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 $P_i = 1 / n$ and $T_i = i$. This yields

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

Thus, on average, about half the array will be checked. Running time for this algorithm is $(n+1)/2 = \Theta(n)$. 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 $a \wedge \log_b n = n \wedge \log_b a$

Take log to the base b of both sides of equation. This gives

$\log_b n * \log_b a = \log_b a * \log_b n$
LHS = RHS

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**

Proof by counterexample.
Let f(n) = n and $g(n) = n^2$
n^2 is an upper bound on g(n) while the reverse is not true

b.) f(n) + g(n) = Θ(min(f(n), g(n))). **FALSE**

Proof by counterexample
Let f(n) = n, $g(n) = n^2$. So for f(n) + g(n), min(f(n), g(n) = n
By definition it can be seen that $n + n^2 = \Theta(n)$ does not hold

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

By definition f(n) = O(g(n)) can be written as 0 <= f(n) <= c1 g(n) for all n > n0 (1)
Taking log on both sides 0 <= lg(f(n)) <= c2 lg(g(n)) for all n > n0 (2)

Using (2) and substituting one lg(f(n) <= c2 lg(k * f(n)) for all n > n0. This is always true.

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

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

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

Let f(n) be any positive, decreasing function, e.g. 1/n.

f.) f(n) = O(g(n)) implies g(n) = Ω(f(n)) **TRUE**

By transpose symmetry, f(n) = O(g(n)) iff g(n) = Ω(f(n))

g.) f(n) = Θ(f(n/2)) **FALSE**

Proof by counterexample
Let $f(n) = 2^n$
$2^n \neq O(2^{n/2})$

h.) f(n) + o(f(n)) = Θ(f(n)) **TRUE**

For whatever function g(n) = o(f(n)) is chosen, ∃ n_0, c such that g(n) <= cf(n) when n >= n_0.
From this and exercise 3.1-1 (done in lecture) max(g(n), cf(n)) = cf(n) = Θ(f(n)).

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

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

$$= 2 \sum_{k=1}^{n} k - \sum_{k=1}^{n} 1$$

$$= 2 * (n/2) * (n+1) - n$$

$$= n^2 + n - n$$

$$= n^2$$

9. A.2-2
Find an asymptotic upper bound on

$$\sum_{k=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^k} \right\rceil$$

$$\leq \sum_{k=0}^{\lg n} \left(\frac{n}{2^k} + 1 \right) = \lg n + n \sum_{k=0}^{\lg n} \frac{1}{2^k} \leq \lg n + n \frac{1}{1 - \frac{1}{2}} = \lg n + 2n = O(n)$$

10. A.2-3

Show that the nth harmonic is $\Omega(\lg n)$ by splitting summations

$$H_n = \sum_{k=1}^{n} 1/k >= \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{i=0}^{(2^i - 1)} 1/(2^i + j)$$

$$>= \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{i=0}^{(2^i - 1)} 1/(2 * 2^i)$$

$$>= (1/2) \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{i=0}^{(2^i - 1)} 1/(2^i)$$

$$>= (1/2) \sum_{i=0}^{\lfloor \lg n \rfloor - 1} 1$$

$$>= (1/2) \lg n$$

$H_n = \Omega(\lg n)$

11. A.2-4

Approximate $\sum_{k=1}^{n} k^3$ with an intergral.

$$\int_0^n k^3 \, dk < = \sum_{k=1}^{n} k^3 <= \int_1^{n+1} k^3 \, dk$$

$$[k^4 / 4]_0^n <= \sum_{k=1}^{n} k^3 <= [k^4 / 4]_1^{n+1}$$

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

$n^4 = \Theta(n^4)$ and $(n+1)^4 / 4 - ¼ = \Theta(n^4)$

Thus $\sum_{k=1}^{n} k^3 = \Theta(n^4)$.

12. 4.1-1

Show that the solution of $T(n) = T(\lceil n/2 \rceil) + 1$ is $O(\lg n)$.

We have to show that $T(n) <= c * \lg n$ for some $c>0$

Assume that $T(n/2) <= c * \lg(n/2)$ and prove $T(n) <= c * \lg n$.

$T(n) = T(n/2) + 1 <= c * \lg(n/2) + 1 <= c*\lg n - c*\lg 2 + 1 <= c*\lg n - c + 1 <= c*\lg n - (c - 1)$

For c >= 1, we can add (c-1) to the RHS, and preserve inequality.

This yields $T(n) <= c * \lg n - (c-1) + (c - 1) <= c * \lg n$.

Thus, $T(n) = O(\lg n)$ for $c >= 1$

13. 4.1-2
Show the solution of $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is $\Omega(n \lg n)$ and conclude that the solution is $\Theta(n \lg n)$.

1) We have to show that $T(n) >= c * n \lg n$ for some $c >= 0$.
   Assume that $T(n/2) >= c(n/2) \lg (n/2)$ and prove $T(n) >= c * n \lg n$.
   $T(n) = 2T(n/2) + n >= 2c(n/2) \lg(n/2) + n >= c*n \lg n - cn + n >= cn \lg n - (c-1)n$ (1)
   For $c <= 1$, we can add $(c-1)n$ to the RHS and preserve inequality.
   This yields $T(n) >= c*n \lg n - (c-1)n + (c-1)n >= c*n \lg n$ (2)
   Thus, $T(n) = \Omega(n \lg n)$ for $0 < c <= 1$.

2) Flipping relational operation in (1), we obtain
   $T(n) <= c*n \lg n - (c-1)n$
   For $c >= 1$, we can add $(c-1)n$ to the RHS, and preserve the inequality.
   This yields $T(n) <= c*n \lg n$
   We proved that $T(n) = O(n \lg n)$.

