1. (a) Show the following

$$2n^2 = \Theta(n^2),$$
$$2n^2 \neq O(n). \qquad (10)$$

(b) Let $a > 1$ and $b > 1$ be constants. Let $f(n)$ be a function, and. Let $T(n)$ be defined on nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Show that if $f(n) = \Omega(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$. (10)

1. (a) Show the following

$$2n^2 = \Theta(n^2),$$
$$2n^2 \neq O(n). \qquad (10)$$

- We can find $c_1 = 1$, $c_2 = 3$, and $n_0 = 0$ such that when $n > n_0$ we have

$$c_1 n^2 < 2n^2 < c_2 n^2.$$

Thus, $2n^2 = \Theta(n^2)$ holds.

- Assume that we have $2n^2 = O(n)$, which implies that there exist c and n_0 such that when $n > n_0$, we have

$$2n^2 \leq cn,$$

which shows

$$n \leq c_2/2.$$

This is contradicting to the assumption that we have the inequality for any $n > n_0$.

(b) Let $a > 1$ and $b > 1$ be constants. Let $f(n)$ be a function, and. Let $T(n)$ be defined on nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Show that if $f(n) = \Omega(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$. (10)

By using the recursion tree method, we will get

$$T(n) = \sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right)$$

If $f(n) = \Omega(n^{\log_b a-\varepsilon})$, we have

$$T(n) = \sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right) = \sum_{i=0}^{\log_b n} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\varepsilon}$$

$$T(n) = \sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right) = \sum_{i=0}^{\log_b n} a^i \left(\frac{n}{b^i}\right)^{log_b a-\varepsilon}$$

$$= n^{log_b a-\varepsilon} \sum_{i=0}^{log_b n} \left(\frac{ab^{\varepsilon}}{b^{log_a a}}\right)^i$$

$$= n^{log_b a-\varepsilon} \sum_{i=0}^{log_b n} \left(\frac{ab^{\varepsilon}}{a}\right)^i = n^{log_b a-\varepsilon} \sum_{i=0}^{log_b n} (b^{\epsilon})^i$$

$$= n^{log_b a-\varepsilon} \left(\frac{b^{\varepsilon log_b n} - 1}{b^{\epsilon} - 1}\right)$$

$$= n^{log_b a-\varepsilon} \left(\frac{n^{\epsilon} - 1}{b^{\epsilon} - 1}\right)$$

$$= n^{log_b a}$$

2. The following algorithm is used by the Merge sort to merge two sorted subsequences.

```
Merge(A, p, q, r)
    n1 ← q – p + 1
    n2 ← r – q
    for i ← 1 to n1
        do L[i] ← A[p + i – 1]
    for j ← 1 to n2
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ← p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                i ← i + 1
            else A[k] ← R[j]
                j ← j + 1
```

Show its correctness by establishing the loop invariant.

Merge(*A*, *p*, *q*, *r*)

```
n1 ← q − p + 1
n2 ← r − q
    for i ← 1 to n1
        do L[i] ← A[p + i − 1]
    for j ← 1 to n2
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ← p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                i ← i + 1
            else A[k] ← R[j]
                j ← j + 1
```

Loop Invariant for the *for* loop

- At the start of each iteration of the for loop:

subarray $A[p \,..\, k - 1]$ contains the $k - p$ smallest elements of L and R in sorted order.

- $L[i]$ and $R[j]$ are the smallest elements of L and R that have not been copied back into A.

Initialization:

Before the first iteration:

- $A[p \,..\, k - 1]$ is empty.
- $i = j = 1$.
- $L[1]$ and $R[1]$ are the smallest elements of L and R not copied to A.

Merge(A, p, q, r)

1 $n_1 \leftarrow q - p + 1$
2 $n_2 \leftarrow r - q$
3 **for** $i \leftarrow 1$ **to** n_1
4   **do** $L[i] \leftarrow A[p + i - 1]$
5 **for** $j \leftarrow 1$ **to** n_2
6   **do** $R[j] \leftarrow A[q + j]$
7 $L[n_1+1] \leftarrow \infty$
8 $R[n_2+1] \leftarrow \infty$
9 $i \leftarrow 1$
10 $j \leftarrow 1$
11 **for** $k \leftarrow p$ **to** r
12   **do if** $L[i] \leq R[j]$
13   **then** $A[k] \leftarrow L[i]$
14             $i \leftarrow i + 1$
15   **else** $A[k] \leftarrow R[j]$
16             $j \leftarrow j + 1$

Maintenance:

Case 1: $L[i] \leq R[j]$

- By Loop Invariant (LI), A contains $k - p$ smallest elements of L and R in *sorted order*.
- By LI, $L[i]$ and $R[j]$ are the smallest elements of L and R not yet copied into A.
- Line 13 results in A containing $k - p + 1$ smallest elements (again in sorted order). Incrementing i and k reestablishes the LI for the next iteration.

