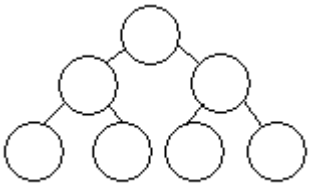
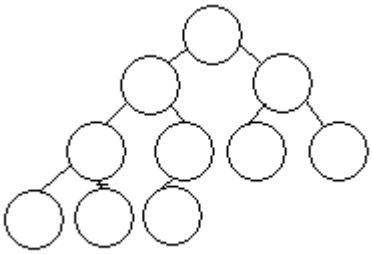
Estructura de datos heap o montículo (Arbol heap)

Una estructura heap es un árbol binario completo o casi completo y parcialmente ordenado.
(No confundir este concepto con el de la zona de memoria dinámica llamada heap)
Completo significa que tiene 'aspecto triangular', es decir que contiene un nodo en la raíz, dos en el nivel siguiente, y así sucesivamente teniendo todos los niveles ocupados totalmente con una cantidad de nodos que debe ser 2.
Casi completo significa que tiene todos los niveles menos el de las hojas 'saturados', ocupados con 1,2,4,8,16, etc. Nodos; en el nivel de las hojas puede no cumplirse esta condición, pero los nodos existentes deben estar ubicados 'a izquierda'.
Ejemplo:

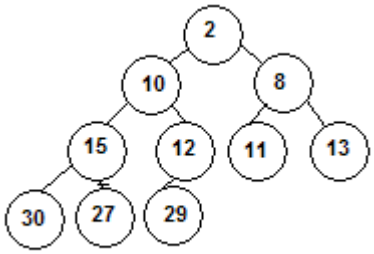


Arbol binario completo



Arbol binario casi completo

'Parcialmente ordenado' significa que se cumple, o bien que para cada nodo del árbol los valores de sus nodos hijos (si los tuviera) son ambos menores o iguales que el del nodo padre (y se llama árbol heap de mínimo), o bien que los valores de sus hijos son ambos mayores o iguales que los de su nodo padre (y se llama árbol heap de máximo). Entre los hijos no se exige ningún orden en particular.
Ejemplo:



Arbol Heap

La estructura que hemos representado arriba se almacena en un array, del siguiente modo:

2	10	8	15	12	11	13	30	27	29

Considerando que las posiciones del array son 0, 1, ...n-1, se verifica que las posiciones de los nodos hijos con respecto a su nodo padre cumplen :

Padre en posición **k** => Hijos en posiciones **2*k+1** y **2*k+2**

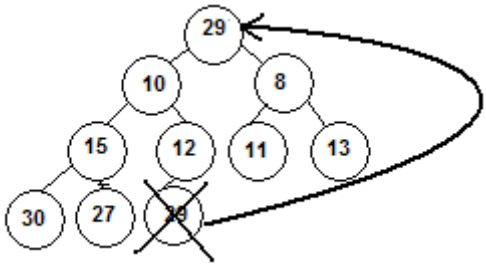
Análogamente, si el hijo está en posición **h**, su padre estará en posición **(h-1)/2 , considerando cociente entero.**

En estas estructuras las operaciones básicas son:

a) Extraer raíz:

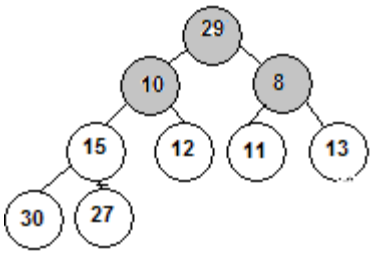
Esta operación se lleva a cabo de la siguiente forma:
Se remueve la raíz
Se reemplaza el nodo removido por la última hoja
Se restaura el heap o montículo, es decir se compara el valor de la nueva raíz con el de su hijo menor (si es un montículo ordenado de esta forma) y se realiza el eventual intercambio, y se prosigue comparando

hacia abajo el valor trasladado desde la raíz hasta llegar al nivel de las hojas o hasta ubicar el dato en su posición definitiva.
Ejemplo:
En el montículo del ejemplo anterior eliminamos la raíz y la reemplazamos



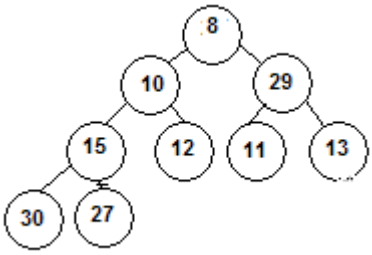
Se reemplaza 2 por 29
Se elimina el nodo hoja con 29

Ahora se analiza la situación de la nueva raíz con respecto a sus hijos:

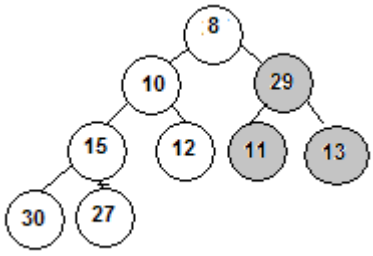


El menor de los hijos es 8, que es menor que 29, por lo cual se intercambia con él

Queda:

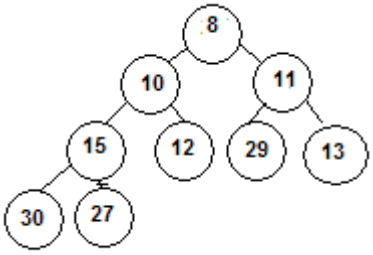


Ahora el problema se trasladó al nodo con 29 y sus nuevos hijos, 11 y 13



El menor de los hijos es 11, que es menor que 29.
Entonces, 29 se intercambia con 11

Queda finalmente el heap o montículo restaurado



Observar que para restaurar el montículo y ubicar el valor que se ha colocado en la raíz en su posición definitiva se lleva a cabo, a lo sumo un par de comparaciones y un intercambio hasta llegar al nivel de las hojas. Como el árbol está completo o casi completo, su altura es aproximadamente $\log_2 N$.
Así que, en el peor caso, para restaurar el montículo, el número de 'pasos' (entendiendo por paso el par de comparaciones más el eventual intercambio) es aproximadamente $\log_2 N$.

Tener en cuenta que estas etapas que se han mostrado con el diagrama de árbol, en realidad se llevan a cabo en el array en el cual está implementado el árbol heap.
Por lo tanto, considerando la estructura del array, se realiza lo siguiente:

2	10	8	15	12	11	13	30	27	29

Situación inicial del heap. N=10

29	10	8	15	12	11	13	30	27	~~29~~

Se reemplaza 2 por 29. N=9

29	10	8	15	12	11	13	30	27	~~29~~

Se compara 29 con sus hijos (la posición se determina como se indicó antes)

8	10	29	15	12	11	13	30	27	~~29~~

Se intercambia 29 con el menor de sus hijos.

8	10	29	15	12	11	13	30	27	~~29~~

Se analiza ahora la situación de 29 con respecto a sus 'nuevos' hijos'.

8	10	29	15	12	11	13	30	27	~~29~~

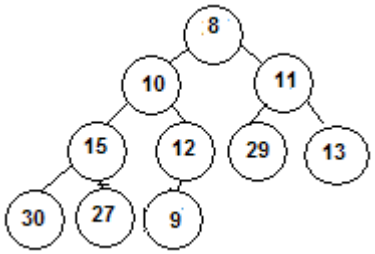
Se intercambia 29 con 11, el menor de sus hijos

8	10	11	15	12	29	13	30	27	~~29~~

El montículo ha sido restaurado. Ahora N=9

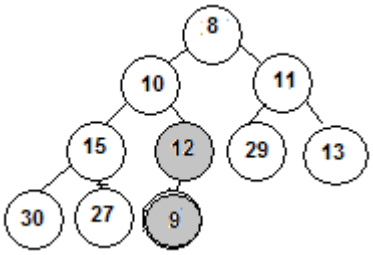
b) Agregar nuevo elemento :

El nuevo elemento se ubica como 'última hoja'. Luego se restaura el montículo analizando ternas 'hacia arriba' hasta ubicar el nuevo elemento en su posición definitiva:
Ejemplo:



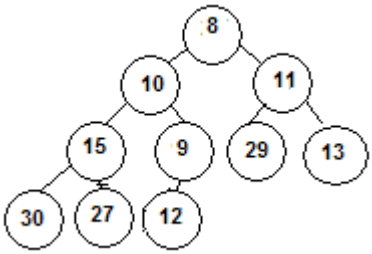
Se agrega el valor 9 al heap, como última hoja

Ahora el montículo debe ser restaurado:
Analizamos la situación existente entre 9 y su nodo padre, (sólo dos nodos porque 9 ha quedado como hijo izquierdo de 12)

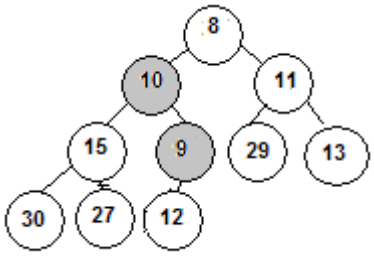


Como 9 es menor que 12, se intercambia con él

Queda:

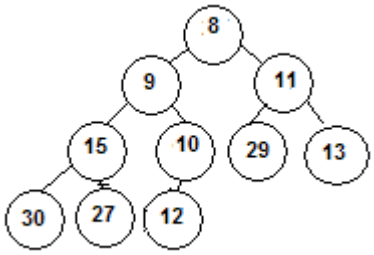


Ahora el análisis se debe realizar entre 9, y su padre, 10.