   $T(n) = \Omega(n \lg n)$ and $T(n) = O(n \lg n)$ implies that $T(n) = \Theta(n \lg n)$

14. 4.2-1
Determine a good asymptotic upper bound on the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$ by iteration

Let $T(1) = 1$ and let $n = 2^k$

$T(n) = 3T(\lfloor n/2 \rfloor) + n$
$T(n) = 3(3T(\lfloor n/4 \rfloor) + n/2) + n = 3^2T(\lfloor n/4 \rfloor) + 3(n/2) + n$
$T(n) = 3^2 (3T(\lfloor n/8 \rfloor) + n/4) + 3(n/2) + n = 3^3 T(\lfloor n/8 \rfloor) + 3^2(n/4) + 3(n/2) + n$
…
$T(n) = 3^k T(\lfloor n/2^k \rfloor) + (3/2)^{k-1} n + (3/2)^{k-2} n + \ldots + (3/2)n + n$

$T(n) = 3^k T(1) + n((3/2)^{k-1} + (3/2)^{k-2} + \ldots + (3/2)^2 + (3/2) + 1)$ <= Geometric Series

$$T(n) = 3^k + n \frac{(3/2)^k - 1)}{(3/2) - 1)}$$ $(3/2) - 1 = (1/2)$. So multiply top half by 2

$$T(n) = 3^{\log_2 n} + 2n \frac{(3^k - 2^k)}{2^k}$$ Remember that $n = 2^k$, so $k = \log_2 n$
Multiply top and bottom by 2^k

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

$T(n) = n^{\log_2 3} + 2(n^{\log_2 3} - n) \quad = \quad n^{\log_2 3} + 2n^{\log_2 3} - 2n \quad = \quad 3n^{\log_2 3} - 2n$

Therefore $T(n) <= 3n^{\log_2 3}$

$T(n) = O(n^{\log_2 3})$ or **$O(n^{1.58})$**

15. 4.2-2

Argue that the solution to the recurrence
$T(n) = T(n/3) + T(2n/3)$ is **Ω**(nlgn) by appealing to the recursion tree.

From Figure 4.2, the least depth of complete levels is $\log_{3/2} n$, and each level adds n to the algorithm's running time. Thus, the algorithm does not run less than nlgn.

Hence, $T(n) = \Omega(nlgn)$.

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,

Expectation = $\sum_{i=1}^{n}$ (i/n) = (i/n) * $\sum_{i=1}^{n}$ (i) = l/n * n(n + 1) /2 = (n + 1)/2

The probability of the maximum element in any of the n positions is l/n. Similarly E for the **minimum** element is also (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

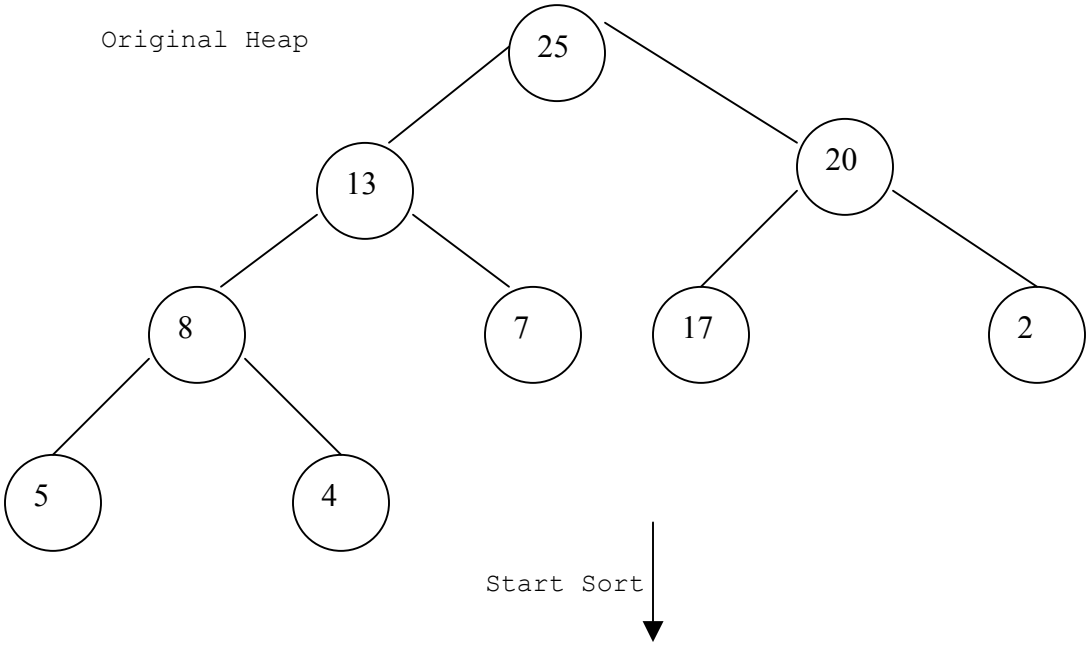
Expectation = -(125 / 216) + (75 / 216) + (30 / 216) + (3 / 216)
= -(17/216)

