of those activities
   m[1…n] contains the number of activities from 1 .. i that can be scheduled m_i in the
          problem
   fm[1…n] indicates the finishing times of the tasks scheduled in each of m[1…n]  */


Begin
        m[1] = 1
        fm[1] = f[1]
        for i = 2 to n do
                Begin
                        if( fm[i-1] ≤ s[i] then
                           Begin
                                m[i] = m[i-1] + 1
                                fm[i] = f[i]
                           End
                        else
                             Begin
                                fm[i] = fm[i-1]
                                m[i] = m[i-1]
                             End
                End i
End
```

31. 16.1-4

```
n ← length [s]
for i ← 1 to n
        A[i] ← { ∅ }                    //each A[i] (lecture Hall) has a set of activities
        LIST_INSERT( L, i );
k ← 0
while L ≠ ∅
        do k ← k + 1
           i ← head [L]
          for j ← i + 1 to tail[L]
                do if s_j ≤ f_i
                        then A[k] ← A[k] U { j }
                                i ← j
                                LIST_DELETE( L, j )
return L // the final value of 'k' has the number of lecture halls
```

32. 16.2-4

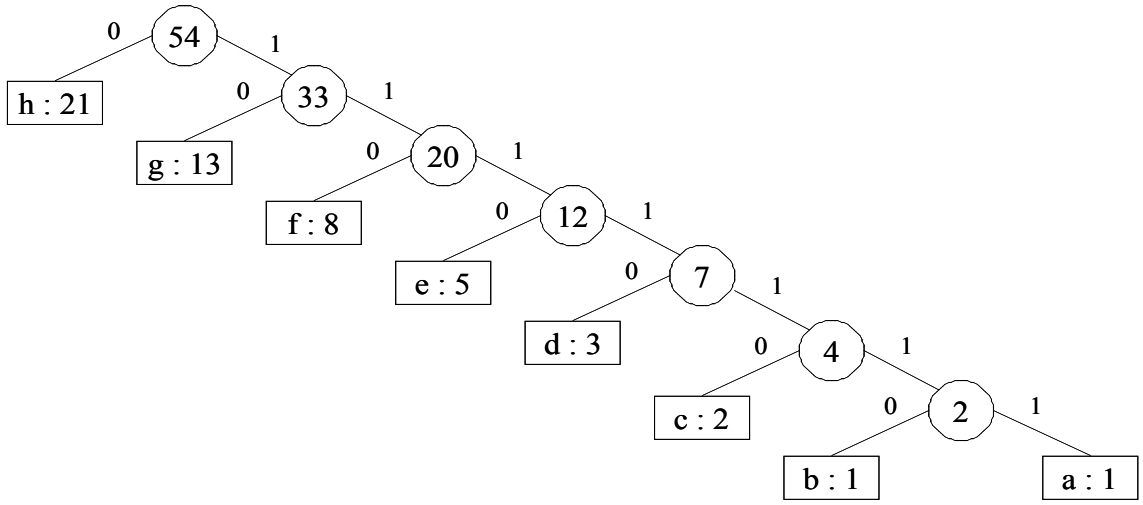
The greedy strategy would be to fill up the water bottle at the last moment i.e., Travel to the farthest water station that can be reached from the current water station (without falling short)

33. 16.2-5

Sort the points in ascending order of their k values

The greedy strategy would be to enclose the leftmost unenclosed point and all points that lie within a unit distance of this point. The next interval will begin at the closest point to the right of this interval

34. 16.3-3



Generalization:

code = $\begin{cases} \text{k-1 1's followed by a '0', if } k \le n-1 \\ \text{k 1's, } k = n \end{cases}$