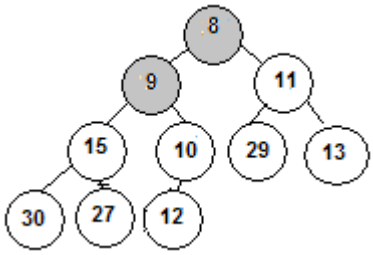


Como 9 es menor que 10, se intercambia con él

Resulta



Ahora se analiza la situación de 9, y su padre 8.



Como 9 supera a 8 el proceso termina.

Esto se realiza, como se indicó antes, en el array. Entonces, es.

8	10	11	15	12	29	13	30	27	9

Se almacena 9 como nueva hoja, es decir, se pone a continuación del último valor almacenado en el array. De esta forma, N, número de elementos del array, pasa de 9 a 10.

8	10	11	15	12	29	13	30	27	9

Se analiza la situación de 9 con respecto a su padre.

8	10	11	15	9	29	13	30	27	12

Como 9 es menor que 12, se intercambian los valores.

8	10	11	15	9	29	13	30	27	12

Ahora se analiza la situación de 9 con respecto a su padre 10

8	9	11	15	10	29	13	30	27	12

Como 9 es menor que 10, se intercambian los valores.

8	9	11	15	10	29	13	30	27	12

Se analiza la situación de 9 con respecto a 8; como se verifica la condición del heap, no se realiza intercambio, y el proceso concluye.

8	9	11	15	10	29	13	30	27	12

El heap o montículo ha sido restaurado.

Observar que, análogamente a lo sucedido en el caso de la extracción de la raíz, al agregar un nuevo elemento, se realiza una comparación y un eventual intercambio desde el nivel de las hojas hasta el nivel de la raíz. Es decir que , considerando que la altura del árbol es aproximadamente $\log_2 N$, se llevarán a cabo $\log_2 N$ pasos (cada paso consiste en una comparación y un eventual intercambio).

Por eso se indica que las operaciones de alta y extracción de raíz en una estructura heap o montículo tienen un costo temporal O(log N).

Uso de la estructura heap:

Esta estructura es muy eficiente para implementar TDA cola con prioridad y para ordenar arrays.

Cola con prioridad:

Es un TDA cuyas primitivas son:

Creación de la Cola con Prioridad
Destrucción de la Cola con Prioridad
Alta de un elemento
Extracción del mínimo elemento (el de mayor prioridad)

Como se puede apreciar, a un coste de logN se puede resolver mediante la implementación en heap, las primitivas de Alta y Extracción del Mínimo (corresponde a extracción de la raíz)

Aplicación al ordenamiento de arrays: el heapsort.

Este método de ordenamiento tiene 2 partes :

En la primera parte se convierte el array en un montículo, reubicando los elementos del mismo.
En la segunda parte, en sucesivas etapas intercambia el elemento de la raíz con la última hoja, decrementa la posición considerada última del array y recompone el montículo desde la primera posición del array (posición de la raíz) hasta la última considerada (recompone el montículo en una zona del array que va disminuyendo paulatinamente).
Al agotarse los elementos del array que se consideran parte del montículo, el proceso termina.
Observar que si se quiere ordenar el array de modo ascendente, el montículo debe verificar que el valor de cada nodo debe ser mayor que el de sus hijos, y lo contrario para un ordenamiento descendente.

Código del Heapsort:

```
void Heapsort( int a[], int n )
{
  int k, x;

  for (k = n/2; k >= 1; k--) Recomponer(a, k, n);
  while (n > 1)
    {      x = a[1]; a[1] = a[n]; a[n] = x;
           Recomponer(a, 1, --n);
    }
}

void Recomponer( int a[], int izqu, int der)
{
  int i, j, x;
  x = a[izqu];
  i = izqu; j = 2*i;
  if ((j < der) && (a[j] < a[j+1])) j++;
  while ((j <= der) && (x < a[j]))
 {
     a[i] = a[j];
     i = j; j = 2*j;
     if ((j < der) && (a[j] < a[j+1])) j++;
 }
  a[i] = x;
}
```

Ejemplo:

Este es el array original

3	2	1	9	7	6

Las siguientes son las etapas de la conformación del heap

3	2	6	9	7	1

3	9	6	2	7	1

9	3	6	2	7	1

9	7	6	2	3	1

Ahora ya se tiene un montículo formado. Se lleva a cabo la segunda parte.

1	7	6	2	3	9

7	1	6	2	3	9

7	3	6	2	1	9

1	3	6	2	7	9

6	3	1	2	7	9

2	3	1	6	7	9

3	2	1	6	7	9

1	2	3	6	7	9

2	1	3	6	7	9

1	2	1	6	7	9