19. 6.2-5
lterative Heapify

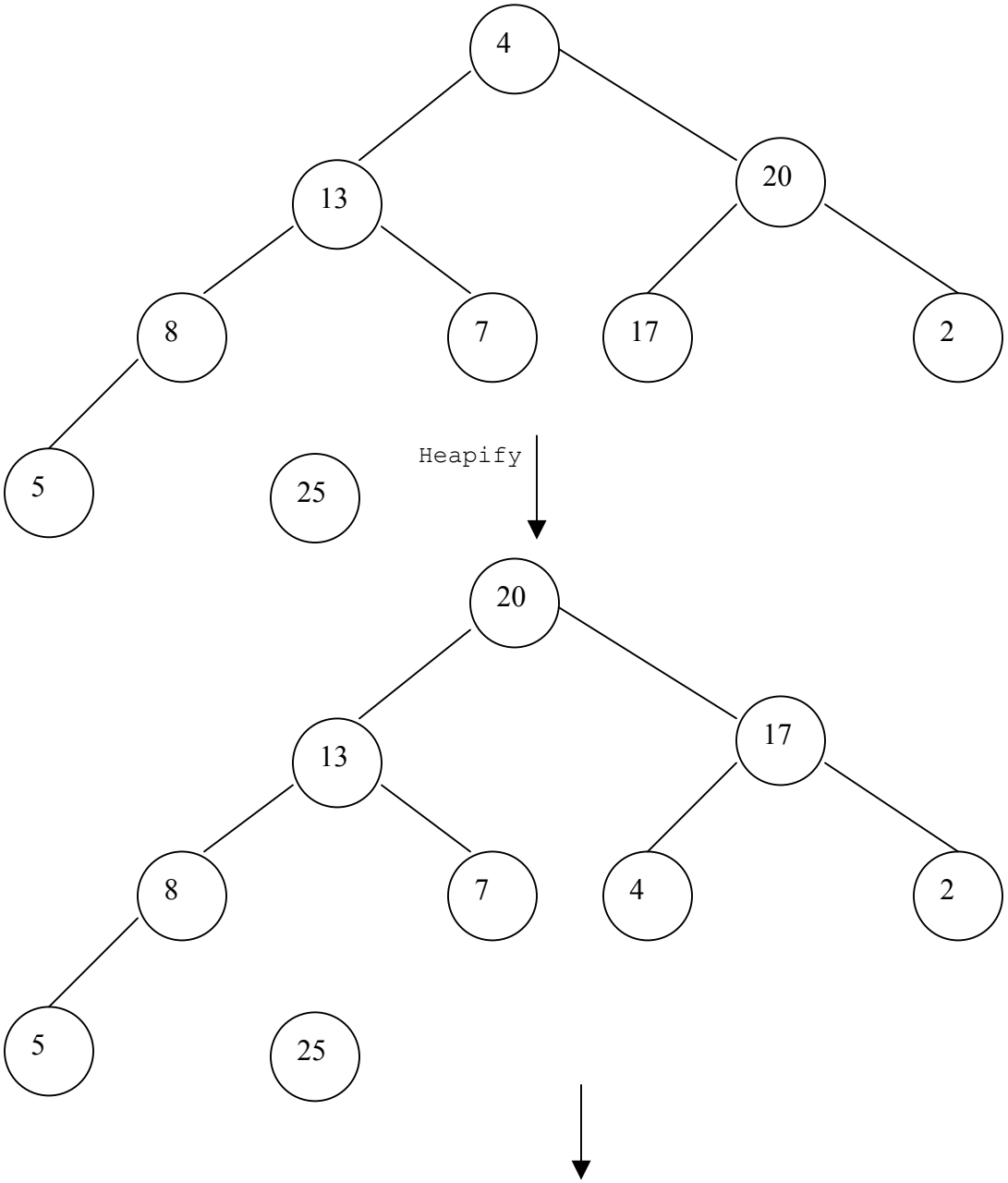
```
Heapify (A, i)
{
        do
        {
                p = i
                l = left(i)
                r = right(i)

                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)
}
```