Similarly for Case 2: $L[i] > R[j]$.

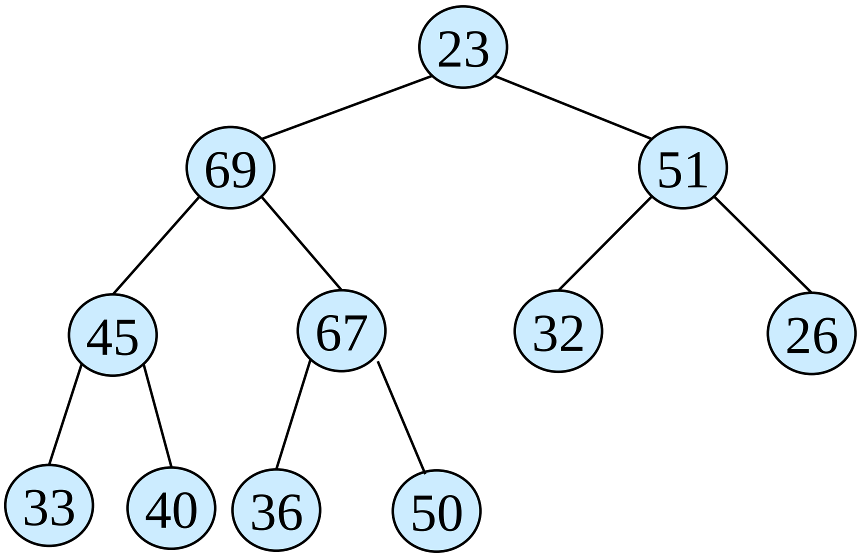
Termination:

- On termination, $k = r + 1$.
- By LI, A contains $r - p + 1$ smallest elements of L and R in sorted order.
- L and R together contain $r - p + 3 = 2$ elements.

