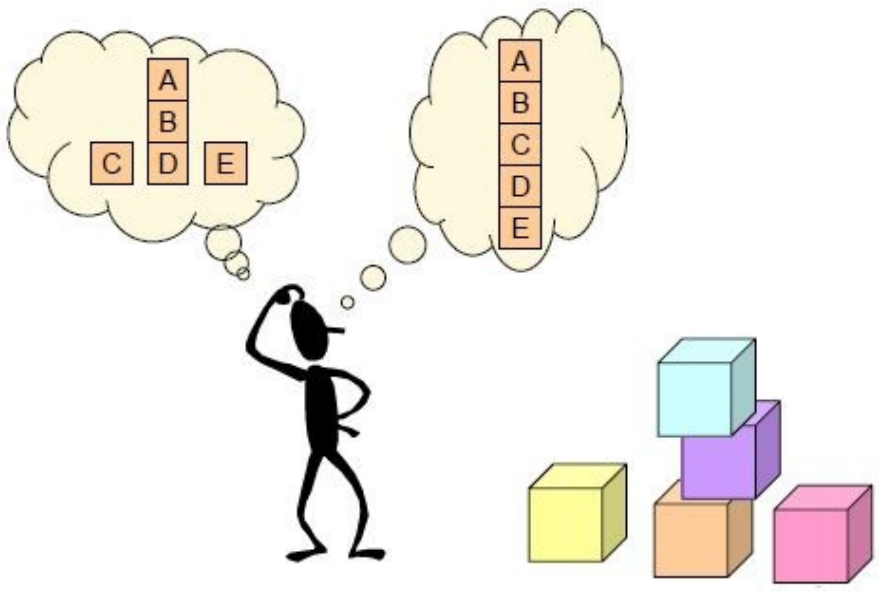
# Estructura de Datos y Algoritmos II.




Algoritmos de Ordenamiento.

Heapsort.

Ing. Mercedes Aguilar Chagoyán.

# Contenido.

- Objetivo.
- Repaso.
- Heapsort.
  - Explicación.
  - Ejemplo.
  - Ejercicio.
- Fuentes.

# Objetivo.

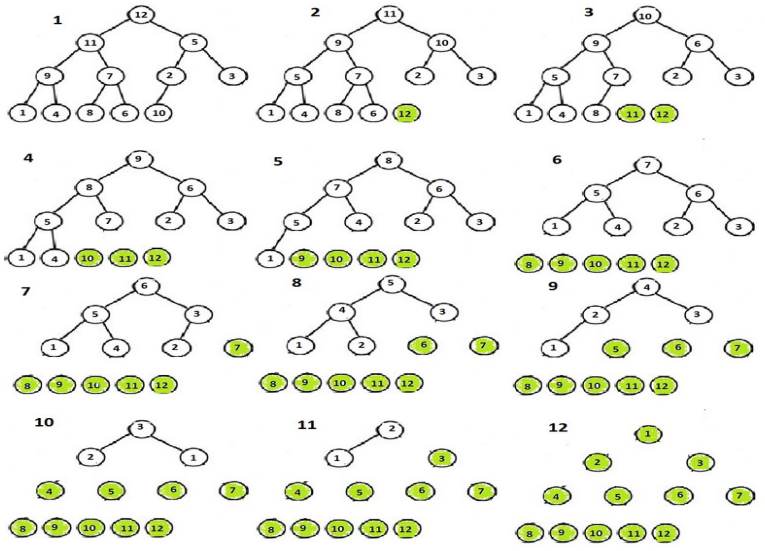
- El alumno aprenderá, comprenderá e implementará todos los conocimientos adquiridos en esta clase y las anteriores.

