ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΔΕΝΤΡΑ

5.7 Ο ΑΤΔ Μέγιστος Σωρός

Εισαγωγή

Ο ΑΤΔ του μέγιστου σωρού (max heap) περιλαμβάνει τις ακόλουθες βασικές λειτουργίες:

- **δημιουργία** ενός κενού σωρού
- **έλεγχος αν ο σωρός είναι άδειος**
- **εισαγωγή στοιχείου** στο σωρό
- **διαγραφή του μεγαλύτερου στοιχείου** από το σωρό

Ορισμός του ΑΤΔ Μέγιστος Σωρός

Συλλογή στοιχείων δεδομένων:

Ένα πλήρες δυαδικό δέντρο με τα στοιχεία του οργανωμένα έτσι ώστε η τιμή σε κάθε κόμβο να είναι τουλάχιστο τόσο μεγάλη όσο εκείνη των παιδιών της.

Βασικές λειτουργίες:

Δημιουργία κενού σωρού (CreateMaxHeap):

Λειτουργία: Δημιουργεί ένα κενό σωρό.

Επιστρέφει: Ένα κενό σωρό.

Έλεγχος άδειου σωρού (EmptyHeap):

Δέχεται: Ένα σωρό.

Λειτουργία: Ελέγχει αν ο σωρός είναι άδειος.

Επιστρέφει: TRUE, αν ο σωρός είναι άδειος, FALSE διαφορετικά.

Ορισμός του ΑΤΔ Μέγιστος Σωρός

Εισαγωγή στοιχείου (InsertMaxHeap):

Δέχεται: Ένα σωρό και ένα στοιχείο δεδομένων.
Λειτουργία: Εισάγει το στοιχείο στο σωρό, αν ο σωρός δεν είναι γεμάτος.
Επιστρέφει: Τον τροποποιημένο σωρό.

Διαγραφή του μεγαλύτερου στοιχείου (DeleteMaxHeap):

Δέχεται: Ένα σωρό.
Λειτουργία: Ανακτά και διαγράφει το μεγαλύτερο στοιχείο του σωρού.
Επιστρέφει: Το μεγαλύτερο στοιχείο του σωρού και τον τροποποιημένο σωρό.

Η υλοποίηση θα γίνει στατικά (με πίνακα)

Έλεγχος γεμάτου σωρού (FullHeap):

Δέχεται: Ένα σωρό.
Λειτουργία: Ελέγχει αν ο σωρός είναι γεμάτος.
Επιστρέφει: TRUE, αν ο σωρός είναι γεμάτος, FALSE διαφορετικά.

Υλοποίηση δομής

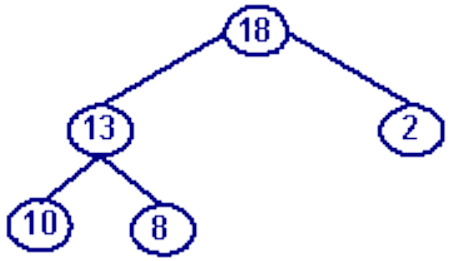
Μια τέτοια δομή μπορεί να υλοποιηθεί με μια **εγγραφή** (struct), όπως φαίνεται παρακάτω:

```
#define  MaxElements 10          /*μέγιστο πλήθος στοιχείων του σωρού*/
typedef int  HeapElementType;    /*ο τύπος των στοιχείων του σωρού*/
typedef  struct {
    HeapElementType key;
    // int Data;                 /*οποιοσδήποτε τύπος δεδομένων για τα
} HeapNode;                      παρελκόμενα στοιχεία του κόμβου*/

typedef struct {
      int Size;
      HeapNode Element[MaxElements+1];
} HeapType;
...
HeapType Heap;
```

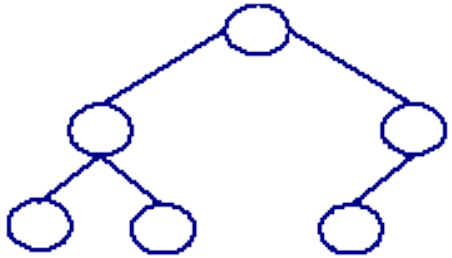
	Size	πίνακας Element[ ] κάθε στοιχείο του τύπου HeapElementType								
		0	1	2	3					MaxElements
Heap	3	?	8	23	16	?	?	?	…	?

Εισαγωγή στοιχείου στο σωρό

Στο σχήμα που ακολουθεί φαίνεται ένας μέγιστος σωρός με 5 στοιχεία:



Αν στο σωρό αυτό εισάγουμε ένα στοιχείο, τότε ο σωρός θα έχει τη δομή που φαίνεται παρακάτω, αφού ο μέγιστος σωρός είναι ένα πλήρες δυαδικό δέντρο.



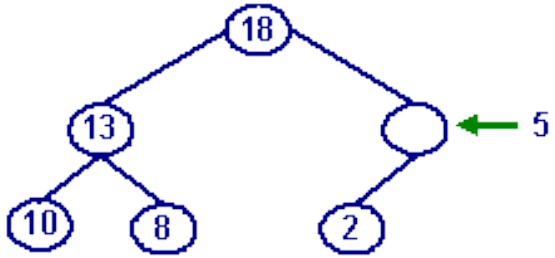
Εισαγωγή στοιχείου με **τιμή κλειδιού 1**:

- το στοιχείο μπορεί να εισαχθεί στο μέγιστο σωρό ως αριστερό παιδί του κόμβου με τιμή κλειδιού 2, οπότε θα προκύψει ο σωρός με τη δομή που φαίνεται παραπάνω

Εισαγωγή στοιχείου στο σωρό

Εισαγωγή στοιχείου με **τιμή κλειδιού 5**:

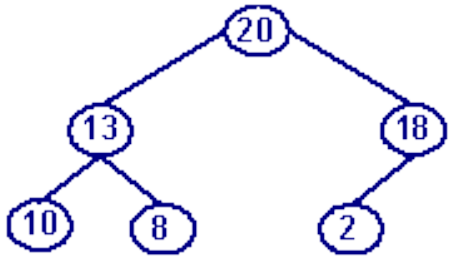
- στην περίπτωση αυτή το στοιχείο δεν μπορεί να εισαχθεί ως αριστερό παιδί του 2, γιατί τότε το πλήρες δυαδικό δέντρο που θα προκύψει δεν θα είναι ταυτόχρονα και μέγιστο δέντρο.
- Η επόμενη κίνησή μας είναι να μετακινήσουμε το 2 προς τα κάτω και συγκεκριμένα στη θέση του αριστερού παιδιού του (όπως φαίνεται στο Σχήμα) και να ελέγξουμε αν η εισαγωγή του 5 στην παλιά θέση του 2 οδηγεί ή όχι σε μέγιστο σωρό.
- Εφόσον, η τιμή κλειδιού του γονέα, η οποία είναι το 18, είναι τουλάχιστον τόσο μεγάλη όσο η τιμή του κλειδιού (5) του στοιχείου που εισάγεται, το στοιχείο μπορεί να εισαχθεί στη συγκεκριμένη θέση.



Εισαγωγή στοιχείου στο σωρό

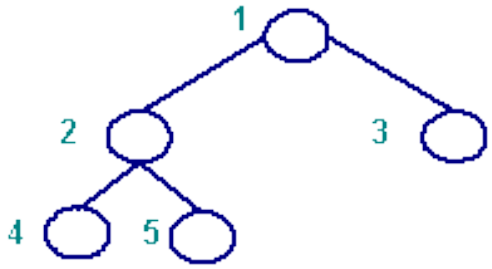
Εισαγωγή στοιχείου με **τιμή κλειδιού 20**:

- αρχικά το 2 μετακινείται στη θέση του αριστερού παιδιού του, όπως φαίνεται στο Σχήμα.
- Στη συνέχεια ελέγχουμε αν το 20 μπορεί να εισαχθεί στην παλιά θέση του 2.
- Είναι προφανές ότι το 20 δεν μπορεί να εισαχθεί στη συγκεκριμένη θέση, αφού ο γονέας του κόμβου που βρίσκεται στη θέση αυτή έχει τιμή 18, η οποία είναι μικρότερη του 20.
- Συνεπώς, το 18 μετακινείται στη θέση του δεξιού παιδιού του και το νέο στοιχείο με τιμή κλειδιού 20 εισάγεται στη ρίζα του σωρού.

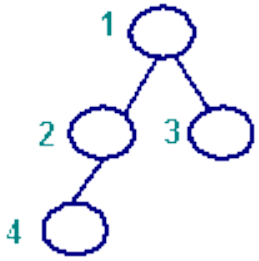


Διαγραφή στοιχείου από μέγιστο σωρό

Στο παρακάτω Σχήμα φαίνεται ένας μέγιστος σωρός με 5 στοιχεία:



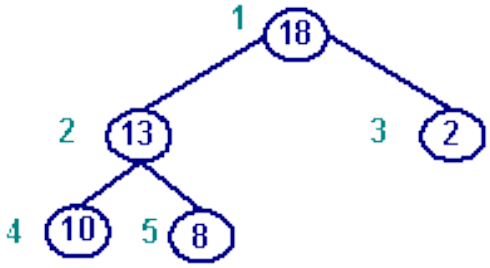
Αν από το σωρό αυτό διαγράψουμε το μεγαλύτερο στοιχείο, τότε ο σωρός θα έχει τη δομή που φαίνεται παρακάτω, αφού ο μέγιστος σωρός είναι ένα πλήρες δυαδικό δέντρο:



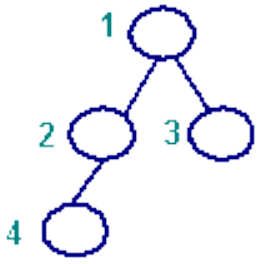
Διαγραφή στοιχείου από μέγιστο σωρό

Διαγραφή του μεγαλύτερου στοιχείου με **τιμή κλειδιού 18**:

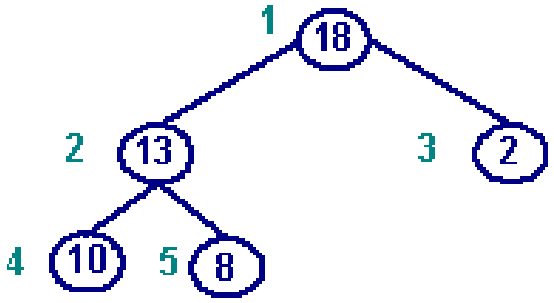
αν διαγράψουμε από τον μέγιστο σωρό που φαίνεται στο Σχήμα



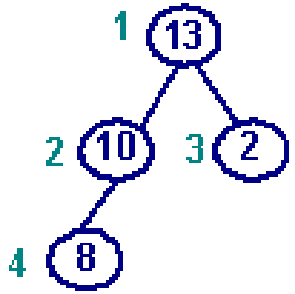
το μεγαλύτερο στοιχείο με τιμή κλειδιού 18 τότε ο σωρός που θα προκύψει θα πρέπει να έχει τη δομή που φαίνεται στο Σχήμα