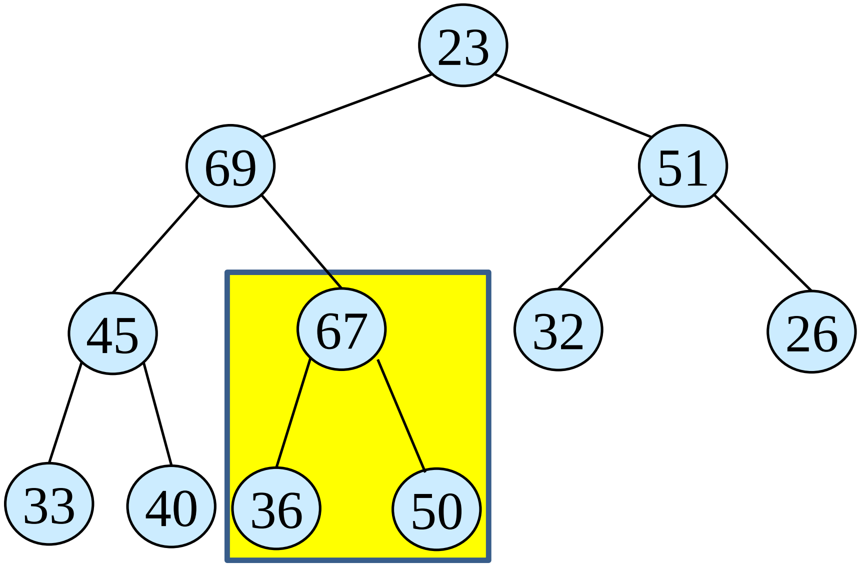
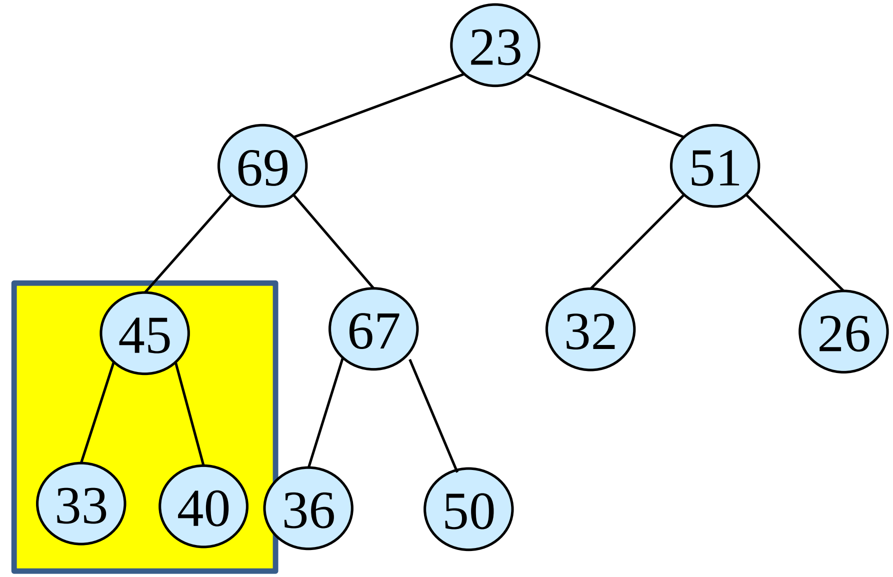
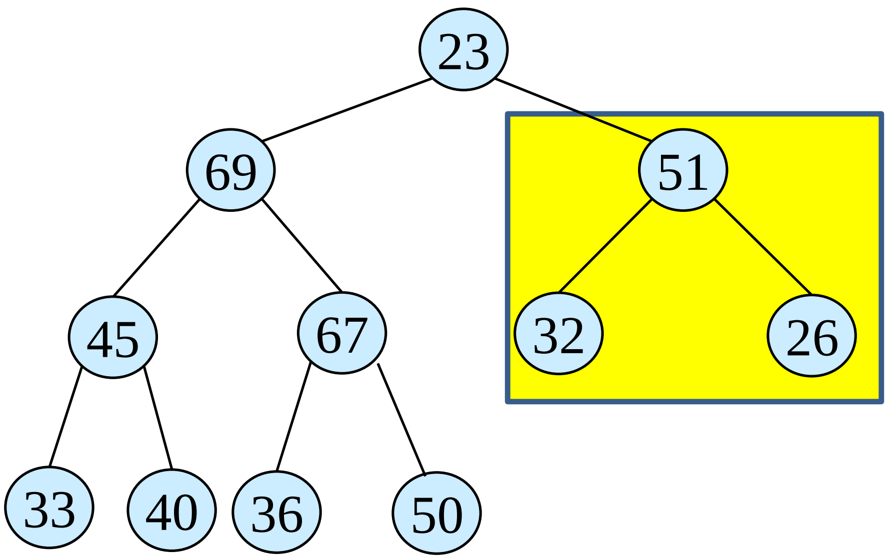
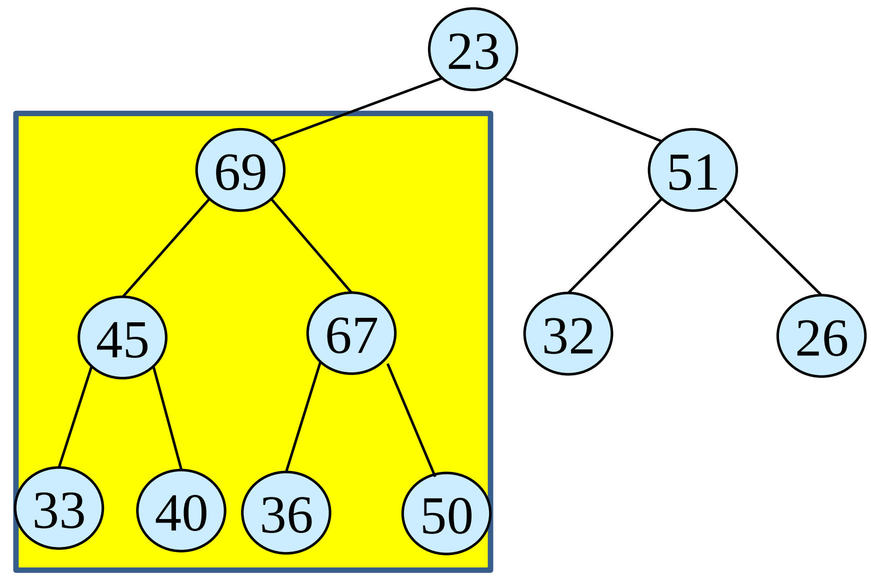
All but the two sentinels have been copied back into A.