# Repaso.

- Resuelve el siguiente formulario de google, ten en cuenta que este tendrá solo 5 minutos habilitado para poder ser resuelto, después de ese tiempo el formulario se cerrará y no podrás enviar tus respuestas.
- https://forms.gle/mkaKUM8zKMtLi3nE9

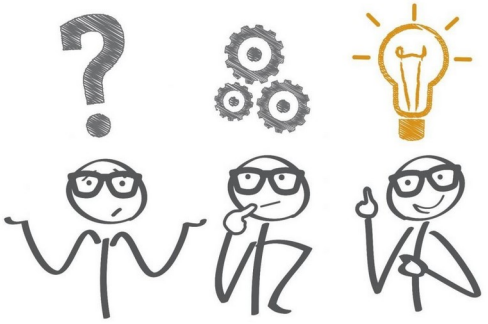
# ¿Qué es HeapSort?

- Es un algoritmo de ordenamiento con complejidad computacional 0(n log n).

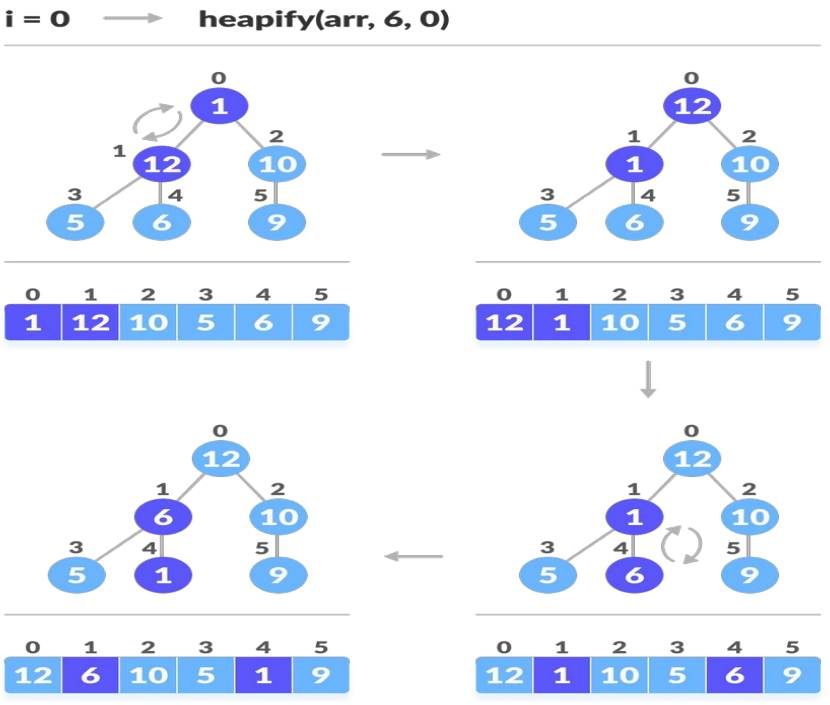


# ¿En qué consiste?

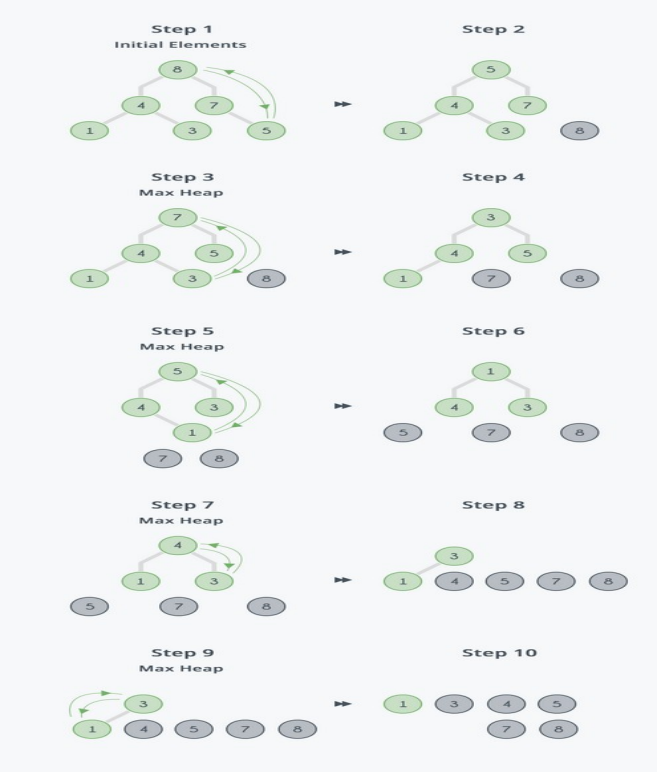
- Almacena todos los elementos del vector a ordenar en un montículo (heap), luego extrae el nodo que queda como un nodo raíz de la cima en sucesivas iteraciones obteniendo el conjunto ordenado.