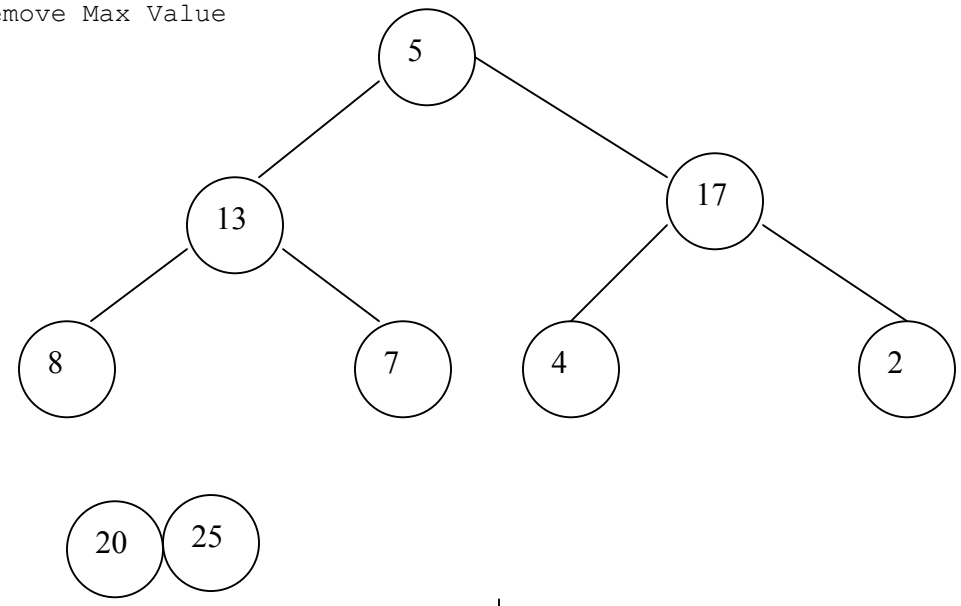
20. 6.4-1



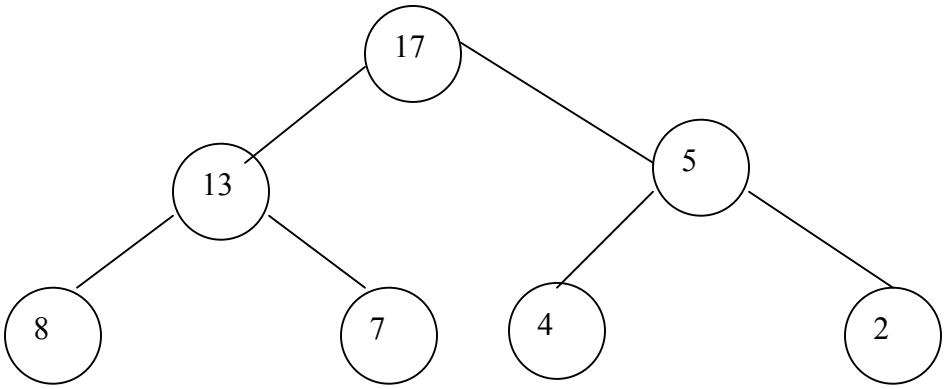
Remove Max Value

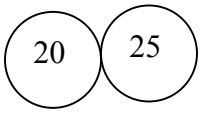

4
13
20
8
7
17
2
5
25
Heapify
20
13
17
8
7
4
2
5
25

Remove Max Value

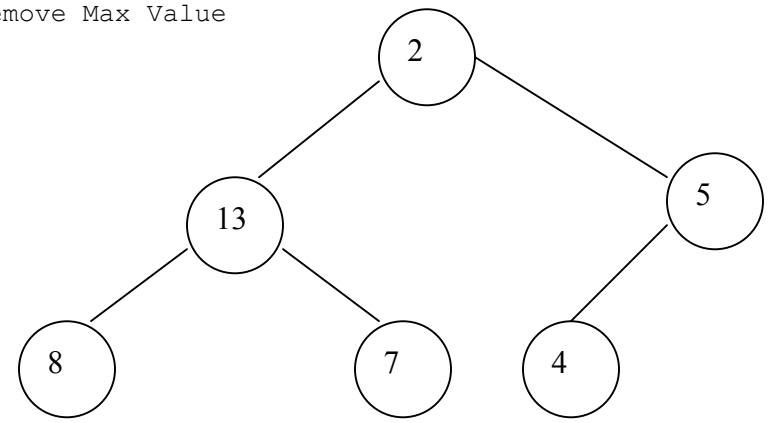


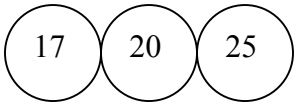
Heapify



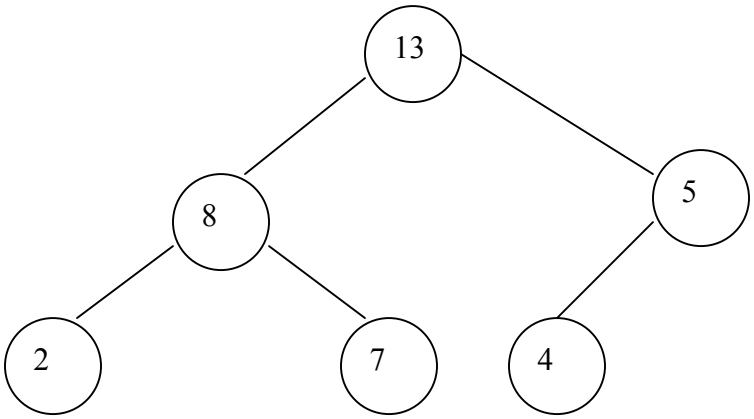


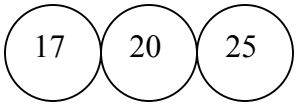
Remove Max Value



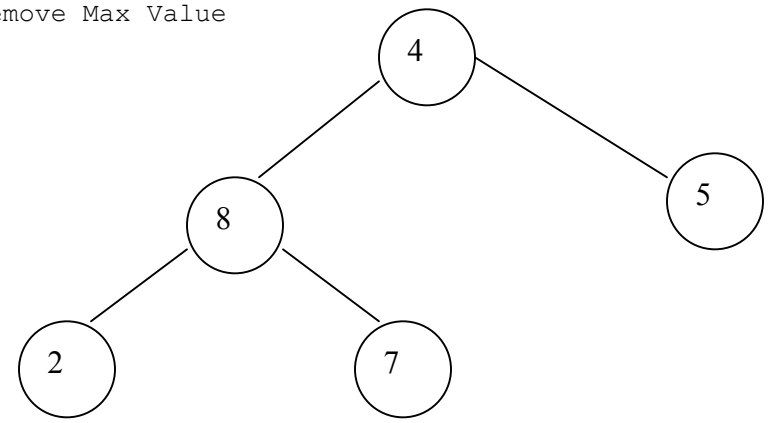


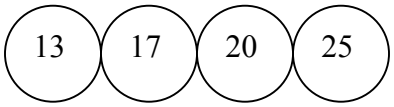
Heapify



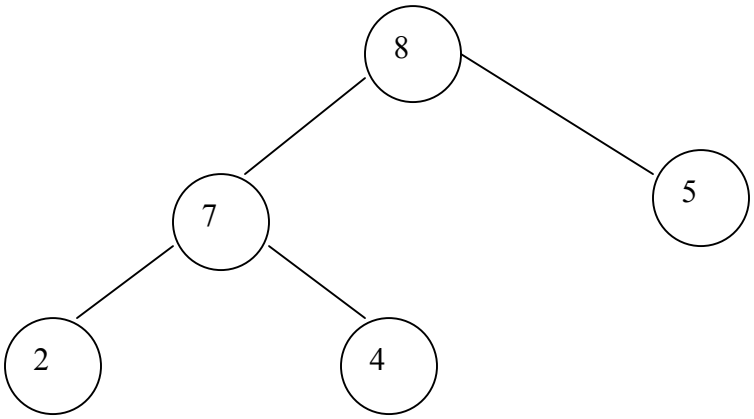


