Heap Sort vs Bubble Sort

Dimael Rivas
Fernando Guillen

Intro Sort

El Intro Sort es un algoritmo de ordenamiento híbrido, es decir que utiliza más de un algoritmo de ordenamiento. Los algoritmos que utiliza son: Quicksort, Heapsort y el Insertion sort.

Intro sort comienza con el quicksort y si la recursión va más allá de un parámetro establecido, cambia a Heapsort para evitar el peor caso de Quicksort. (Complejidad de tiempo).

(MaxDepth>logN)

También utiliza la ordenación por inserción cuando el número de elementos a ordenar es bastante menor.

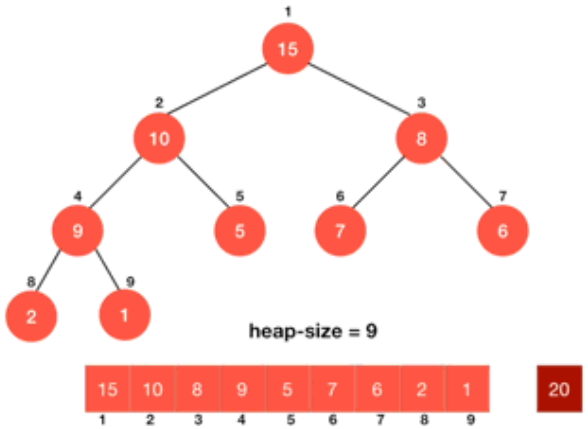
(N<16)

*El algoritmo C++ STL más popular: sort() utiliza Introsort.

Heap Sort

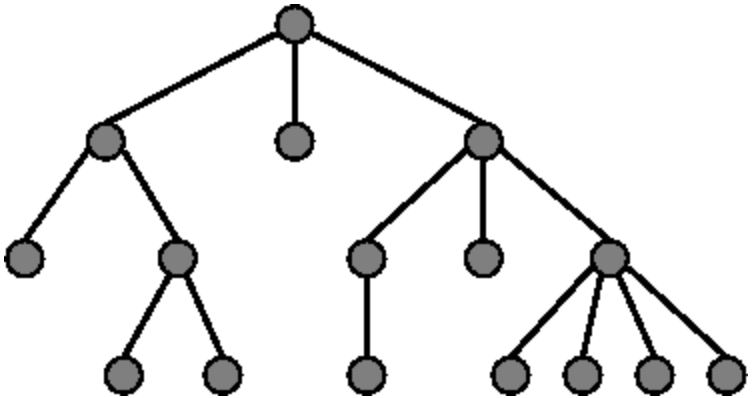
Conceptos básicos:

- Tree
- Heap



Tree

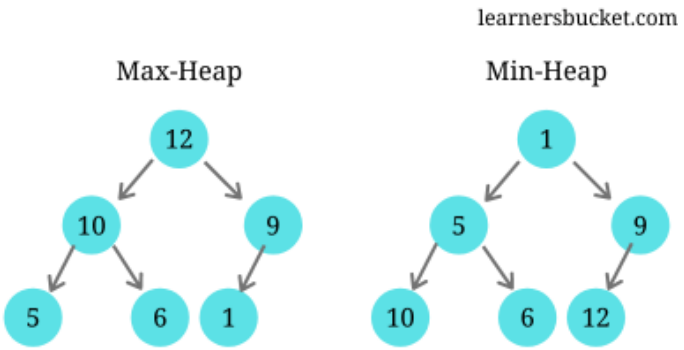
Definición simple: Un grafo conexo y sin ciclos



Heap

Restricciones:

Debe ser un árbol **casi completo**, y en el caso de hablar de un MAX-heap y MIN-heap, los hijos de cada nodo deben ser menores y mayores, respectivamente.



Características:

La estructura heap se puede recorrer realizando simples operaciones.

$$i_{left} = i_{parent} * 2$$

$$i_{right} = i_{parent} * 2 + 1$$

$$i_{parent} = i/2$$

Heap Sort

Para ordenar los elementos utilizamos el build_max_heap, el cual empieza por N/2, debido a que este siempre será el penúltimo nivel.

Y dado que podemos considerar a los nodos como MAX_HEAPS, podemos ejecutar el heapify para el sub árbol de arr[i].

Cabe resaltar que el heapify arregla la propiedad del MAX heap, cambiando el hijo mayor con el padre.