# ¿Cómo funciona?



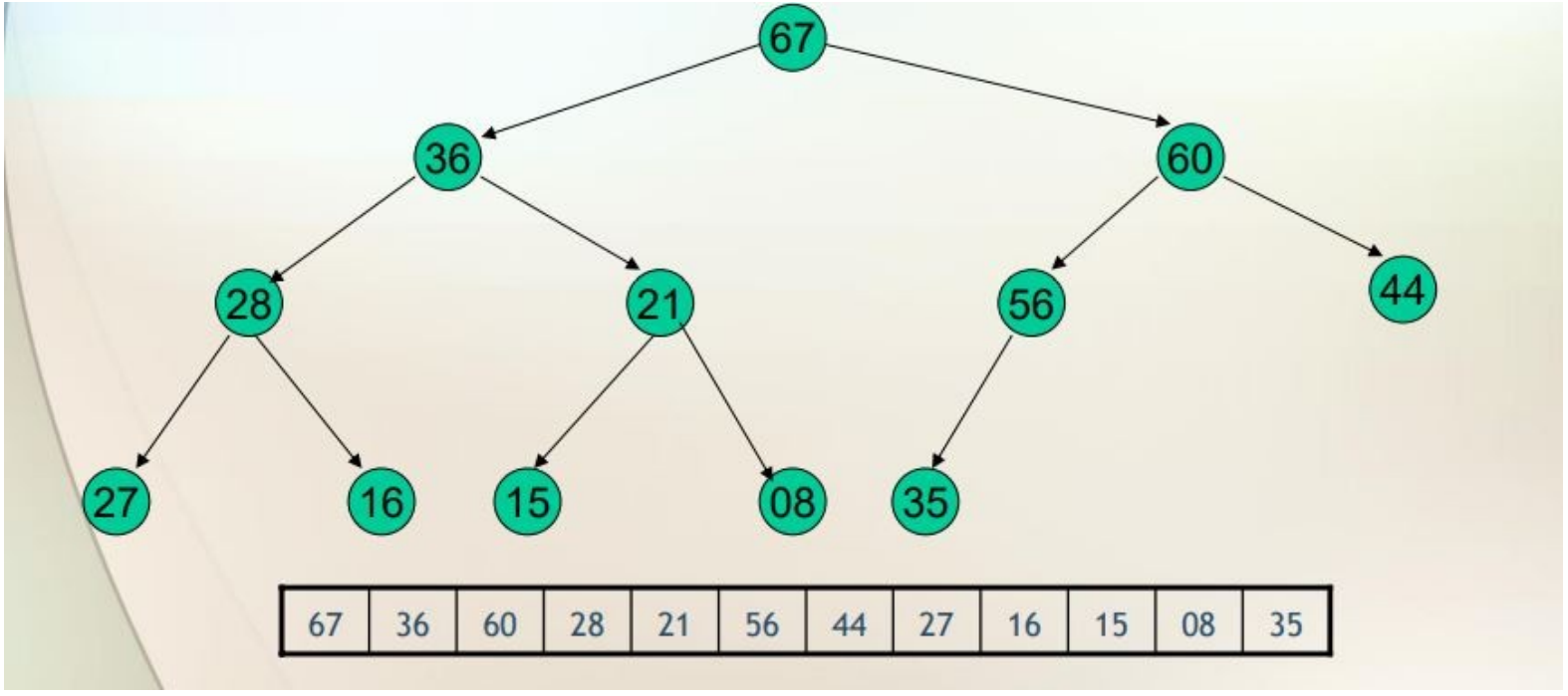
# Ejemplo 1.


