

Actividad 3.2 – Árbol Heap: Implementando una fila priorizada

Push

```
int parent(int i){ return (i - 1) / 2; }

/*Agregue un dato a la fila priorizada*/
void priority_queue::push(int dato){

    datos.push_back(dato); // Se hace un append a la lista de datos
    int index = datos.size() - 1;

    while(index > 0 && datos[parent(index)] < datos[index]){
        // Cambiamos los elementos de posición.
        swap(datos[index], datos[parent(index)]);

        index = parent(index); // la POSICIÓN del padre pasa a ser nodo hijo. Es su posición correcta
    }
}
```

Actividad 3.2

Void push(int dato)

datos.push_back(dato) C_1

index = datos.size() - 1 C_2

divide entre 2

$(i-1)/2$

↓

while index > 0 and datos[parent(index)] < datos[index]

C_3 if-1 swap(datos[index], datos[parent(index)])

index = parent(index) C_5 if-1

$T(n) = C_1 + C_2 + (C_3 \log \xi + C_4 \log(\xi-1) + C_5 \log(\xi-1))$

$T(n) = C_1 + C_2 + C_4 + C_5 + \log \xi + (\log \xi)(C_3 + C_4 + C_5)$

$T(n) = \log(\xi + j) = \boxed{O(\log n)}$

Pop

```
void priority_queue::pop(){  
  
    //cout << heapExtractMax(datos) << endl;  
  
    if (size() < 0){  
        cout << "Heap underflow" << endl;  
        return;  
    }  
  
    if (empty() == true){  
        cout << "La lista ya esta vacia" << endl;  
        return;  
    }  
  
    // El más grande está siempre al inicio.  
    int max = datos[0];  
  
    // el último (menor valor) pasalo al frente para reorganizar todo el árbol y conservar propiedad.  
    datos[0] = datos[datos.size() - 1];  
  
    datos.pop_back(); // ya pasaste el más pequeño al frente. Borra el que está al final que es el mismo valor  
  
    maxHeapify(datos, 0); // reorganizas el árbol  
}
```

5.5 heap/23A

void pop()

if size() < 0
 print "Heap underflow"
 return

if empty() == true
 print "is empty"
 return

max = datos[0]

datos[0] = datos[datos.size()-1]; (2 1

datos.pop_back() (3 1

maxHeapify(datos, 0) (4 log(n)

$T(n) = c_1 + c_2 + c_3 + c_4 \log(n)$

$T(n) = b + a \log n \rightarrow O(\log n)$

Top, Empty & Size

```
/*Regresa el valor del dato que esta con mayor prioridad en la fila priorizada.*/
int priority_queue::top(){

    if (datos.size() >= 1){
        return datos[0];
    }

    cout << "No hay elementos en la lista" << endl;
    return -1;
}

/*Regresa un valor booleano diciendo si la fila priorizada esta vacía o tiene datos.*/
bool priority_queue::empty(){

    return (datos.size() == 0) ? true : false;
}

/*Regresa la cantidad de datos que tiene la fila priorizada*/
int priority_queue::size(){

    return datos.size();
}
```

int top()

if (datos.size() >= 1) C_1

return datos[0] C_2

Print "no hay elementos" C_3

return -1 C_4

$T(n) = C_1 + C_2 + C_3 + C_4 = O(1)$

bool empty()

return (datos.size() == 0) ? true : false; C_1

$T(n) = C_1 = O(1)$

int size()

return datos.size() C_1

$T(n) = C_1 = O(1)$