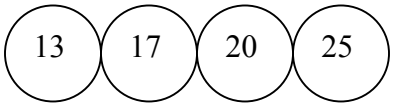
Remove Max Value



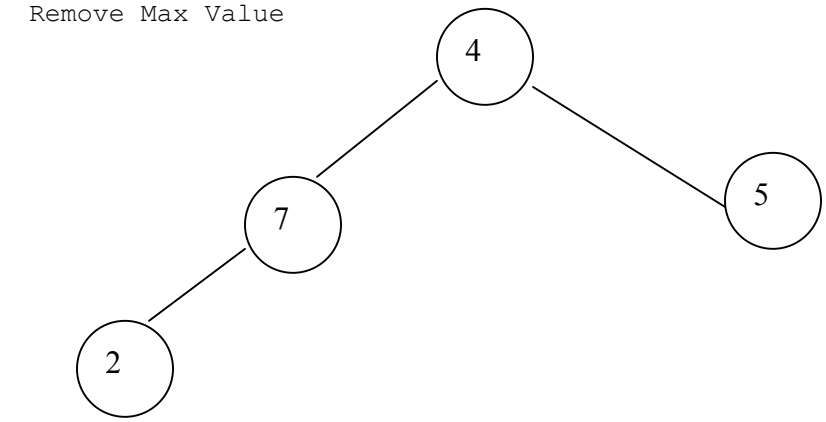


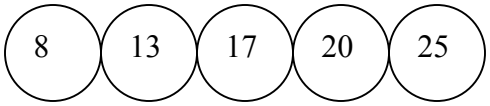
Heapify



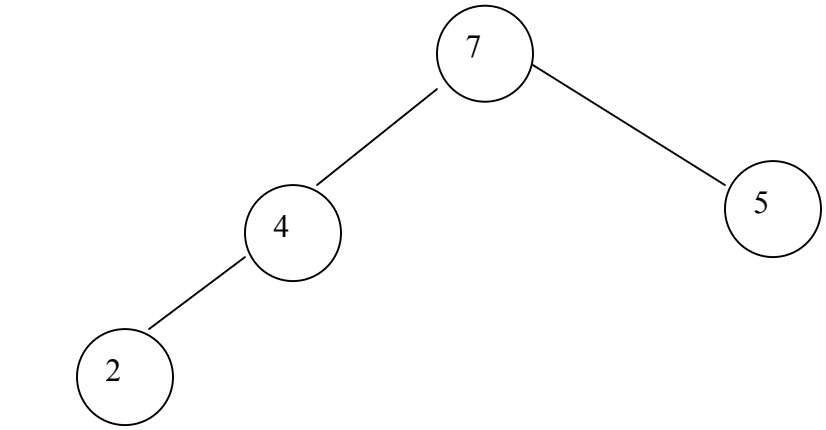


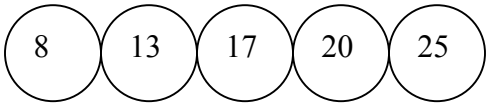
Remove Max Value

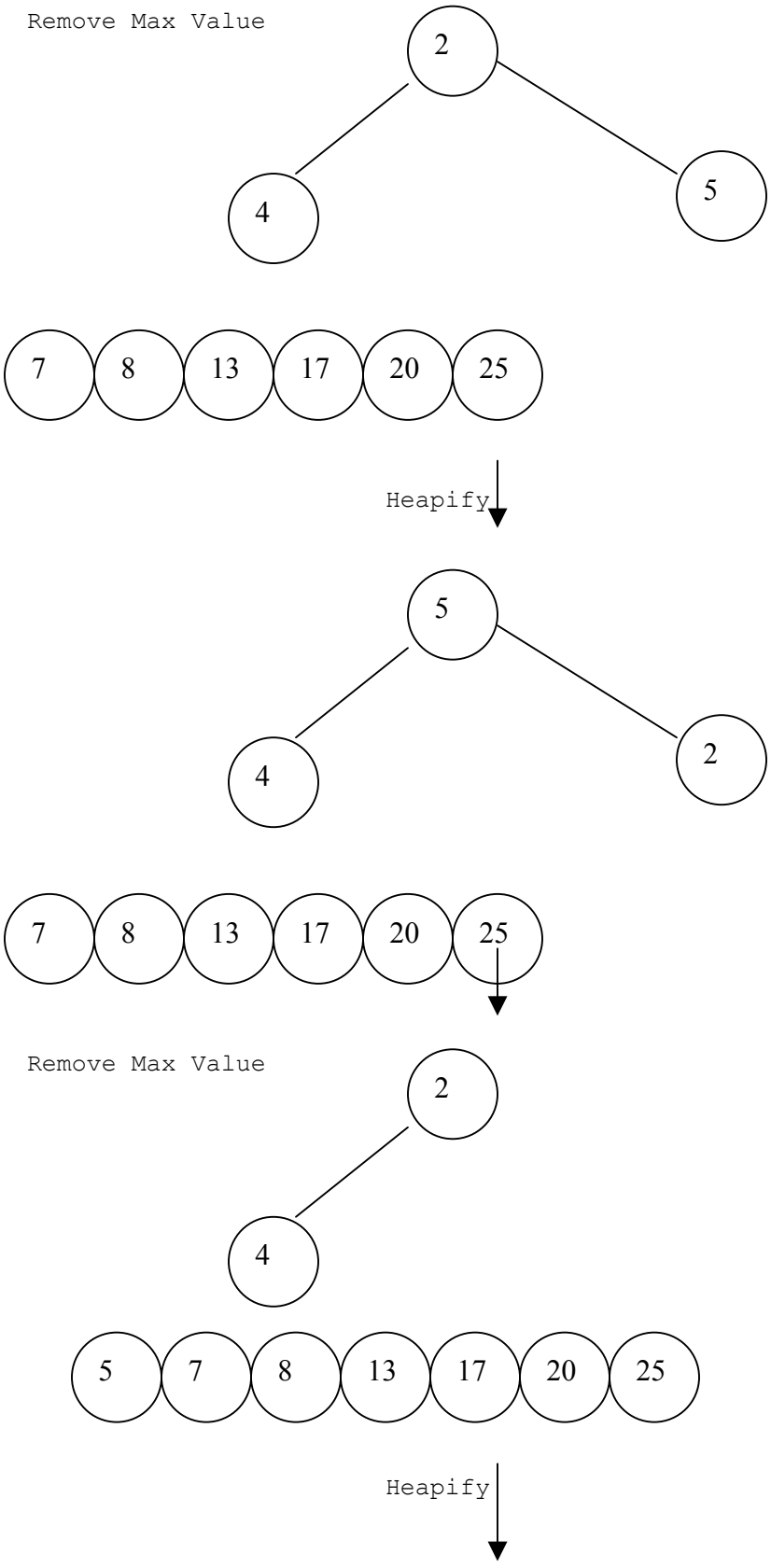


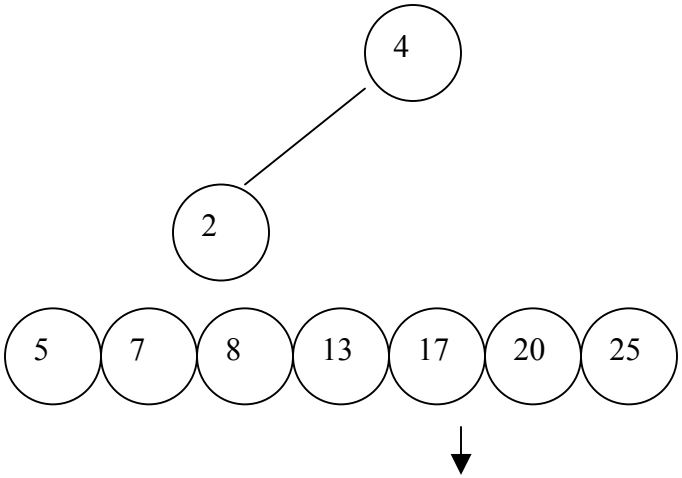


Heapify

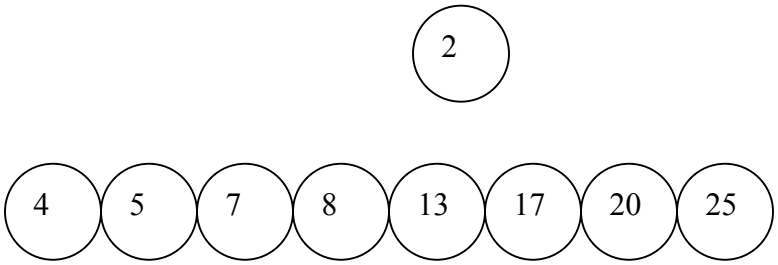




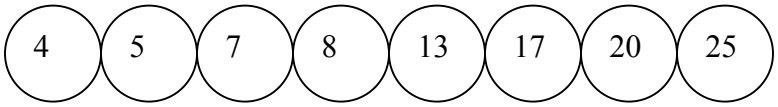

Remove Max Value
2
4
5
7
8
13
17
20
25
Heapify
5
4
2
7
8
13
17
20
25
Remove Max Value
2
4
5
7
8
13
17
20
25
Heapify