Step 1
Initial Elements
Step 2
Step 3
Max Heap
Step 4
Step 5
Max Heap
Step 6
Step 7
Max Heap
Step 8
Step 9
Max Heap
Step 10

# Representación.

1) El nodo *K* se almacena en la posición *k* correspondiente del arreglo.
2) El hijo izquierdo del nodo *k* se almacena en la posición *(2*(K+1))-1.*
3) El hijo derecho del nodo *k* se almacena en la posición *2*(K+1).*

# Representación.



# ¿Cómo insertar un elemento?

- Se inserta el elemento en la primer posición disponible,
- Verificamos el valor y si es mayor al del padre. Si esto se cumple, se efectúa un intercambio. Si no se cumple, entonces el algoritmo se detiene y el elemento queda ubicado en la posición correcta dentro del heap.

# Ejemplo 2.

- 63 en el heap.
  - ***67 36 60 28 21 56 44 27 16 15 08 35 63***
- 63>56
  - ***67 36 60 28 21 63 44 27 16 15 08 35 56***
- 63>60
  - ***67 36 63 28 21 60 44 27 16 15 08 35 36***
- 63>67 no hay intercambio, y queda ubicado el nuevo elemento.

# Eliminación

- Para obtener los elementos en orden, se efectúa eliminando la raíz del heap de forma repetitiva. Los paso que se deben llevar a cabo, son:
  - Reemplazar la raíz con el elemento que ocupa la última posición.
  - Se verifica si el valor es más grande de sus hijos. Si se cumple, entonces se efectúa el intercambio. Si no se cumple, entonces el algoritmo para y y el elemento queda ubicado en su posición.

# Ejemplo.

Montículo: 67 56 60 44 21 28 36 15 35 16 13 08 27 12 07 10

1ra. Eliminación:

Intercambia la raíz 67 con el último elemento 10

A[0]<A[2] (10<60) sí hay intercambio, A[2] es el hijo mayor de A[0]

A[2]<A[6] (10<36) sí hay intercambio, A[6] es el hijo mayor de A[2]

A[6]<A[13] (10<12) sí hay intercambio, A[13] es el hijo mayor de A[6]

60 56 36 44 21 28 12 15 35 16 13 08 27 10 07 67

2da. Eliminación:

Intercambia la raíz 60 con el último elemento 07

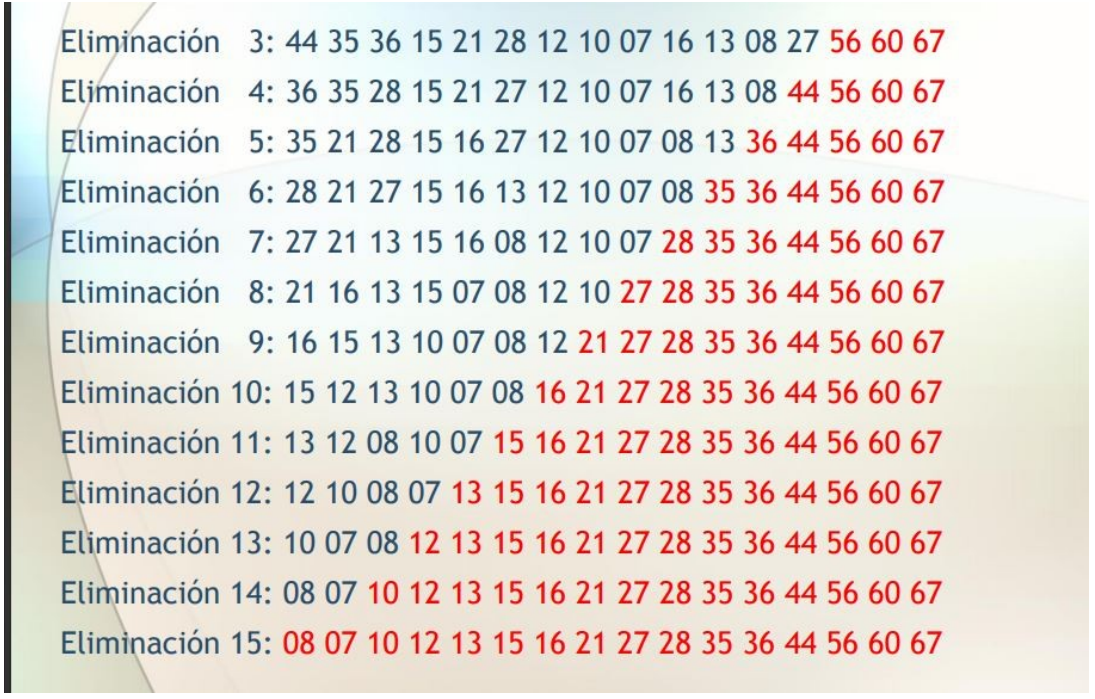
A[0]<A[1] (07<56) sí hay intercambio, A[1] es el hijo mayor de A[0]