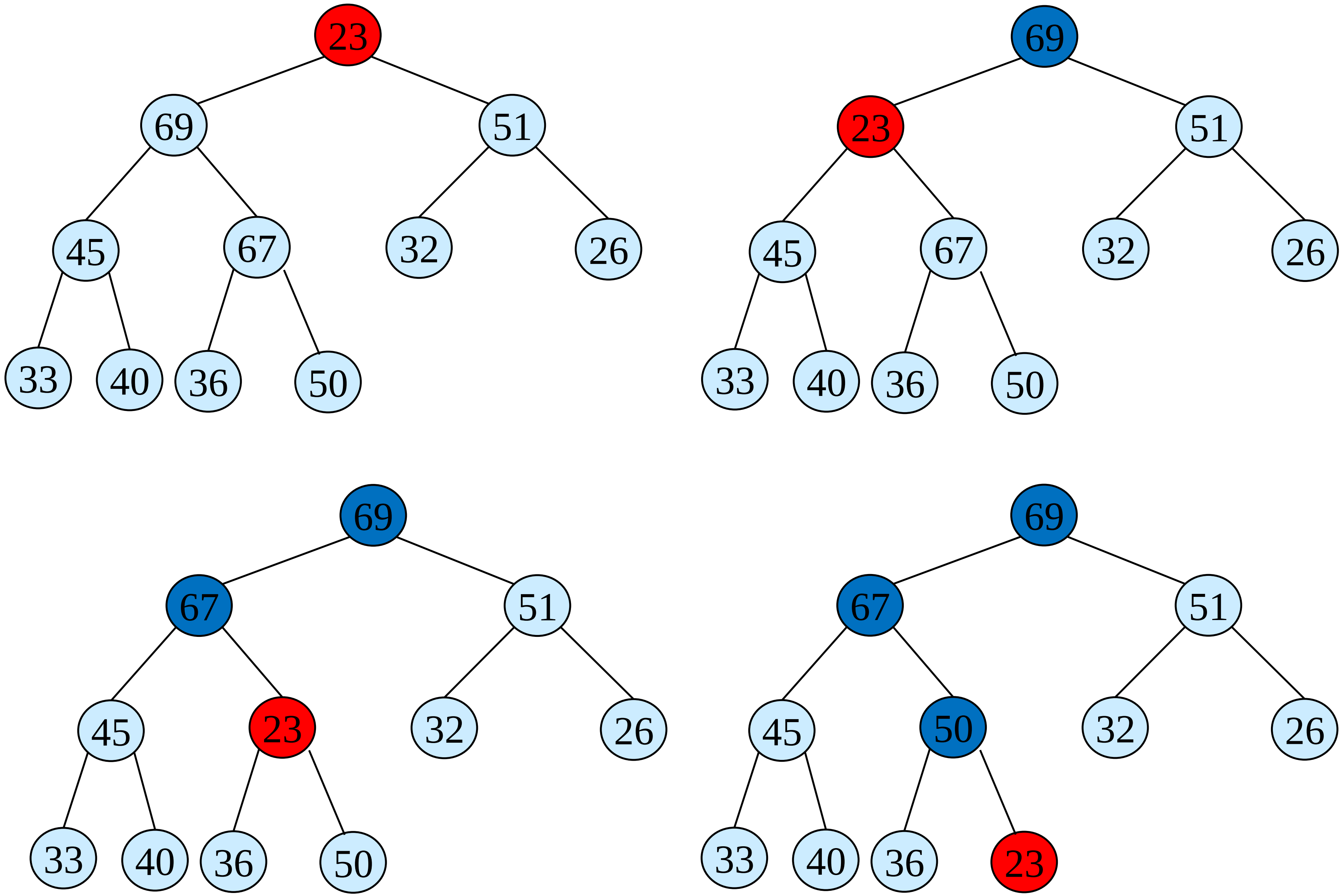
3. The following is a heap, but not a max heap. Show the whole process to transform it to a max heap by using *MaxHeapify*.

(10)




```
MaxHeapify(A, i)
l ← left(i)
r ← right(i)
if l ≤ heap-size[A] and A[l] > A[i]
   then largest ← l
   else largest ← i
if r ≤ heap-size[A] and A[r] > A[largest]
   then largest ← r
if largest ≠ i
   then exchange A[i] ↔ A[largest]
        MaxHeapify(A, largest)
```

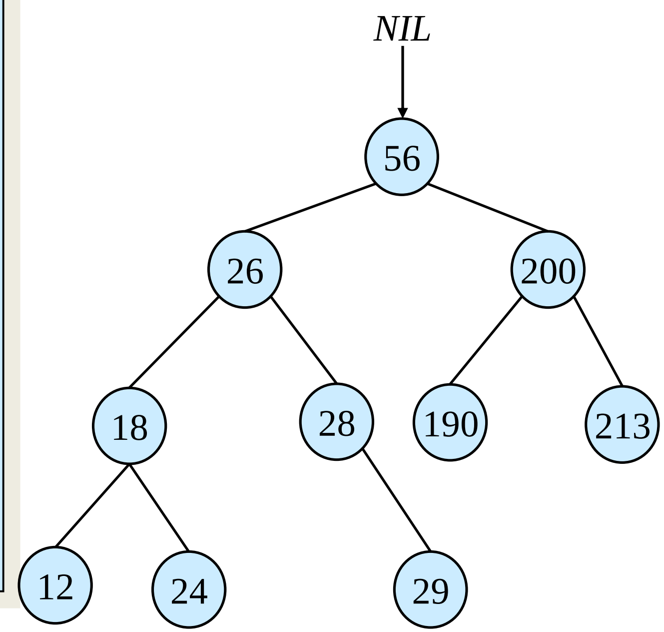

23
69
51
45
67
32
26
33
40
36
50

23
69
51
45
67
32
26
33
40
36
50

23
69
51
45
67
32
26
33
40
36
50

23
69
51
45
67
32
26
33
40
36
50


23
69 51
45 67 32 26
33 40 36 50
69
23 51
45 67 32 26
33 40 36 50
69
67 51
45 23 32 26
33 40 36 50
69
67 51
45 50 32 26
33 40 36 23