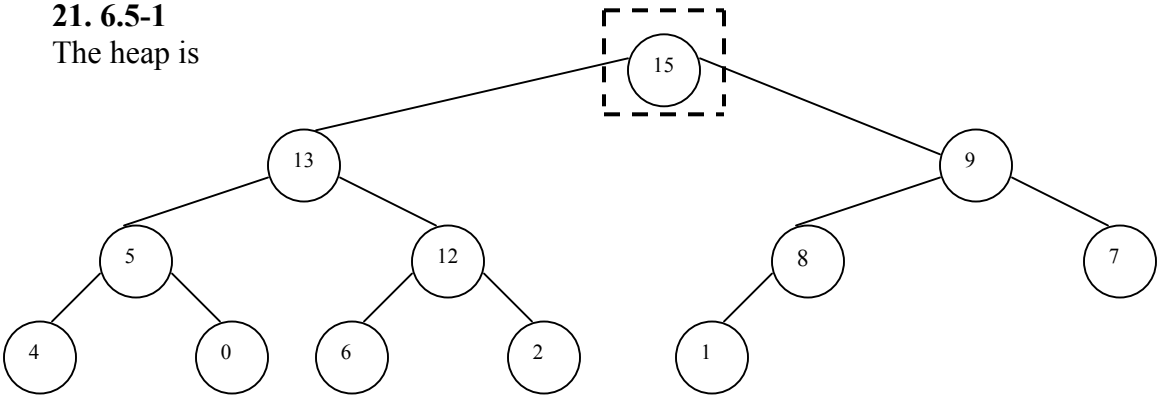
Remove Max Value



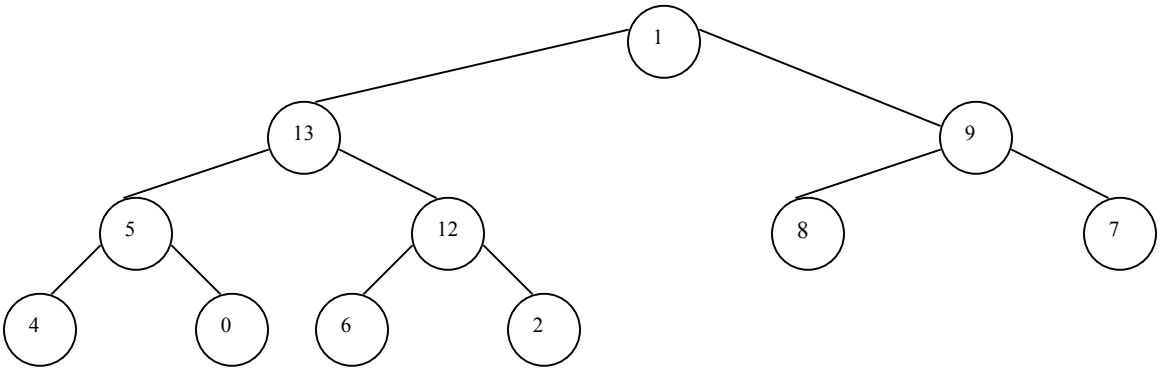
Heapify



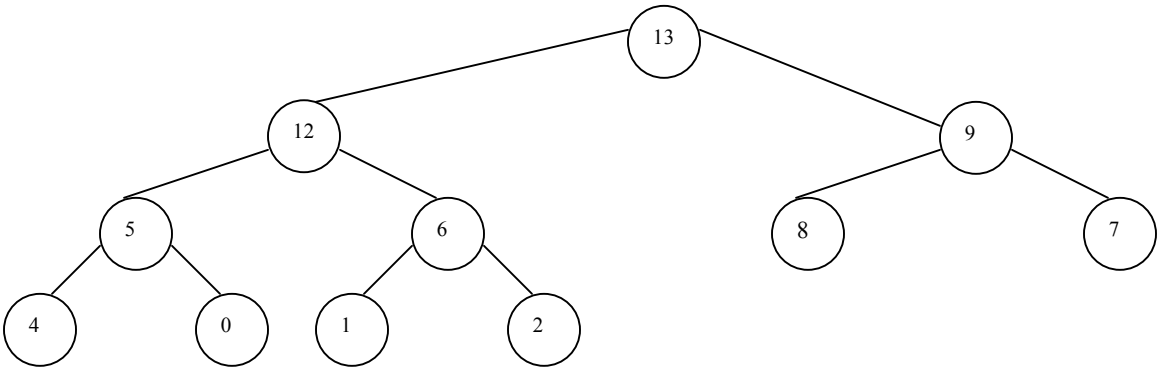
21. 6.5-1
The heap is



After deleting the root and moving 1 at the root

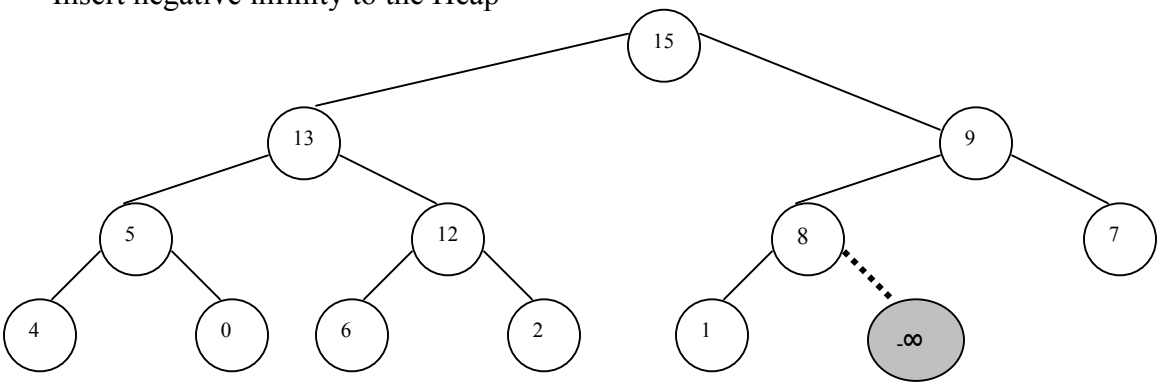