Διαγραφή στοιχείου από μέγιστο σωρό



- Η πρώτη μας κίνηση είναι να διαγράψουμε τον κόμβο (με τιμή κλειδιού 8) που βρίσκεται στη θέση 5 του σωρού και να τον εισάγουμε στη ρίζα, η οποία είναι άδεια μετά τη διαγραφή του κόμβου με τιμή κλειδιού 18.
- Στη συνέχεια ελέγχουμε αν το πλήρες δυαδικό δέντρο που προκύπτει είναι ταυτόχρονα και μέγιστο δέντρο. Στο συγκεκριμένο παράδειγμα, αυτό δεν συμβαίνει, αφού το στοιχείο της ρίζας με τιμή κλειδιού 8 δεν είναι μεγαλύτερο από τα παιδιά της που έχουν τιμές κλειδιού 13 και 2.
- Άρα, το στοιχείο με τιμή κλειδιού 13 που βρίσκεται στη θέση 2 μετακινείται προς τα πάνω και συγκεκριμένα στη ρίζα, ενώ το στοιχείο της ρίζας εισάγεται στον κενό πλέον κόμβο της 2ης θέσης.
- Η επόμενη κίνησή μας είναι να ελέγξουμε αν το πλήρες δυαδικό δέντρο που προκύπτει είναι ταυτόχρονα και μέγιστο δέντρο. Αυτό δεν συμβαίνει, αφού το στοιχείο στη θέση 2 με τιμή κλειδιού 8 δεν είναι μεγαλύτερο από το παιδί του που έχει τιμή κλειδιού 10.
- Άρα, το στοιχείο με τιμή κλειδιού 10 που βρίσκεται στη θέση 4 μετακινείται προς τα πάνω και συγκεκριμένα στη θέση 2, ενώ το στοιχείο 8 της θέση 2 μετακινείται στον κόμβο της 4ης θέσης. Το πλήρες δυαδικό δένδρο που προκύπτει είναι και μέγιστο.



Πακέτο για τον ΑΤΔ Σωρός

```
//filename : HeapADT.h

#define  MaxElements 10          //το μέγιστο πλήθος των στοιχείων του σωρού

typedef int  HeapElementType;    //ο τύπος δεδομένων κάθε στοιχείου του σωρού
typedef  struct {
    HeapElementType key;
    //int Data;
} HeapNode;

typedef struct {
       int Size;
       HeapNode Element[MaxElements+1];
} HeapType;

typedef enum {
   FALSE, TRUE
} boolean;
```

Πακέτο για τον ΑΤΔ Σωρός

```
void CreateMaxHeap(HeapType *Heap);

boolean FullHeap(HeapType Heap);

boolean EmptyHeap(HeapType Heap);

void InsertMaxHeap(HeapType *Heap, HeapNode Item);

void DeleteMaxHeap(HeapType *Heap, HeapNode *Item);
```

Πακέτο για τον ΑΤΔ Σωρός

```
void CreateMaxHeap(HeapType *Heap)
/*Λειτουργία:       Δημιουργεί ένα κενό σωρό.
  Επιστρέφει:      Ένα κενό σωρό.*/
{
        (*Heap).Size = 0
}
```

```
boolean FullHeap(HeapType  Heap)
/*Δέχεται:          Ένα σωρό.
  Λειτουργία:      Ελέγχει αν ο σωρός είναι γεμάτος.
  Επιστρέφει:      TRUE αν ο σωρός είναι γεμάτος, FALSE διαφορετικά.*/
{
        return (Heap.Size == MaxElements)
}
```

Πακέτο για τον ΑΤΔ Σωρός

```
boolean EmptyHeap(HeapType  Heap)
/*Δέχεται:          Ένα σωρό Heap.
  Λειτουργία:      Ελέγχει αν ο σωρός είναι κενός.
  Επιστρέφει:     TRUE αν ο σωρός είναι κενός, FALSE διαφορετικά.*/
{
        return (Heap.Size == 0)
}
```

Πακέτο για τον ΑΤΔ Σωρός

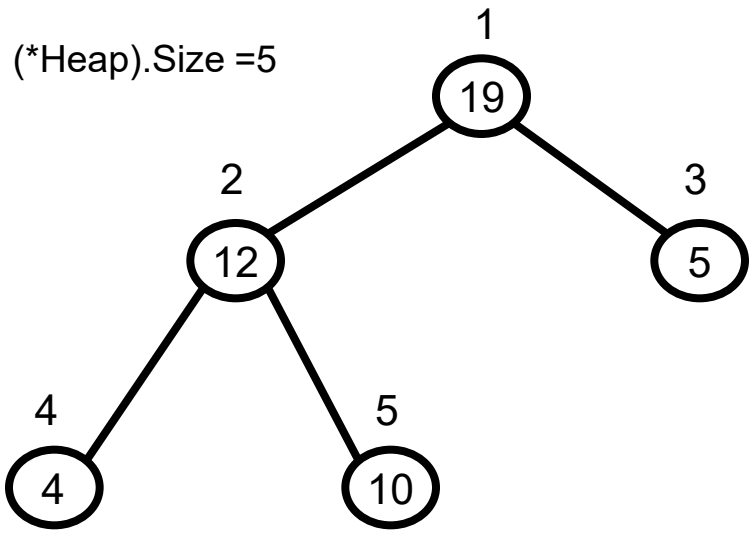
```
void InsertMaxHeap(HeapType *Heap, HeapNode Item)
/*Δέχεται:   Ένα σωρό Heap και ένα στοιχείο δεδομένου Item .
Λειτουργία: Εισάγει το στοιχείο Item στο σωρό, αν ο σωρός δεν είναι γεμάτος.
Επιστρέφει: Τον τροποποιημένο σωρό.
Έξοδος:     Μήνυμα γεμάτου σωρού αν ο σωρός είναι γεμάτος.*/

{
      int hole;

      if (!FullHeap(*Heap))
      {
          (*Heap).Size++;

          hole=(*Heap).Size;
          while (hole>1 && Item.key > Heap->Element[hole/2].key)
          {
                    (*Heap).Element[hole]=(*Heap).Element[hole/2];
                    hole=hole/2;
          }
          (*Heap).Element[hole]=Item;
  }
```

Παράδειγμα εισαγωγής

```
void InsertMaxHeap(HeapType *Heap, HeapNode Item)
{
    int hole;
    if (!FullHeap(*Heap)) {
        (*Heap).Size++;
        hole=(*Heap).Size;
        while (hole>1 && Item.key > Heap->Element[hole/2].key)
        {
                (*Heap).Element[hole]=(*Heap).Element[hole/2];
                hole=hole/2;
        }
        (*Heap).Element[hole]=Item;
}
```