4. The predecessor of a node x in a binary search tree is a node y such that $key(y)$ is the largest key less than $key(x)$. Please give an algorithm to find the predecessor of a node x. (12)

Tree-Predecessor(x)

```
if left[x] ≠ NIL
    then return Tree-Maximum(left[x])
y ← p[x]
while y ≠ NIL and x = left[y]
do x ← y
    y ← p[y]
return y
```

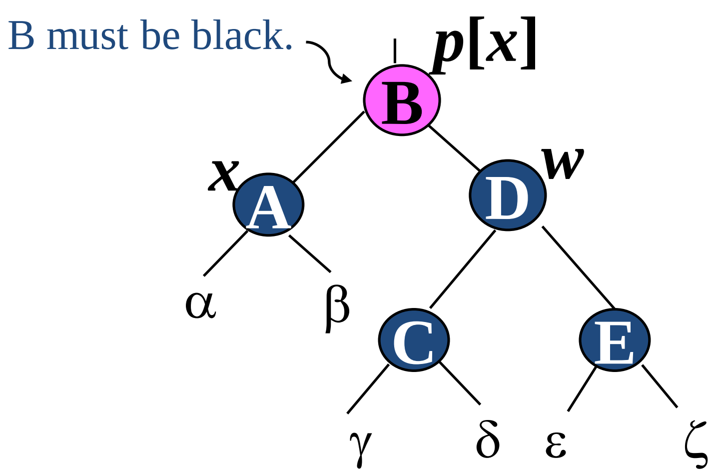


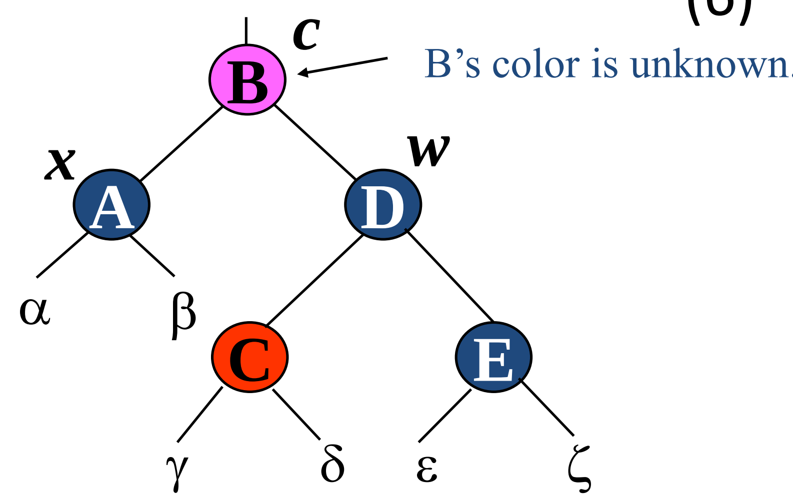
5. Regarding the algorithm to delete a node from a red-black tree, answer the following three questions:

a) Why the fix-up is not needed if the deleted node is colored red? (3)