Limitaciones

Complejidad Computacional: O(nlog(n)) | En un analisis más detallado O(n)

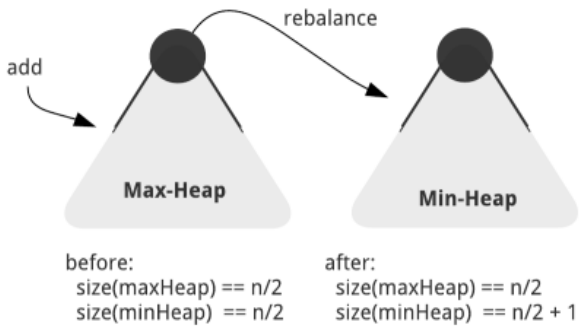
Complejidad espacial: O(1)

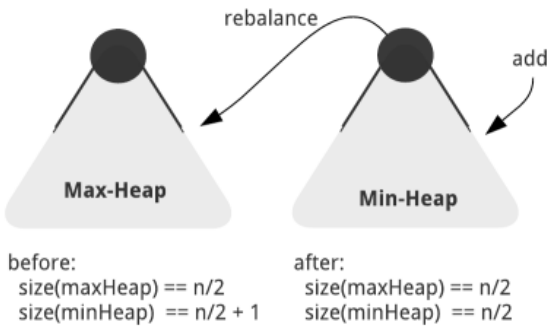
El heapsort a pesar de tener la misma complejidad que el mergesort o el quicksort. El heapsort lleva a cabo más comparaciones al momento de imprimir sus valores.

A pesar de esto, el heapsort puede ser utilizado en caso de evadir usar espacio en la memoria y tener una complejidad más baja que el bubble sort.

Real Life Example

Calcular la mediana de un arreglo en tiempo constante:





Código

```
// To heapify a subtree rooted with node i
// which is an index in arr[].
// n is size of heap
void heapify(int arr[], int N, int i)
{

    // Initialize largest as root
    int largest = i;

    // left = 2*i + 1
    int l = 2 * i + 1;

    // right = 2*i + 2
    int r = 2 * i + 2;

    // If left child is larger than root
    if (l < N && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest
    // so far
    if (r < N && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected
        // sub-tree
        heapify(arr, N, largest);
    }
}
```

```
// Main function to do heap sort
void heapSort(int arr[], int N)
{

    // Build heap (rearrange array)
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    // One by one extract an element
    // from heap
    for (int i = N - 1; i > 0; i--) {

        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}
```

Bubble Sort

El bubble sort es considerado como uno de los algoritmos de ordenamiento más sencillos. Este algoritmo se basa en recorrer la lista o arreglo comparando los elementos adyacentes con el fin de saber si se encuentran en el orden correcto, de lo contrario intercambiarlos de posición.

*Este algoritmo no es recomendado para una gran cantidad de datos, debido a que su complejidad de tiempo promedio y en el peor de los casos es bastante alta.

Show me the code

```
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
```

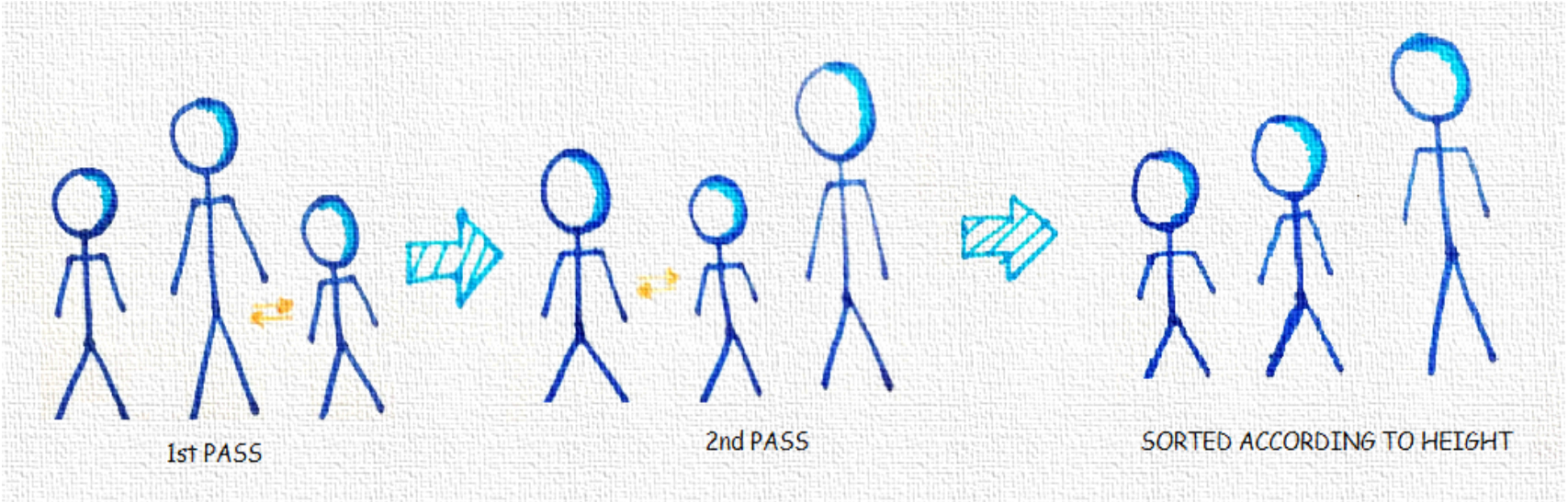
Complejidad

Worst and Average Case Time Complexity: $O(N^2)$. (Cuando la lista está en el orden inverso)

Best Case Time Complexity: O(N). (Cuando la lista ya se encuentra ordenada).