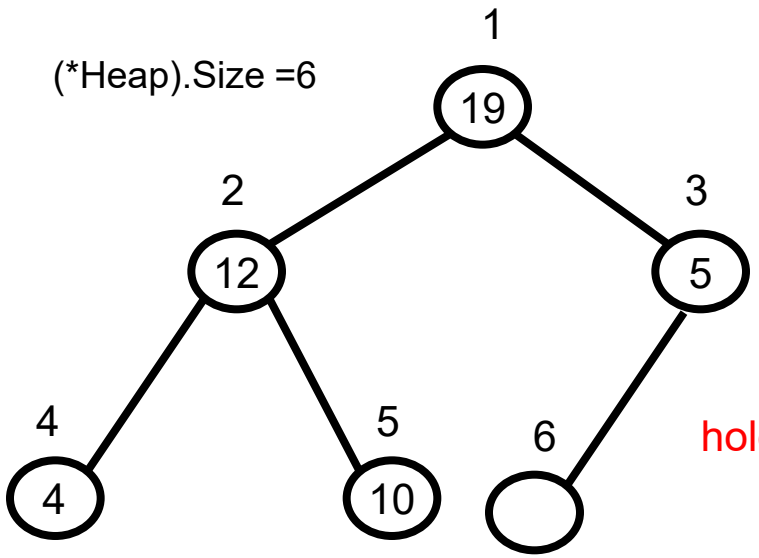
Item.key =20

(*Heap).Size =5



1	2	3	4	5
19	12	5	4	10

(*Heap).Size =6



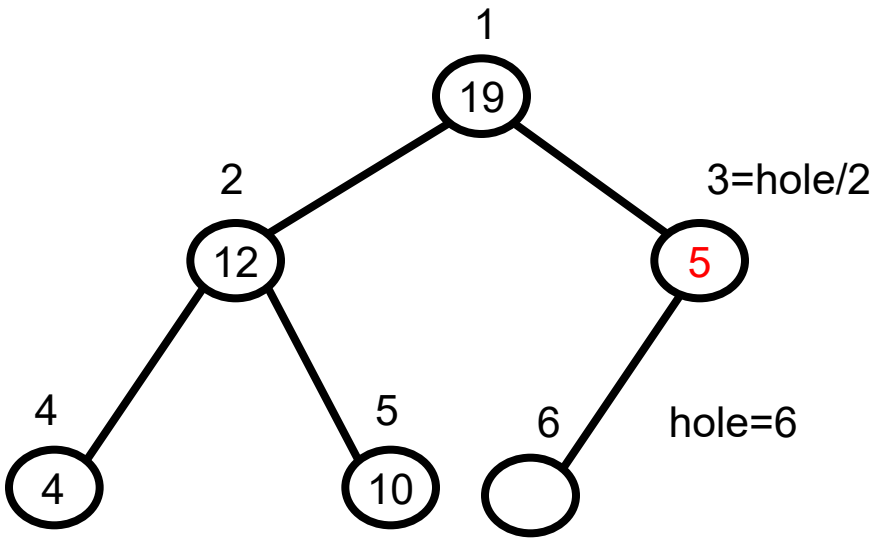
hole=6

1	2	3	4	5	6
19	12	5	4	10	

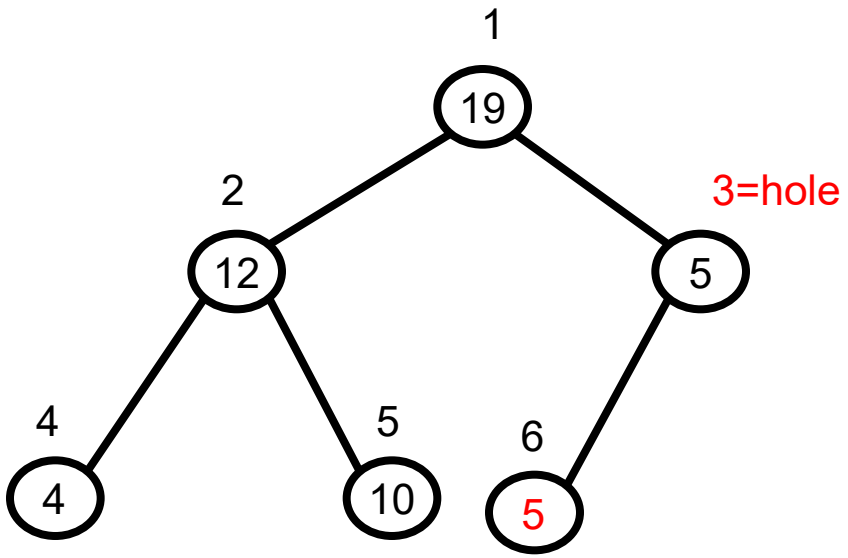
Παράδειγμα εισαγωγής

```
void InsertMaxHeap(HeapType *Heap, HeapNode Item)
{
    int hole;
    if (!FullHeap(*Heap)) {
        (*Heap).Size++;
        hole=(*Heap).Size;
        while (hole>1 && Item.key > Heap->Element[hole/2].key)
        {
                (*Heap).Element[hole]=(*Heap).Element[hole/2];
                hole=hole/2;
        }
        (*Heap).Element[hole]=Item;
  }
```