A[1]<A[3] (07<44) sí hay intercambio, A[3] es el hijo mayor de A[1]

A[3]<A[8] (07<35) sí hay intercambio, A[8] es el hijo mayor de A[4]

56 44 36 35 21 28 12 15 07 16 13 08 27 10 60 67

# Ejemplo.


Eliminación 3: 44 35 36 15 21 28 12 10 07 16 13 08 27 56 60 67
Eliminación 4: 36 35 28 15 21 27 12 10 07 16 13 08 44 56 60 67
Eliminación 5: 35 21 28 15 16 27 12 10 07 08 13 36 44 56 60 67
Eliminación 6: 28 21 27 15 16 13 12 10 07 08 35 36 44 56 60 67
Eliminación 7: 27 21 13 15 16 08 12 10 07 28 35 36 44 56 60 67
Eliminación 8: 21 16 13 15 07 08 12 10 27 28 35 36 44 56 60 67
Eliminación 9: 16 15 13 10 07 08 12 21 27 28 35 36 44 56 60 67
Eliminación 10: 15 12 13 10 07 08 16 21 27 28 35 36 44 56 60 67
Eliminación 11: 13 12 08 10 07 15 16 21 27 28 35 36 44 56 60 67
Eliminación 12: 12 10 08 07 13 15 16 21 27 28 35 36 44 56 60 67
Eliminación 13: 10 07 08 12 13 15 16 21 27 28 35 36 44 56 60 67
Eliminación 14: 08 07 10 12 13 15 16 21 27 28 35 36 44 56 60 67
Eliminación 15: 08 07 10 12 13 15 16 21 27 28 35 36 44 56 60 67

# Ejercicio.

heap.py

```
def heapsort(alist):
    build_max_heap(alist)
    for i in range(len(alist) - 1, 0, -1):
        alist[0], alist[i] = alist[i], alist[0]
        max_heapify(alist, index=0, size=i)

def parent(i):
    return (i - 1)//2

def left(i):
    return 2*i + 1

def right(i):
    return 2*i + 2

def build_max_heap(alist):
    length = len(alist)
    start = parent(length - 1)
    while start >= 0:
        max_heapify(alist, index=start, size=length)
        start = start - 1

def max_heapify(alist, index, size):
    l = left(index)
    r = right(index)
    if (l < size and alist[l] > alist[index]):
        largest = l
    else:
        largest = index
    if (r < size and alist[r] > alist[largest]):
        largest = r
    if (largest != index):
        alist[largest], alist[index] = alist[index], alist[largest]
        max_heapify(alist, largest, size)


alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
heapsort(alist)
print('Sorted list: ', end='')
print(blist)
```

- Explica el funcionamiento de este ejemplo de HEAPSORT.

# Fuentes.

- Libro:

  Ordenamiento Ejemplos y Teoría.

# Gracias por su atención.