Auxiliary Space: O(1)

Real Life Example



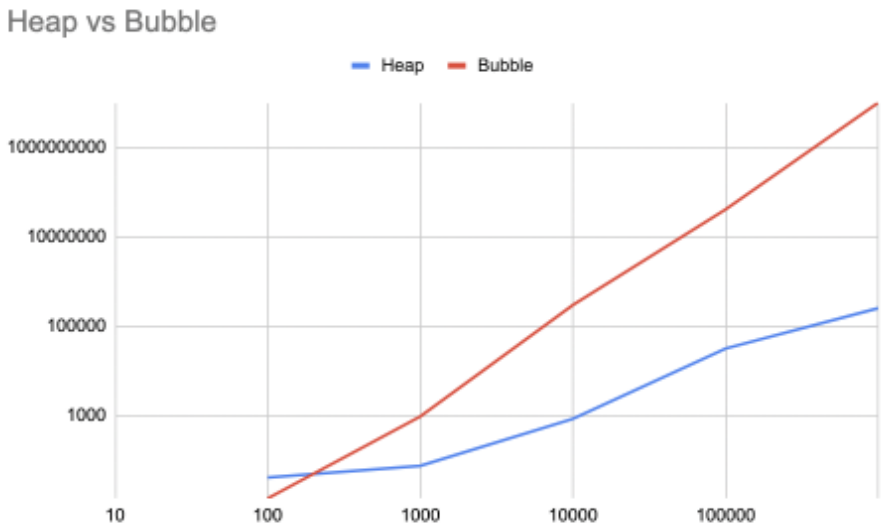
Comparación

Elements	Heap Sort	Bubble Sort
10	0 Microsegundos	0 Microsegundos
100	41 Microsegundos	14 Microsegundos
1000	75 Microsegundos	953 Microsegundos
10000	849 Microsegundos	296951 Microsegundos
100000	31631 Microsegundos	41109317 Microsegundos
1000000	251919 Microsegundos	Time limit exceed

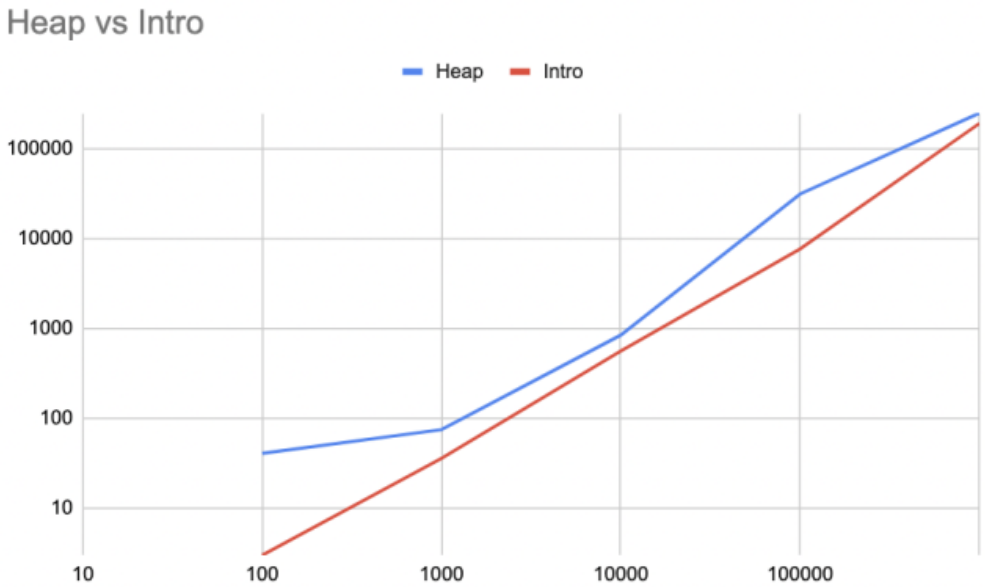
Comparación

Elements	Heap Sort	Intro Sort
10	0 Microsegundos	0 Microsegundos
100	41 Microsegundos	3 Microsegundos
1000	75 Microsegundos	36 Microsegundos
10000	849 Microsegundos	563 Microsegundos
100000	31631 Microsegundos	7725 Microsegundos
1000000	251919 Microsegundos	494390 Microsegundos

Who is the winner?



Who is the winner?



Conclusiones

El heapsort es muy conveniente cuando se requiere un algoritmo de ordenamiento O(nlog(n)) sin usar mucha memoria. Además que por las propiedades del heap se puede acceder al máximo o mínimo valor en O(1).

El Bubble Sort sirve como un algoritmo eficaz y fácil de implementar sin usar tanta memoria. Pero consume mucho tiempo.

El IntroSort es muy eficiente, pero muy tedioso de implementar y entender. (Sin embargo es uno de los mas utilizados debido a que está incluido en la librería STL y a que la función sort() lo utiliza.)

Preguntas

Cual es el motivo por el cual el Introsort es uno de los mejores algoritmos

Por que en el for del heap sort se empieza desde N/2?


GRACIAS POR SU ATENCIÓN
CUALQUIER DUDA CONSULTEN GOOGLE
memegen.es