Item.key =20



1	2	3	4	5	6
19	12	5	4	10	

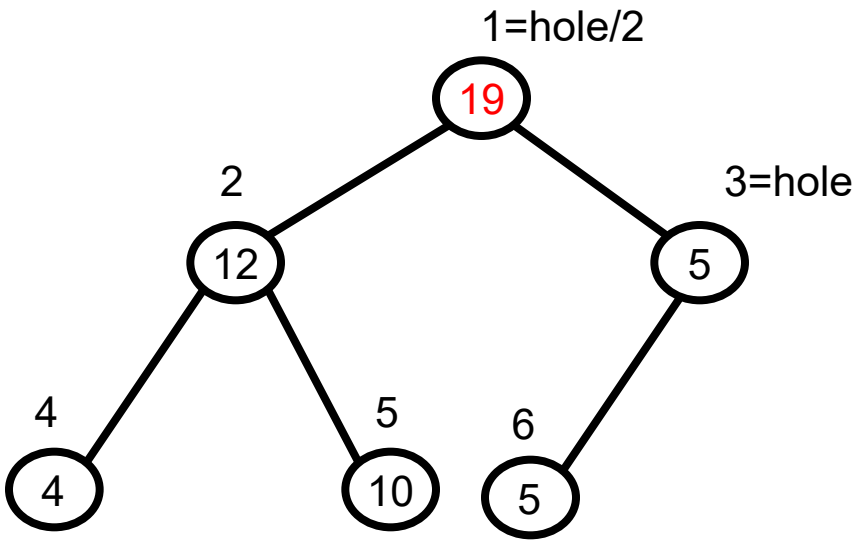


1	2	3	4	5	6
19	12	5	4	10	5

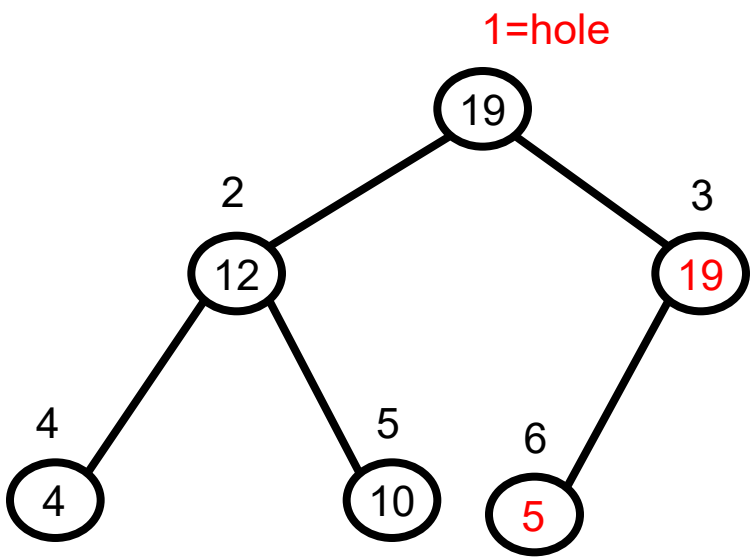
Παράδειγμα εισαγωγής

```
void InsertMaxHeap(HeapType *Heap, HeapNode Item)
{
    int hole;
    if (!FullHeap(*Heap)) {
        (*Heap).Size++;
        hole=(*Heap).Size;
        while (hole>1 && Item.key > Heap->Element[hole/2].key)
        {
                (*Heap).Element[hole]=(*Heap).Element[hole/2];
                hole=hole/2;
        }
        (*Heap).Element[hole]=Item;
  }
```

Item.key =20



1	2	3	4	5	6
19	12	5	4	10	5



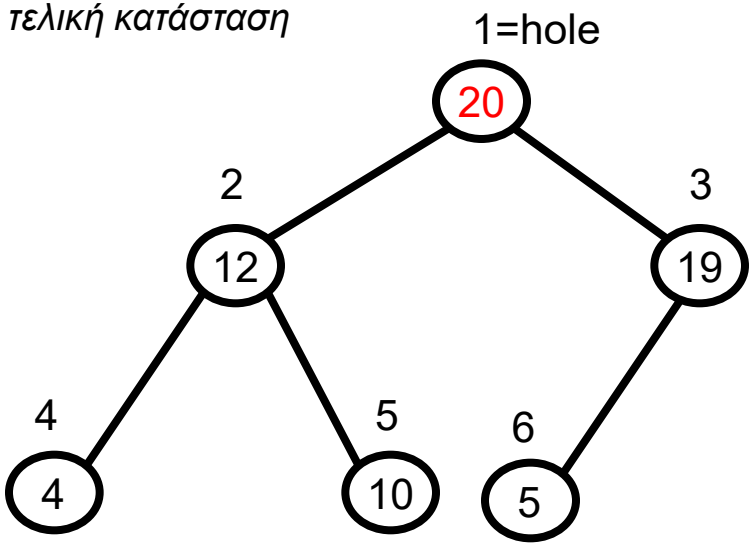
1	2	3	4	5	6
19	12	19	4	10	5

Παράδειγμα εισαγωγής

```
void InsertMaxHeap(HeapType *Heap, HeapNode Item)
{
    int hole;
    if (!FullHeap(*Heap)) {
        (*Heap).Size++;
        hole=(*Heap).Size;
        while (hole>1 && Item.key > Heap->Element[hole/2].key)
        {
                (*Heap).Element[hole]=(*Heap).Element[hole/2];
                hole=hole/2;
        }
        (*Heap).Element[hole]=Item;
}
```