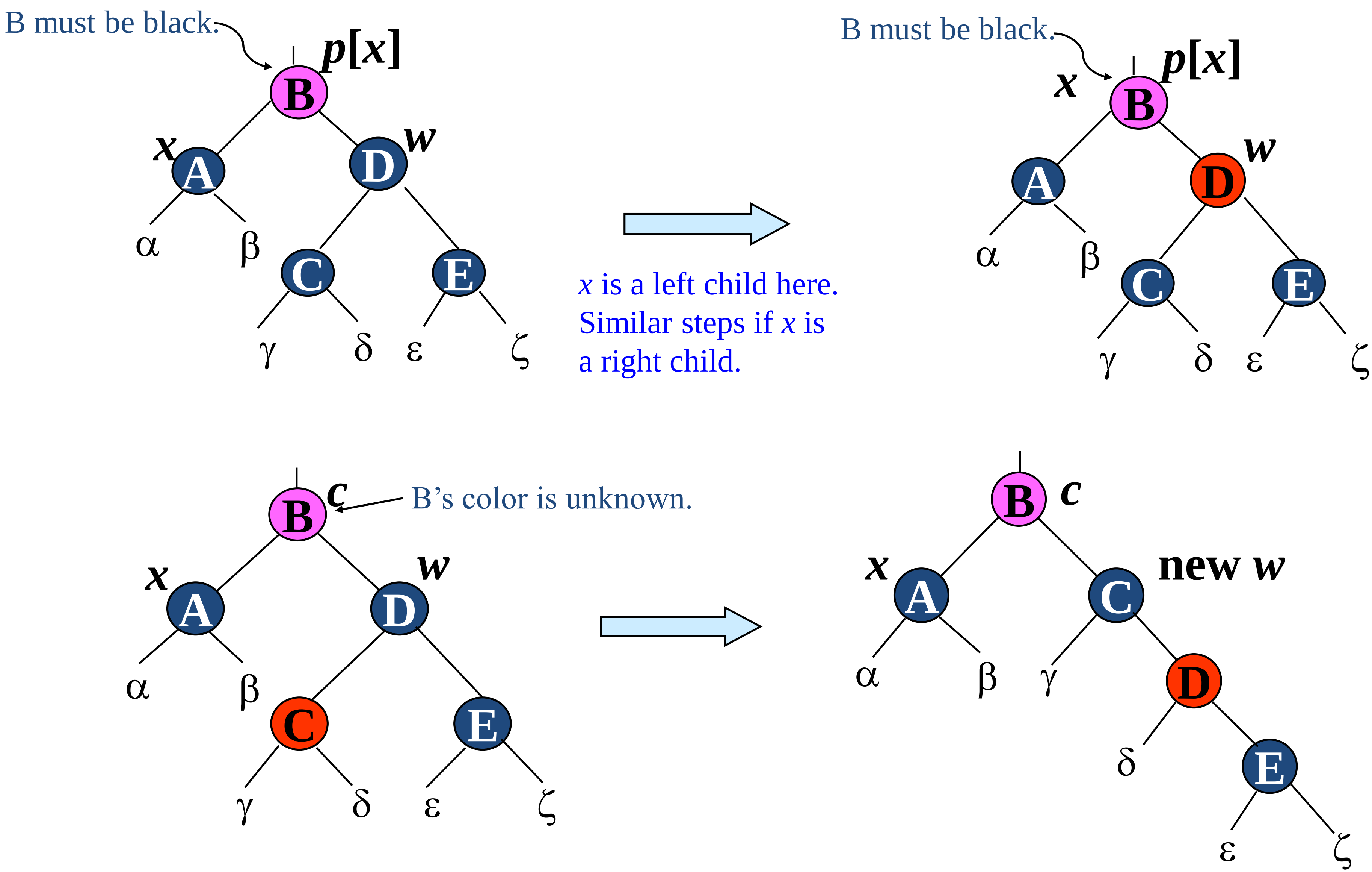
b) In Fig. 2(a), *x* is the child of the deleted node. If the right sibling *w* of *x* is black and both of its children are also black, how the tree will be changed? Also, show the reason. (6)

c) If the right sibling *w* of *x* is black, *w*'s left child is red, and *w*'s right child is black, as shown in Fig. 2(b), how the tree will be changed? (6)

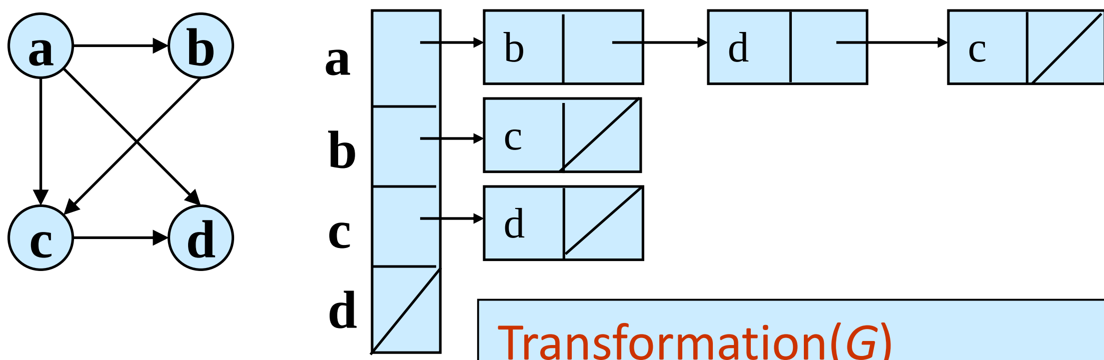




- If the deleted node is red, the red-black properties are still kept not violated.