Heap after applying Heapify

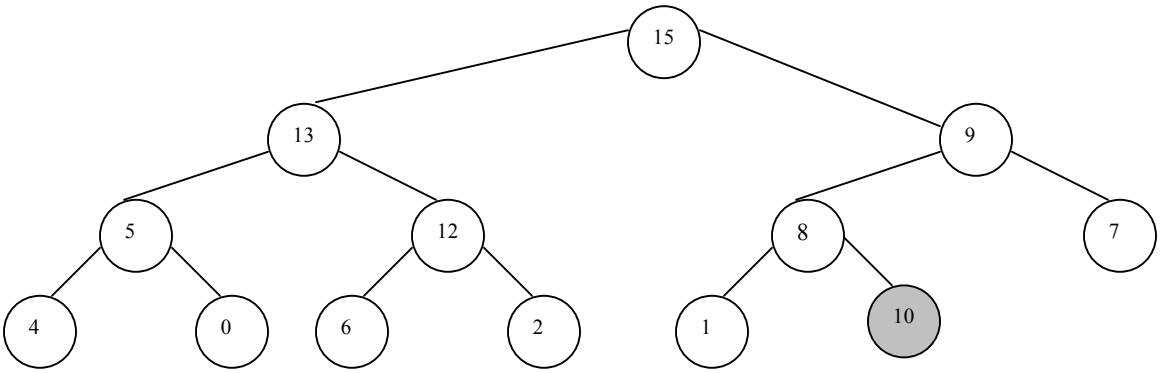


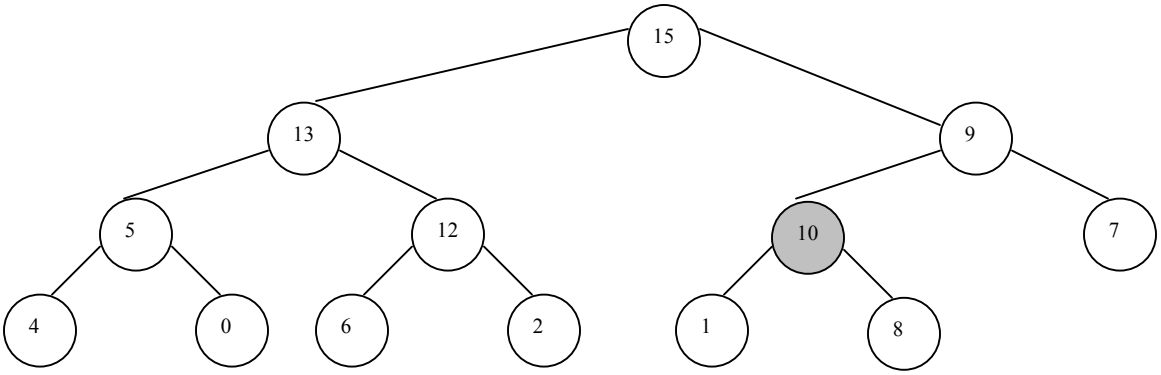
22. 6.5-2

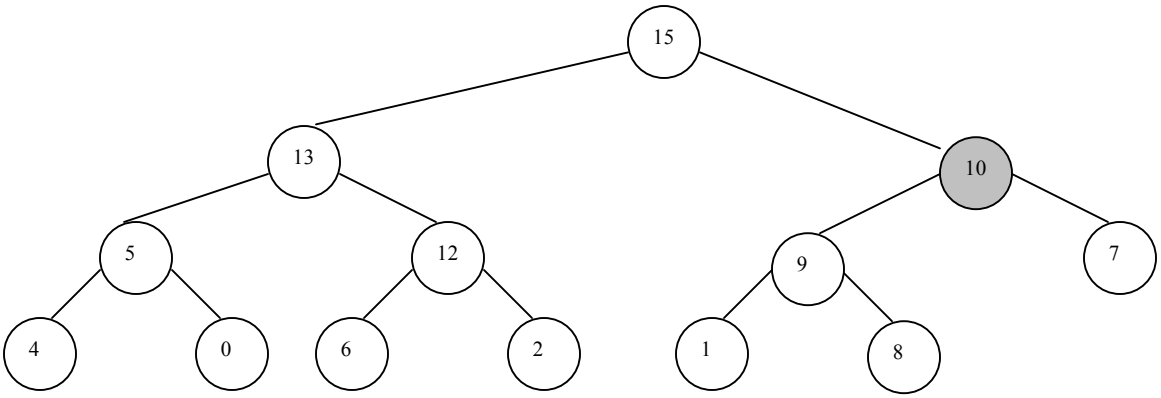
Insert negative infinity to the Heap



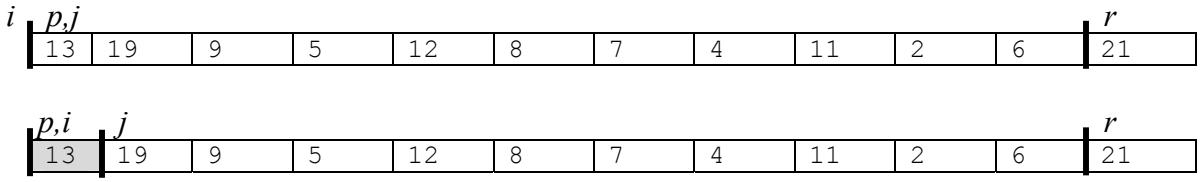
Heap Increase Key from negative infinity to 10