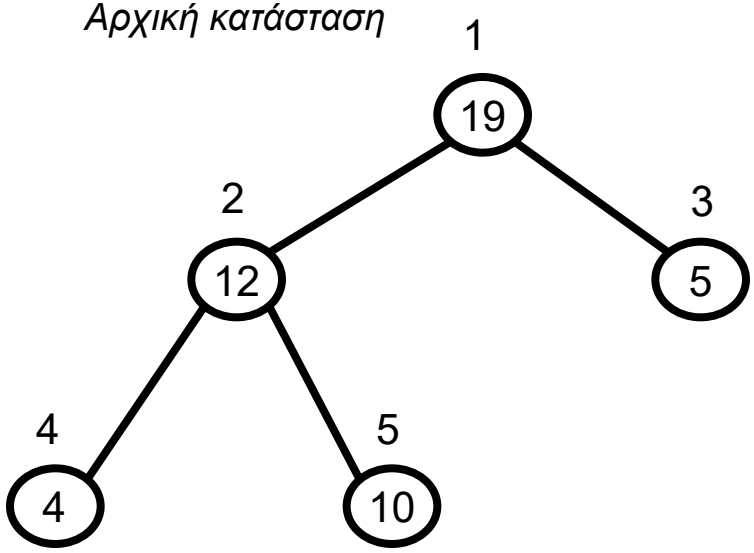
Item.key =20

τελική κατάσταση



1	2	3	4	5	6
20	12	19	4	10	5

Αρχική κατάσταση



1	2	3	4	5
19	12	5	4	10

Πακέτο για τον ΑΤΔ Σωρός

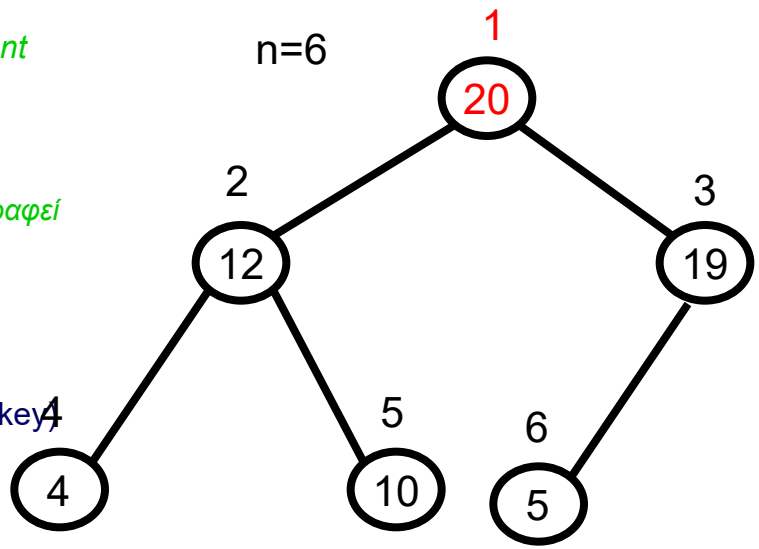
```
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item)
/*Δέχεται:        Ένα σωρό Heap.
  Λειτουργία:    Ανακτά και διαγράφει το μεγαλύτερο στοιχείο του σωρού.
  Επιστρέφει:    Το μεγαλύτερο στοιχείο Item του  σωρού (αυτό που
                 διαγράφεται) και τον τροποποιημένο σωρό.*/
{
        int parent, child;          // child δείχνει το μεγαλύτερο παιδί του κόμβου parent
         HeapNode last;             //last ο τελευταίος κόμβος
         boolean done;

        if (! EmptyHeap(*Heap)) {
              done = FALSE;
             *Item = (*Heap).Element[1];  //item το στοιχείο που θα διαγραφεί
              last = (*Heap).Element[(*Heap).Size];
             (*Heap).Size --;
             parent = 1;
             child = 2;
```

Πακέτο για τον ΑΤΔ Σωρός

```
        while (child<=(*Heap).Size  && !done) {
            if (child < (*Heap).Size)
                if (*Heap).Element[child].key < (*Heap).Element[child +1].key)
                    child ++;
            if (last.key >= (*Heap).Element[child].key)
                    done = TRUE;
             else
                {
                    //αντιγράφουμε την τιμή του παιδιού στον πατέρα του
                    (*Heap).Element[parent] = (*Heap).Element[child];
                    parent = child;      //συνεχίζουμε με τα παιδιά του
                    child = 2* child;
                }
        }
        (*Heap).Element[parent] = last;
    }
    else
        printf("Empty heap...\n");
}
```

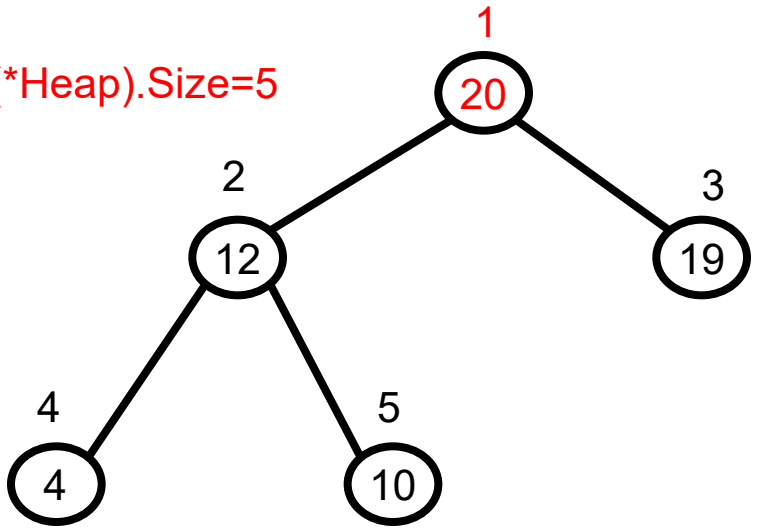
Παράδειγμα διαγραφής

```
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item) {
    int parent, child;          // child δείχνει το μεγαλύτερο παιδί του κόμβου parent
    HeapNode last;              //last ο τελευταίος κόμβος
     boolean done;
    if (! EmptyHeap(*Heap)) {
       done = FALSE; *Item = (*Heap).Element[1];  //item το στοιχείο που θα διαγραφεί
        last = (*Heap).Element[(*Heap).Size]; (*Heap).Size --;
      parent = 1;  child = 2;
      while (child<=(*Heap).Size  && !done) {
                  if (child < (*Heap).Size)
                    if ((*Heap).Element[child].key < (*Heap).Element[child +1].key)
                              child ++;
                              if (last.key >= (*Heap).Element[child].key)
                                        done = TRUE;
                         else {
                                    //αντιγράφουμε την τιμή του παιδιού στον πατέρα του
                                    (*Heap).Element[parent] = (*Heap).Element[child];
                                    parent = child;          //συνεχίζουμε με τα παιδιά του
                                    child = 2* child;
                                }
               }
               (*Heap).Element[parent] = last;
          }
          else  printf("Empty heap...\n");
}
```