6. The transpose of a DAG G is a graph G^T obtained by reversing the direction of each edge in G. Assume that G is stored in a linked list. Give an algorithm which is able to transform the linked list to another one representing G^T. (13)



<u>Transformation(*G*)</u>

```
Let A be an array containing all nodes
Copy A to B
For i = 1 to n {
    Scan the linked list C associated with A[i]
    for each node j in C add node i to the end of
    the linked list associated with B[j]
}
```

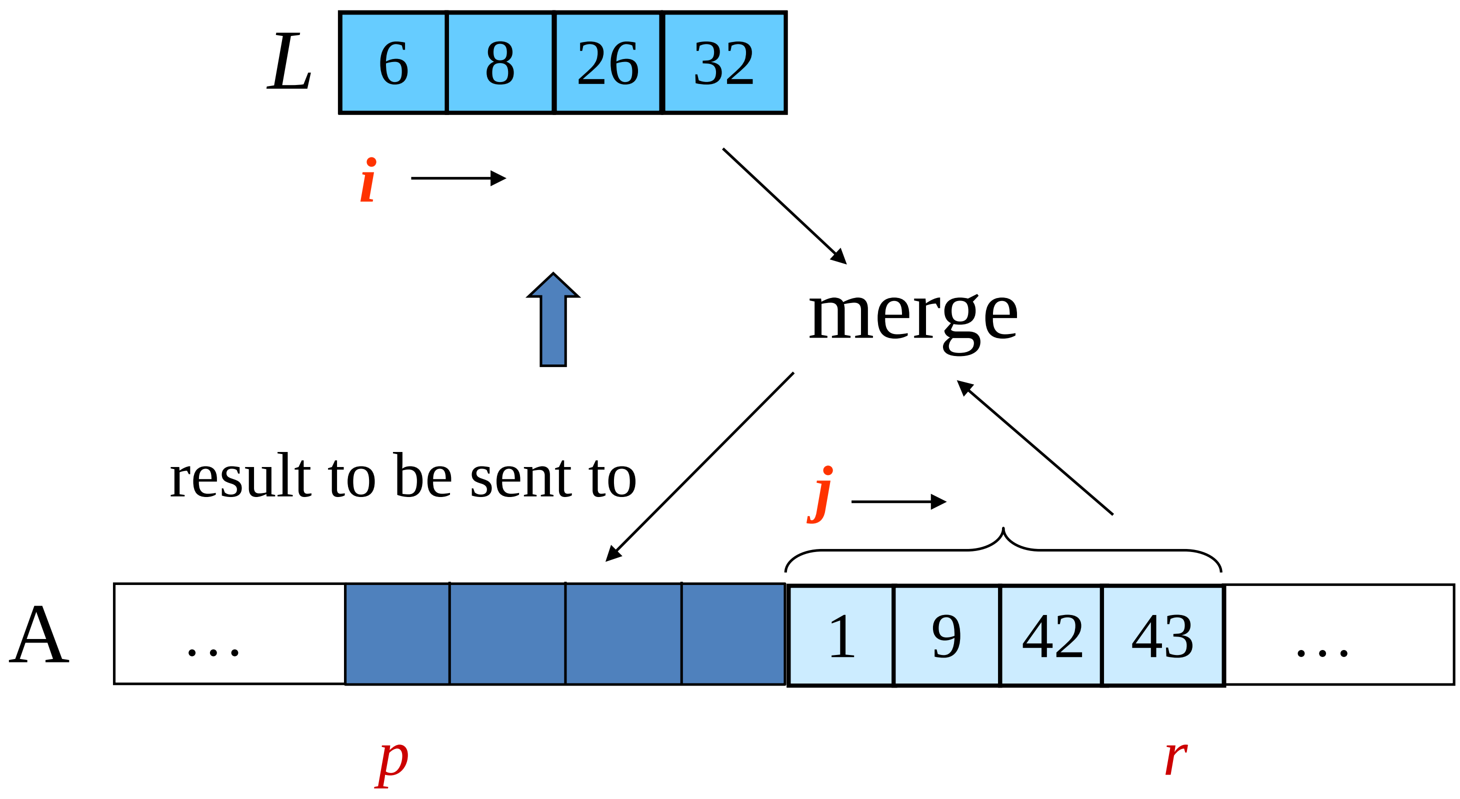
7. Give the improved merge procedure and show its correctness. (10)