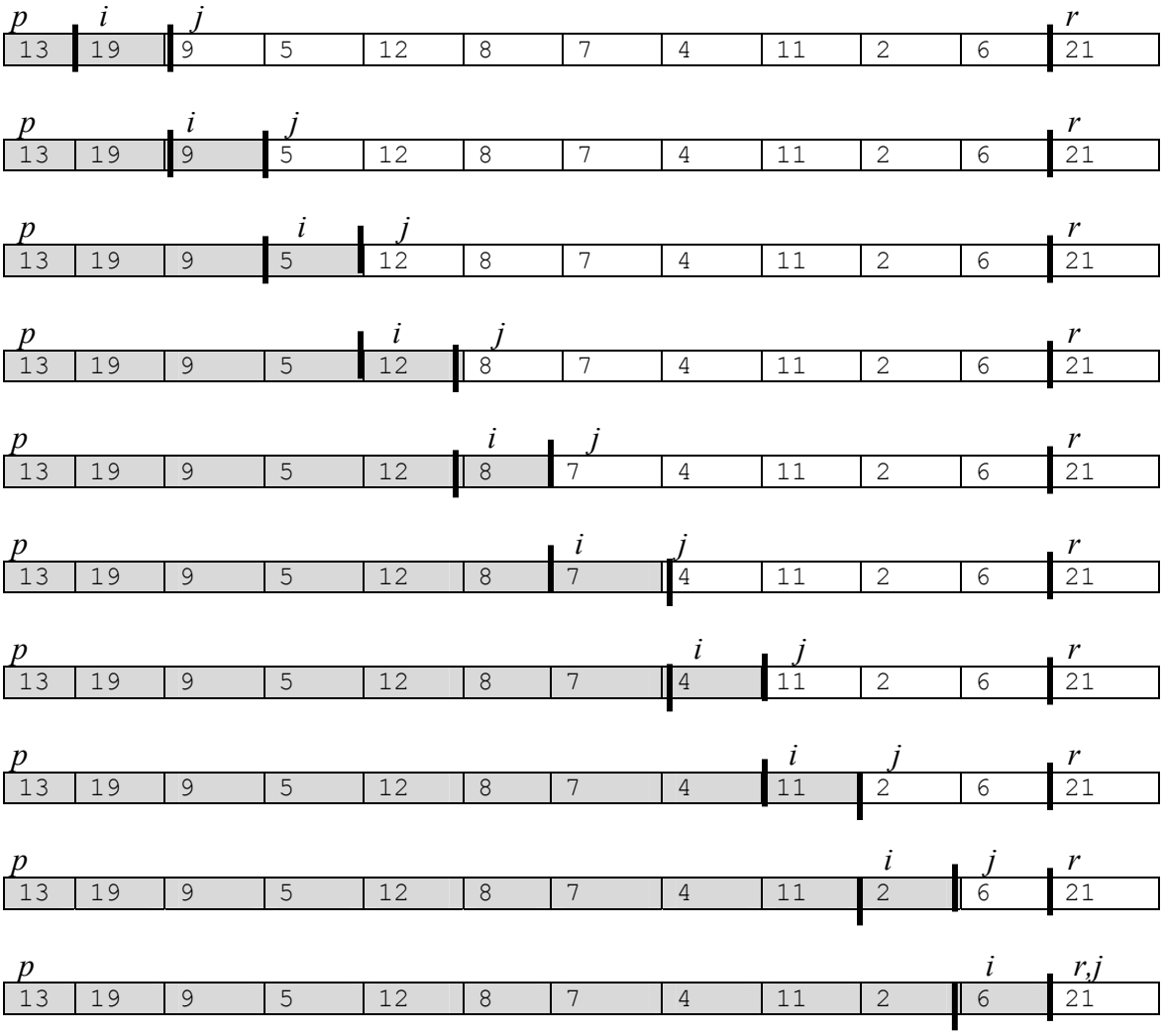

15
13
9
5
12
8
7
4
0
6
2
1
10


15
13
9
5
12
10
7
4
0
6
2
1
8


15
13
10
5
12
9
7
4
0
6
2
1
8

23. 7.1-1


i
p,j
r
13 19 9 5 12 8 7 4 11 2 6 21
p,i
j
r
13 19 9 5 12 8 7 4 11 2 6 21



24 & 25. 7.1-2, 7.2-2

Show that the running time of the quicksort is $\Theta(n^2)$, when all the elements in the array have the same value.

Solution: It is a special case of quicksort, when all the elements of the array have the same value. **For this special case the algorithm "partition" always returns the value of q as r.** Where r is the size of the partition (i.e. passed to the partition algorithm). So the array gets divided by (n-1) and 1 elements

Thus if we have an array of n elements, we can write the following binary tree

$T(n) = T(n-1) + \Theta(1)$

$= \sum_{k=1}^{n} \Theta(k)$

$= \Theta(\sum_{k=1}^{n})$

$T(n) = \Theta(n^2)$

PARTITION Modification
To return q = (p+r)/2 when all elements in the array has the same value

PARTITION (A,p,r)

1. $x \leftarrow A[r]$
2. $i \leftarrow p-1$
3. $flag \leftarrow 0$
4. **for** $j \leftarrow p$ **to** $r-1$
5. **do if** $A[j] \leq x$
6. **then** $i \leftarrow i+1$
7. exchange $A[i] \Leftrightarrow A[j]$
8. **if** $A[j] \neq x$
9. $flag \leftarrow 1$
10. exchange $A[i+1] \Leftrightarrow A[j]$
11. **if** $flag = 1$

12. **return** i+1
13. **else**
14. **return** (p+r)/2

26. 8.2-1

(a)

A

1	2	3	4	5	6	7	8	9	10	11
7	1	3	1	2	4	5	7	2	4	3

C

1	2	3	4	5	6	7
2	2	2	2	1	0	2

(b)

C

1	2	3	4	5	6	7
2	4	6	8	9	9	11

(c)

B

1	2	3	4	5	6	7	8	9	10	11
					3					

C

1	2	3	4	5	6	7
2	4	5	8	9	9	11

(d)

B

1	2	3	4	5	6	7	8	9	10	11
					3		4			

C

1	2	3	4	5	6	7
2	4	5	7	9	9	11

(e)

B

1	2	3	4	5	6	7	8	9	10	11
			2		3		4			

C

1	2	3	4	5	6	7
2	3	5	7	9	9	11

(f)

B

1	2	3	4	5	6	7	8	9	10	11
1	1	2	2	3	3	4	4	5	7	7

27. 8.3-1

Initial	Step1	Step2	Step3
COW	SEA	BAR	BAR
DOG	TEA	EAR	BIG
SEA	MOB	TAB	BOX
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX	DIG	DIG	EAR
TAB	BIG	BIG	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX	FOX	RUG	TEA
COLUMN AFFECTED	↑	↑	↑