1	2	3	4	5	6
20	12	19	4	10	5

(*Heap).Size=5



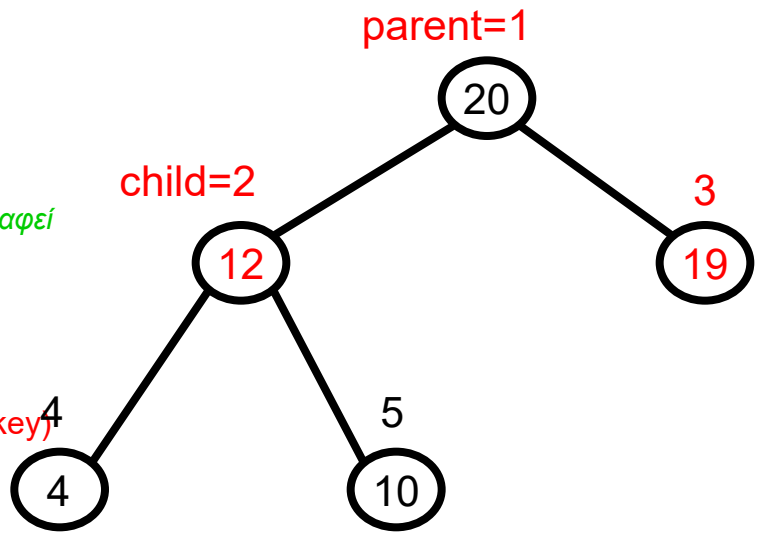
*Item.key=20

last.key=5

1	2	3	4	5
20	12	19	4	10

Παράδειγμα διαγραφής

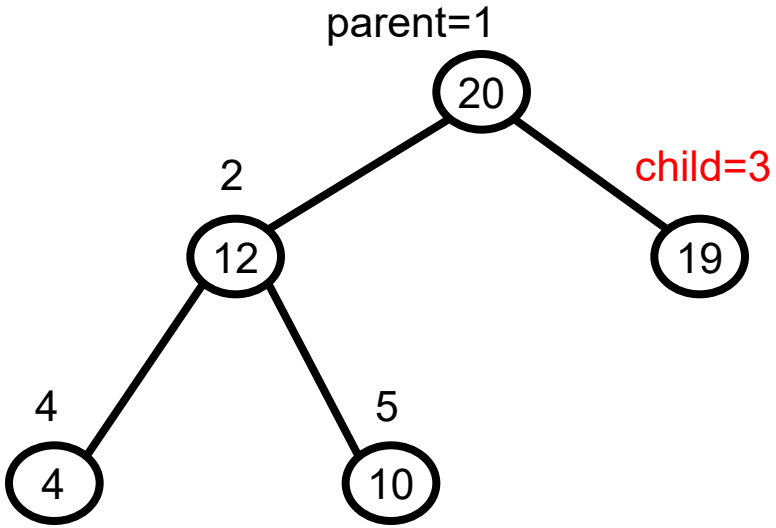
```
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item) {
    int parent, child;          // child δείχνει το μεγαλύτερο παιδί του κόμβου parent
    HeapNode last;              //last ο τελευταίος κόμβος
     boolean done;
    if (! EmptyHeap(*Heap)) {
       done = FALSE; *Item = (*Heap).Element[1];  //item το στοιχείο που θα διαγραφεί
        last = (*Heap).Element[(*Heap).Size]; (*Heap).Size --;
      parent = 1;  child = 2;
      while (child<=(*Heap).Size  && !done) {
                  if (child < (*Heap).Size)
                    if ((*Heap).Element[child].key < (*Heap).Element[child +1].key)
                              child ++;
                              if (last.key >= (*Heap).Element[child].key)
                                        done = TRUE;
                         else {
                                  //αντιγράφουμε την τιμή του παιδιού στον πατέρα του
                                    (*Heap).Element[parent] = (*Heap).Element[child];
                                    parent = child;        //συνεχίζουμε με τα παιδιά του
                                    child = 2* child;
                                }
                }
                (*Heap).Element[parent] = last;
            }
            else  printf("Empty heap...\n");
}
```