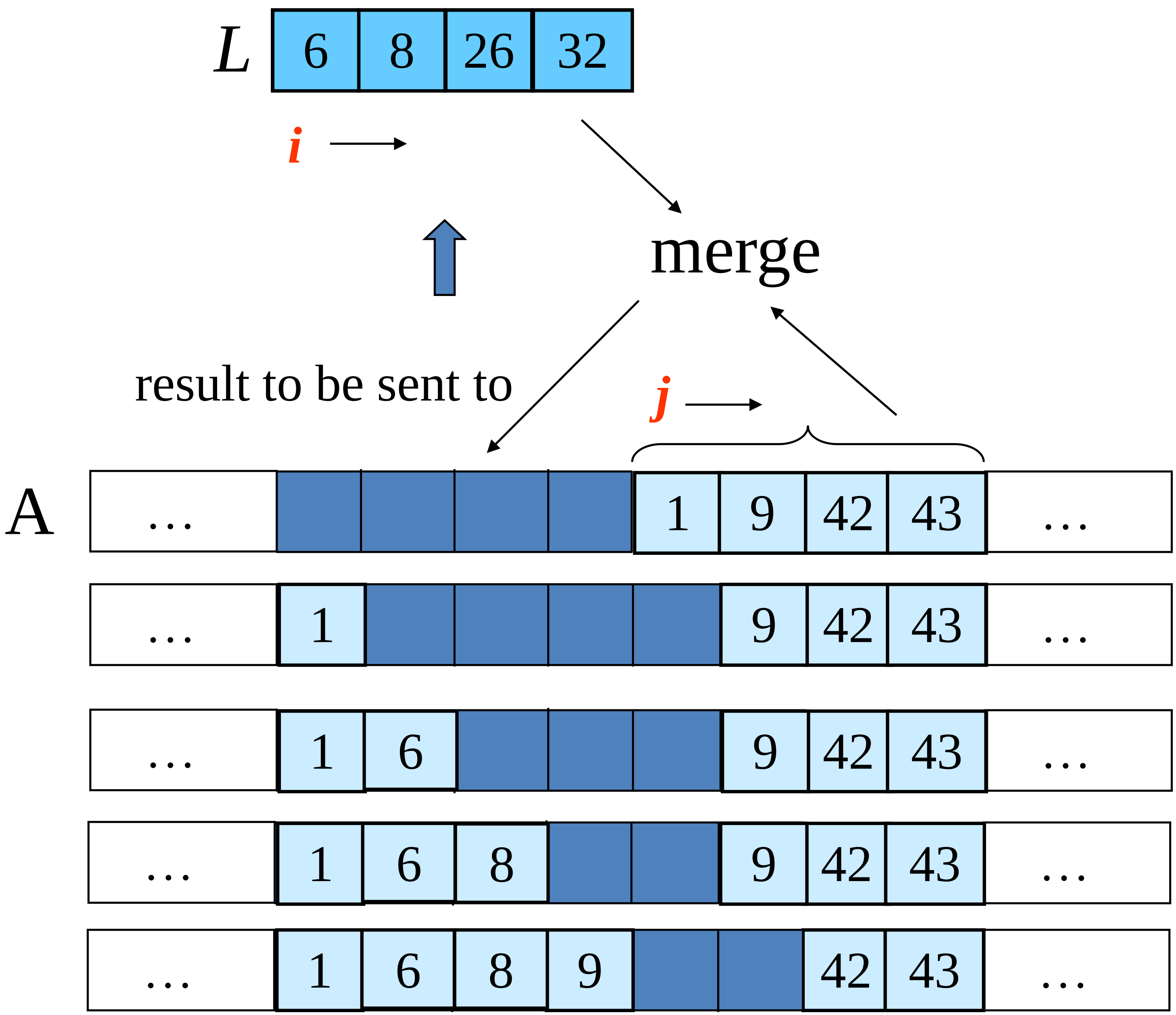
Algorithm: $mergeImpr\ (A, p, q, r\ )$

Input: Both $A[p\ ..\ q]$ and $A[q + 1\ ..\ r\ ]$ are sorted; but A as a whole is not sorted

Output : sorted A

$n_1 := q - p + 1$; $n_2 := r - q$; $k := p$;
let $L[1\ ..\ n_1]$ be a new array;
for $i = 1$ to n_1 **do**
    $L[i] := A[p + i - 1\ ]$
$i := p$; $j := q + 1$;
while $i \le n_1$ and $j \le n_2$ **do**
    if $L[i] \le A[j]$ **then** $\{A[k] := L\ [i]; i := i + 1;\}$
    else $\{A[k] := A[j]; j := j + 1;\}$
    $k := k + 1$;
if $j > n_2$ **then**
copy the remaining elements in L into $A[k\ ..\ r\ ]$;


L
6
8
26
32
i
merge
result to be sent to
j
A
...
1
9
42
43
...
p
r



L
6
8
26
32
i
merge
result to be sent to
j
A
…
1
9
42
43
…
…
1
9
42
43
…
…
1
6
9
42
43
…
…
1
6
8
9
42
43
…
…
1
6
8
9
42
43
…

Why does it work?

- **A is divided to sorted parts: $A[p .. q]$, $A[q + 1 .. r]$. $A[p .. q]$ will be copied to array L and $A[q + 1 .. r]$ stay in A.**

- **Denote by A' the sorted version of A. Denote by $A'(i, j)$ a prefix of A' which contains the first i elements from L and first j elements from $A[q + 1 .. r]$.**

- **Obviously, we can store $A'(i, j)$ in A itself since after the jth element (from $A[q + 1 .. r]$) has been inserted into A', the first $q - p + j + 1$ entries in A are empty and $q - p + 1 \geq i$ (thus, $q - p + j + 1 \geq i + j$).**