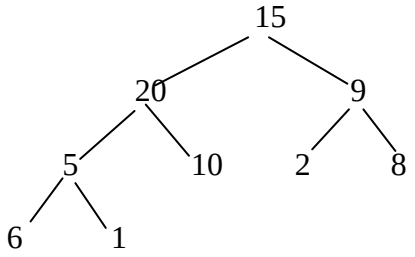
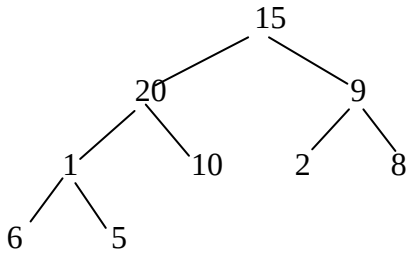
Φροντιστήριο 8 – Σκελετοί Λύσεων

1. Εφαρμογή της BuildHeap στον πίνακα [1 5 2 0 9 , 5, 1 0 ,2 , 8 , 6, 1] έχει τις εξής ενδιάμεσες καταστάσεις.

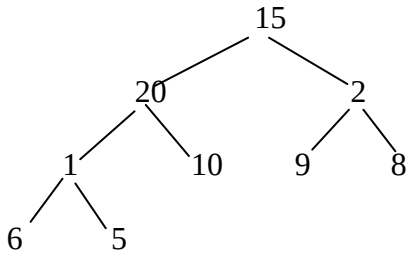
Αρχική Κατάσταση:



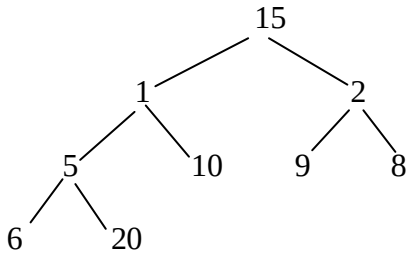
Μετά από εφαρμογή της PercoladeDown του στοιχείου στη θέση 4:



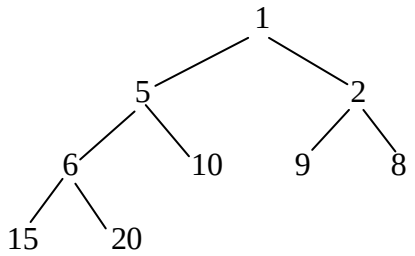
Μετά από εφαρμογή της PercoladeDown του στοιχείου στη θέση 3:



Μετά από εφαρμογή της PercoladeDown του στοιχείου στη θέση 2:



Μετά από εφαρμογή της PercoladeDown του στοιχείου στη θέση 1:



2. Υποθέτουμε ότι ο σωρός είναι υλοποιημένος με τη δομή της προηγούμενης άσκησης. Η ζητούμενη διαδικασία έχει ως εξής:

```
Remove(type x, heap *A){
     position = 1;
     while (A->array[position]!=x AND position < A->size)
          position++;
     if  (A->array[position] == x)
          A->array[position] = A->array[A->size];
          A->size--;
          if (A->array[position] > A->array[position/2]
               PercoladeDown(&A, position);
          else
               PercoladeUp(&A, position);

}
```

όπου, η `PercoladeUp(A, position)` ανεβάζει το στοιχείο A[position] όσο χρειάζεται μέσα στο σωρό A (παρόμοια με την `PercoladeDown(A, position)`).

Ο χρόνος εκτέλεσης της διαδικασίας είναι O(n + lg n), όπου n ο αριθμός των στοιχείων του σωρού, O(n) για την εύρεση του στοιχείου προς εξαγωγή, και O(lg n) για τον επανασχηματισμό του σωρού.

3. (a) *Υλοποίηση με πίνακα γειτνίασης*:
Υποθέτουμε ότι οι γράφοι υλοποιούνται με βάση την πιο κάτω δομή:

```
struct graph{
   int table[maxsize][maxsize];
   int size;
 }
```

Έστω γράφος G. Ο πιο κάτω αλγόριθμός υπολογίζει τον αντίστροφο γράφο του G, G΄.

```
inverse(struct graph G, struct graph G'){
 G'->size = G->size;
 for (u=0; u <= G -> size-1; u++ )
      for ( v=0; v <= G -> size -1; v++ )
       G'-> table[u][v] = G -> table[v][u];
}
```

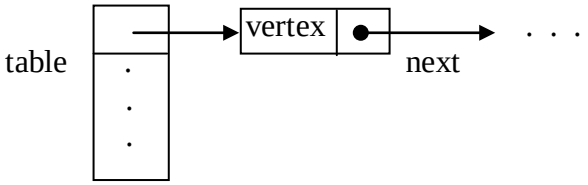
Ο χρόνος εκτέλεσης του αλγόριθμου είναι O(|V|²), όπου n είναι ο αριθμός κορυφών του γράφου.

(b) *Υλοποίηση με λίστα γειτνίασης*: Υποθέτουμε ότι οι γράφοι υλοποιούνται με βάση τη πιο κάτω δομή:

```
struct graph{
   struct node *table[maxsize];
   int size;
 }
```

```
struct node{
          int vertex;
     struct node *next;
}
```

όπου ο πίνακας table στη θέση i περιέχει συνδεδεμένη λίστα με όλες τις κορυφές προς τις οποίες υπάρχει ακμή από την κορυφή i. Υποθέτουμε πως κάθε κόμβος της συνδεδεμένης λίστας είναι τύπου node, όπου το node είναι εγγραφή με δύο πεδία: vertex, το οποίο δίνει το όνομα του κόμβου και next, το οποίο δείχνει τον επόμενο κόμβο μέσα στην λίστα.



Ο πιο κάτω αλγόριθμος υπολογίζει τον αντίστροφο γράφο του G, G΄.

```
inverse(struct graph G, struct graph G'){
 G'->size = G->size;

 for ( i = 0; i <= G -> size-1; i++ )
      G'->table[i] = NULL;

 for ( i = 0; i <= G -> size-1; i++ ){
      p = G->table[i];
      while (p!= NULL){
            w = malloc(sizeof(node));
            w -> vertex = i;
            w ->next = G'->table[p->vertex];
            G'->table[p->vertex] = w;
            p = p->next;
      }
}
```

Ο χρόνος εκτέλεσης του αλγόριθμου είναι O(|E|+|V|).