1	2	3	4	5
20	12	19	4	10

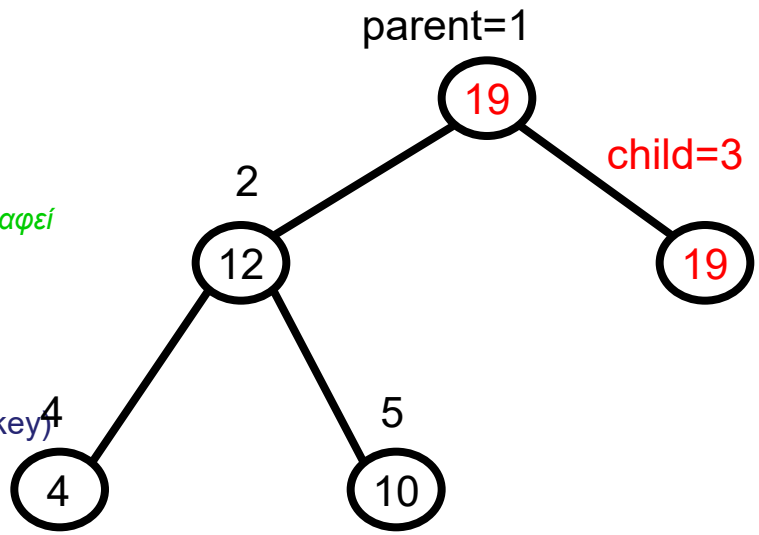
*Item.key=20 last.key=5

1	2	3	4	5
20	12	19	4	10



Παράδειγμα διαγραφής

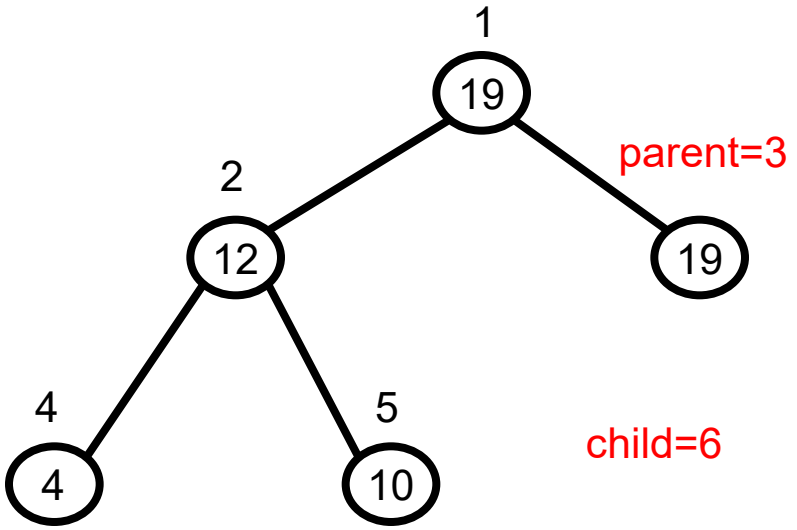
```
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item) {
    int parent, child;          // child δείχνει το μεγαλύτερο παιδί του κόμβου parent
    HeapNode last;              //last ο τελευταίος κόμβος
     boolean done;
    if (! EmptyHeap(*Heap)) {
       done = FALSE; *Item = (*Heap).Element[1];  //item το στοιχείο που θα διαγραφεί
        last = (*Heap).Element[(*Heap).Size]; (*Heap).Size --;
      parent = 1;  child = 2;
      while (child<=(*Heap).Size  && !done) {
                  if (child < (*Heap).Size)
                    if ((*Heap).Element[child].key < (*Heap).Element[child +1].key)
                              child ++;
                              if (last.key >= (*Heap).Element[child].key)
                                        done = TRUE;
                         else {
                                  //αντιγράφουμε την τιμή του παιδιού στον πατέρα του
                                    (*Heap).Element[parent] = (*Heap).Element[child];
                                    parent = child;          //συνεχίζουμε με τα παιδιά του
                                    child = 2* child;
                                }
               }
               (*Heap).Element[parent] = last;
          }
          else  printf("Empty heap...\n");
}
```



1	2	3	4	5
19	12	19	4	10

*Item.key=20 last.key=5

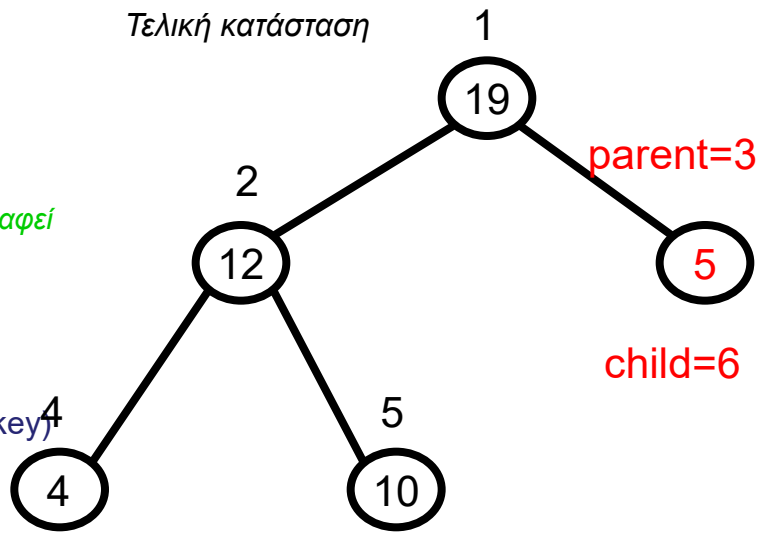
1	2	3	4	5
19	12	19	4	10



Παράδειγμα διαγραφής

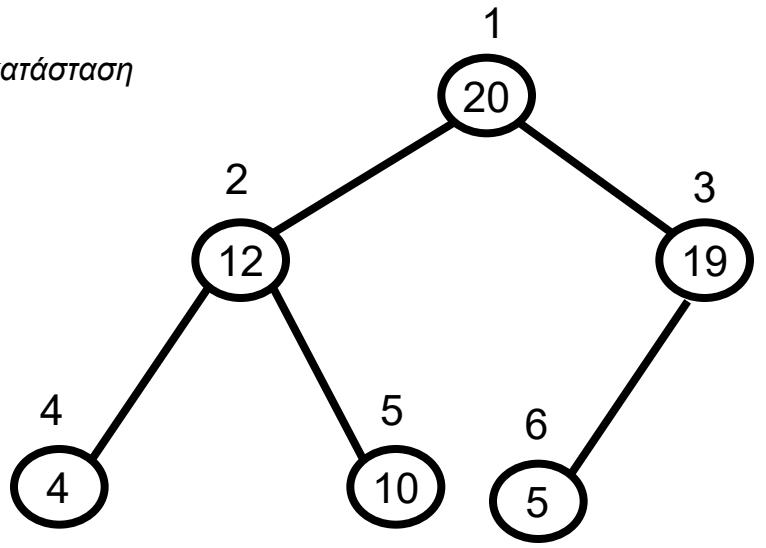
```
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item) {
    int parent, child;          // child δείχνει το μεγαλύτερο παιδί του κόμβου parent
    HeapNode last;              //last ο τελευταίος κόμβος
     boolean done;
    if (! EmptyHeap(*Heap)) {
       done = FALSE; *Item = (*Heap).Element[1];  //item το στοιχείο που θα διαγραφεί
        last = (*Heap).Element[(*Heap).Size]; (*Heap).Size --;
      parent = 1;  child = 2;
      while (child<=(*Heap).Size  && !done) {
                  if (child < (*Heap).Size)
                    if ((*Heap).Element[child].key < (*Heap).Element[child +1].key)
                              child ++;
                              if (last.key >= (*Heap).Element[child].key)
                                        done = TRUE;
                         else {
                                   //αντιγράφουμε την τιμή του παιδιού στον πατέρα του
                                    (*Heap).Element[parent] = (*Heap).Element[child];
                                    parent = child;          //συνεχίζουμε με τα παιδιά του
                                    child = 2* child;
                               }
              }
              (*Heap).Element[parent] = last;
        }
        else  printf("Empty heap...\n");
}
```

Τελική κατάσταση



1	2	3	4	5
19	12	5	4	10

Αρχική κατάσταση



1	2	3	4	5	6
20	12	19	4	10	5